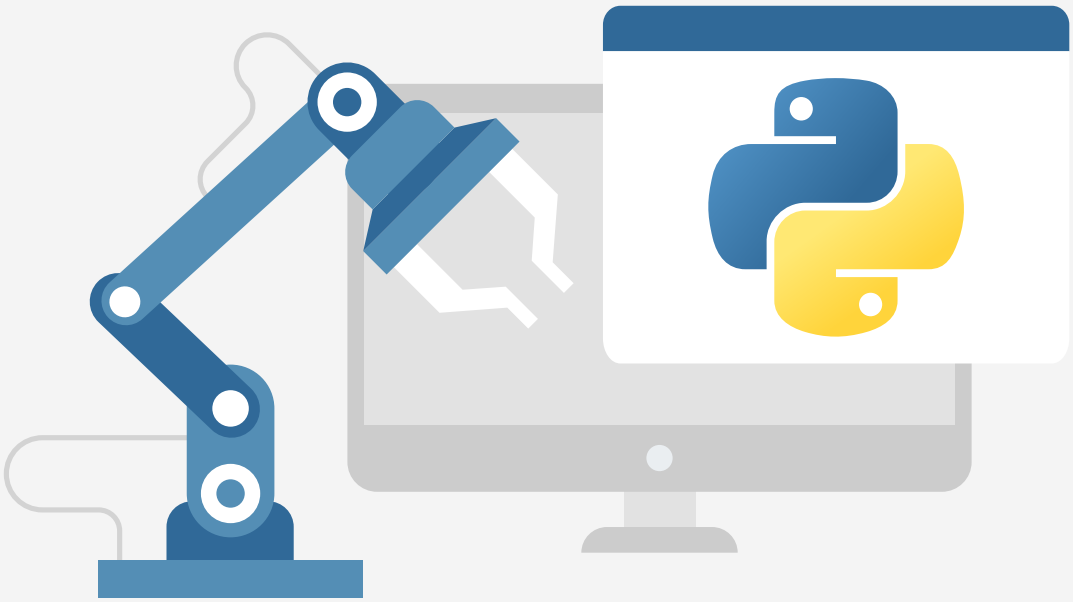


أتمتة المهام المملة عبر بايثون

تأليف

Al Sweigart

ترجمة

عبد اللطيف ايمش

علا عباس

أكاديمية حسوب

أتمتة المهام المملة عبر بايثون

كيفية أتمتة مهامك المكررة بسهولة عبر لغة بايثون

اسم الكتاب: أتمتة المهام المملة عبر بايثون

Book Title: Automate the boring stuff with Python

المؤلف: آل سويغارت

Author: Al Sweigart

المترجم: عبد اللطيف ايمش، علا عباس

Translator: Abdullatif Eymash, Ola Abbas

المحرر: عبد اللطيف ايمش، رقية بورية

Editor: Abdullatif Eymash, Roukia Bouria

تصميم الغلاف: أحمد عماره

Cover Design: Ahmed Umara

سنة النشر: 2024

Publication Year: 2024

رقم الإصدار: 1.0

Edition: 1.0

https://academy.hsoub.com

academy@hsoub.com

عن الناشـر

أُنتج هذا الكتاب برعاية شركة حسوب وأكاديمية حسوب.

تهدف أكاديمية حسوب إلى تعليم البرمجة باللغة العربية وإثراء المحتوى البرمجي العـربي عـبر توفـير دورات برمجة وكتب ودروس عالية الجودة من متخصصين في مجال البرمجة والمجالات التقنية الأخـرى، بالإضـافة إلى توفير قسم للأسئلة والأجوبة للإجابة على أي سؤال يواجه المتعلم خلال رحلته التعليمية لتكون معه وتؤهلـه حـتى دخول سوق العمل.

حسوب شركة تقنية في مهمة لتطوير العالم العـربي. تبـني حسـوب منتجـات تركِّـز على تحسـين مسـتقبل العمل، والتعليم، والتواصل. تدير حسوب أكبر منصتي عمل حر في العالم العربي، مستقل وخمسات ويعمل في فيها فريق شاب وشغوف من مختلف الدول العربية.

المحتويات باختصار

جدول المحتويات

فهرس الأشكال

فهرس الجداول

دورة تطوير التطبيقات
باستخدام لغة بايثون

مميزات الدورة

- بناء معرض أعمال قوي بمشاريع حقيقية
- وصول مدى الحياة لمحتويات الدورة
- تحديثات مستمرة على الدورة مجانًا
- شهادة معتمدة من أكاديمية حسوب
- إرشادات من المدربين على مدار الساعة
- من الصفر دون الحاجة لخبرة مسبقة

اشترك الآن

تمهيد

«لقد أنجزت في ساعتين ما كان يأخذ منا نحن الثلاثة يومين كاملين!»

هذا ما قاله زميلي الذي كان يعمل في متجر بيع الإلكترونيات في بدايات القرن الحالي. إذ كان يتلقى المتجر جداول بيانات بين الحين والآخر، فيها أسعار آلاف المنتجات في المتاجر الأخرى، وكان فريقٌ من الموظفين يطبع تلك الجداول على الورق ويقسمونها بينهم ليعملوا عليها، ثم يضعون ملاحظات على كل المنتجات التي كان يبيعها منافسوهم بسعر أقل، ويأخذ ذلك منهم عادةً عدة أيام.

تدخل أحد معارفي بعد أن رآهم مع الأوراق المبعثرة على الأرض وقال: «يمكنني أن أكتب لكم إن شئتم برنامجًا يؤتمت هذه العملية إن كان لديكم الملف الأصلي».

بعد ساعتين كان قد طور لهم برنامجًا بسيطًا يقرأ أسعار المنافسين من ملف ويوازنها مع سعر المنتج في قاعدة بيانات المتجر ويخبرهم إن كان سعر المنافسين أرخص؛ كان ما يزال رفيقي حديث العهد بالبرمجة وقضى أغلب وقته في البحث في أحد المراجع البرمجية، بينما استغرق تنفيذ البرنامج عدّة ثواني فقط.

هذا هو موطن القوة في البرمجة، فالحاسوب متعدد الإمكانيات ويمكننا استعماله لأداء الكثير والكثير من المهام؛ وللأسف يقضي أشخاصٌ كثر ساعاتٍ من وقتهم بإجراء مهام مكررة غيرَ عالمين بإمكانية أداء الحاسوب لهذه المهام بثوانٍ قليلة إن أعطيناه الأوامر الصحيحة.

1.1 لمن هذا الكتاب؟

تقع البرمجيات في صلب العديد من الأدوات التي نستعملها يوميًا، فتقريبًا كلنا يستعمل الشبكات الاجتماعية للتواصل، ولدينا هواتف ذكية في جيوبنا طوال الوقت، وأغلبية الأعمال المكتبية تتطلب تفاعلًا بشكلٍ أو بآخر مع الحواسيب، ونتيجةً لذلك زادت الحاجة إلى الأشخاص الذين يستطيعون البرمجة ازديادًا عظيمًا

أخيرًا، وظهر عدد كبير من الكتب والمقالات والدورات التي تعد المبتدئين أن يصبحوا مهندسي برمجيات برواتب عالية.

هذا الكتاب ليس لتلكم الفئة، بل لجميع الفئات الأخرى!

لن يحولك هذا الكتب بمفرده إلى مطور برمجيات محترف، لكنك إن كنت تعمل عملًا مكتبيًا أو كنت مديرًا للأنظمة أو كنت أكاديميًا أو كنت تحب أن تمرح مع حاسوبك فهذا الكتاب لك، وستتعلم فيه أساسيات البرمجة لكي تتمكن من أتمتة المهام البسيطة مثل:

- نقل وإعادة تسمية آلاف الملفات وترتيبها ضمن مجلدات.
- تعبئة استمارات مواقع الويب، دون أن تكتبها بيدك كل مرة.
- تنزيل ملفات أو نسخ نصوص أحد المواقع ومعالجها بطريقة ذكية.
- إرسال إشعارات مخصصة لك حين وقوع حدث ما على حاسوبك.
- تحديث أو تنسيق جداول بيانات إكسل.
- التحقق من بريدك الإلكتروني وإرسال ردود جاهزة.

هذه المهام بسيطة، لكنها تأخذ وقتًا طويلًا من وقتنا، وتكون إما بسيطة أو مخصصة جدًا فلا يوجد برنامج متاح لإنجازها، لكن بعد تعلمك لأساسيات البرمجة فيمكنك أن تجعل حاسوبك يقوم بها بدلًا منك.

1.2 الاصطلاحات

هذا الكتاب ليس مرجعًا وإنما دليل للمبتدئين، وهنالك بعض الشيفرات التي يكون أن يكون أسلوب كتابتها غير مستحسن (مثل استخدام المتغيرات العامة)، لكن هذه مفاضلة آثرتها لتكون البرامج أبسط.

لا أركز على جعل الشيفرات في هذا الكتاب من أفضل الشيفرات بنيةً أو شكلًا، ولا أشرح فيه المفاهيم البرمجية المتقدمة مثل البرمجة الكائنية OOP أو المولدات generators لأنها تزيد تعقيد الشيفرة دون فائدة.

قد يشير المبرمجون المخضرمون إلى إمكانية تعديل الشيفرات في الكتاب لتكون أسرع أو أفضل أداءً، لكن هذا الكتاب يركز على أداء المهام وإنشاء البرامج بأقل جهد ممكن.

1.3 ما هي البرمجة؟

تُظهِر الأفلام والمسلسلات المبرمجين على أنهم يكتبون سلاسل طويلة من الأصفار والواحدات على شاشات سوداء ذات نص أخضر، لكن البرمجة واقعيًا أبسط من ذلك بكثير، فهي مجموعة من التعليمات التي نخبر الحاسوب أن يفعلها، مثل إجراء العمليات الحسابية أو تعديل النصوص أو البحث في الملفات أو التواصل مع حواسيب أخرى عبر الإنترنت.

هنالك تعليمات أساسية يستعملها جميع البرامج لبناء التطبيقات، وهذه بعضٌ منها:

- افعل كذا، ثم افعل كذا.
- إذا كان الشرط الفلاني محققًا، فافعل ما يلي، وإلّا فافعل الآتي.
- كرِّر هذا الأمر 27 مرة بالتمام والكمال.
- كرِّر هذا الأمر حتى يصبح الشرط الآتي محققًا.

يمكنك دمج هذه البنى مع بعضها لإنشاء أمور أكثر تعقيدًا، ففيما يلي مثال عن مجموعة من الأوامر البرمجية، التي ندعوها شيفرات مصدرية source code.

المثال الآتي مكتوب بلغة بايثون، وتأتي برمجية بايثون على كل سطر من بداية الملف وتنفذه حتى تصل إلى نهايته:

```
❶ passwordFile = open('SecretPasswordFile.txt')

❷ secretPassword = passwordFile.read()

❸ print('Enter your password.')
  typedPassword = input()

❹ if typedPassword == secretPassword:

  ❺ print('الوصول مسموح')

  ❻ if typedPassword == '12345':

    ❼ print('أنصحك باختيار كلمة مرور أفضل')

  else:

  ❽ print('الوصول ممنوع')
```

قد لا تعرف شيئًا عن البرمجة، لكن من المرجح أن تفهم الغرض من البرنامج السابق بقراءة الشيفرة؛ فبالبداية نفتح الملف `SecretPasswordFile.txt` ❶، ثم نقرأ كلمة المرور التي فيه ❷، ثم نطبع رسالة تطلب من المستخدم كتابة كلمة مروره عبر لوحة المفاتيح ❸، ثم نوازن كلمتَي المرور ❹، فإن كانتا متطابقتين فسيطبع البرنامج العبارة «الوصول مسموح» ❺، وقد يخبرنا البرنامج أن اختيارنا لكلمة المرور ليس جيدًا ❼، وإن كانتا مختلفتين فسيطبع لنا «الوصول ممنوع» ❽.

1.4 ما هي بايثون؟

بايثون هي لغة برمجة، ولغة البرمجة هي مجموعة قواعد لكتابة الشيفرة يجب الالـتزام بهـا ليفهمهـا مـترجم اللغـة، ومـترجم بـايثون Python interpreter هـو برمجيـة تقـرأ الشـيفرات المصـدرية بلغـة بـايثون وتطبـق التعليمات البرمجية فيها.

يمكنك أن تنزل مفسر لغة بايثون مجانًا من موقع اللغة الرسمي python.org، الذي فيه إصدارات لأنظمـة تشغيل ويندوز وماك ولينكس.

إذا كنتَ تتساءل من أين أتى اسم بايثون، فهو من مجموعة كوميدية بريطانية اسمها Monty Python، ولا علاقة للثعابين والأفاعي باسمها.

1.5 ليس من الضروري أن يكون المبرمجون خبراء في الرياضيات

أكثر ما يثير يخيف الراغبين بتعلم البرمجة أنها تستعمل الرياضيات، لكن في الواقـع كـل مـا يحتاجـه أغلبيـة المطورين هو فهم للعمليات الرياضية الأساسية (والتي أضمن لك أنك تعرفها إن كنت تقرأ هذا الكتاب).

أحب تشبيه البرمجة بألغاز سودوكو، فلحل أحد الألغاز يجب أن تستعمل الأرقـام من 1 إلى 9 في كـل سـطر وفي كل عمود، والتي تشكل مربعات صغيرة 3×3 في لوح كبير 9×9.

ستوفر في اللغز بعض الأرقام لتبدأ بها، ويمكنك حل اللغز بالاعتمـاد على اسـتنتاجك لبقيـة الأرقـام اعتمـادًا على الأرقام الأولية. ففي اللغز الموجود في الصورة الآتية يظهر لدينا الرقم 5 في السطر الأول والثـاني وبالتـالي لا يمكن أن يظهر الرقم 5 في هذين السطرين، وإذا ركزنا على المربع في الركن العلوي الأيمن من اللوح فسـنعلم أن الرقم 5 يجب أن يكون في السطر الثالث، لكننا لا نسطيع وضعه على يمين الرقم 6 لأن العمود الأيمن فيه الـرقم 5، لذا يجب أن نضعه على يسار الرقم 6. وبحلنا لأحد الأسطر أو الأعمدة أو المربعات ستكون لدينا أدلـة تسـاعدنا على حل اللغز كله.

- لغز سودوكو
- حل لغز سودوكو

لغز سودوكو (يمين) وحله (يسار)، وعلى الرغم من اعتماد اللغز على الأرقام لكنه على الواقع لا يتضمن الكثـير من الرياضيات.

صحيحٌ أن ألغاز سودكو تحتوي على أرقام كثيرة، لكنك لا تحتاج إلى معرفة معلومات معقـدة في الرياضـيات لحلها، والأمر نفسه ينطبق على البرمجة، فتطوير البرامج يكون بتقسـيمها إلى خطـوات بسـيطة ومفصـلة؛ وحين تنقيح البرنامج (عملية التنقيح debugging هي العثور على الأخطاء وحلها) فستدقق فيما يفعله البرنـامج لتعـثر على سبب العلة، وكلما برمجتَ أكثر زاد مستواك في البرمجة.

1.6 أنت لست كبيرًا في العمر لتبدأ البرمجة

ثاني أمر يخيف المبتدئين في البرمجة أنهم يظنون أنهم كبار في العمر على البدء في البرمجة، وأقرأ تعليقات كثيرة في الويب من أشخاص يظنون أنهم تـأخروا كثـيرًا لأنهم بلغـوا 23 عامًـا! وهـذا ليس سـنًا كبـيرًا لتبـدأ فيـه البرمجة، وكثيرون يتعلمون البرمجة وهم في عمر أكبر من ذلك.

ليس ضروريًا أن تبدأ البرمجة في طفولتك لتصبح مبرمجًا خبيرًا، وتعلم البرمجة الآن أسهل بكثير من تعلمهـا في التسعينيات لوجود كتب ودورات أكثر ومحركات بحث أفضل ومواقع للأسئلة البرمجية وأجوبتهـا. وأصـبحت لغات البرمجة أسهل أيضًا.

من المهم أن تـدرك أن البرمجـة هي مهـارة تنمـو مـع الـوقت، وأن القـدرات البرمجيـة للمطـورين تـزداد بالممارسة، وأنهم لم يولدوا مطورين وإذا لم تكن خبرتك الآن في البرمجة رائعة فإنها ستصبح كذلك مـع الـوقت، ولا يعني ذلك أنك لن تصبح خبيرًا إن تعلمت وطبقت ما تتعلم.

1.7 البرمجة إبداع

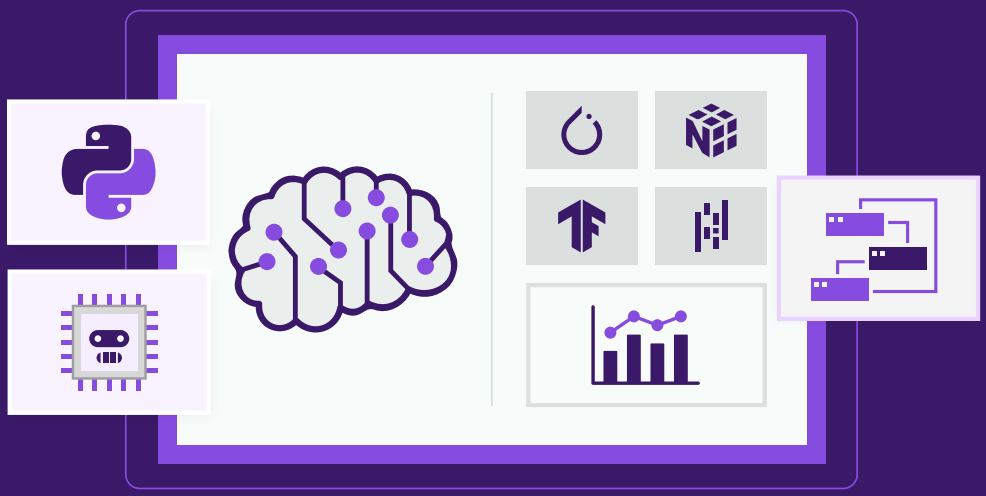
البرمجة هي أمر إبداعي مثل الرسم والكتابة والحياكة وبناء قلاع من الليغـو، ومثـل الرسـم على لوحـة فارغـة فهنالك قيود على كتابة البرامج لكن لها إمكانيات غير محدودة.

الاختلاف الأساسي بين البرمجـة وغيرهـا من الأنشـطة الإبداعيـة هي أن جميـع المـواد الخـادم موجـودة في حاسوبك، فلا حاجة أن تشتري طلاءً أو لوحات رسم أو صوفًا أو قطع ليغو أو قطـع إلكترونيـات، فحاسـوبك الـذي يبلغ عقدًا من العمر مناسب جدًا لكتابة البرامج، وأي بـرامج تكتبهـا يمكنهـا أن تسـتعمل لعـدد غـير محـدود في حواسيب كثيرة وتشاركها مع العالم كله، على عكس قمصان الصوف التي يرتديها شخصٌ واحد فقط.

1.8 المساهمة

إذا كان لديك اقتراح أو تصحيح على النسخة العربية للكتاب أو أي ملاحظة حول المعلومات أو المصـطلحات المستخدمة فيه، فـيرجى إرسـال ذلـك عبـد البريـد الإلكـتروني لأكاديميـة حسـوب academy@hsoub.com، وسيكون أفضل وأسهل لنا إذا ضمنت جزءًا من الجملة التي يظهر لك بها الخطأ مع رقم الصفحة أو القسم الـذي تظهر به.

دورة الذكاء الاصطناعي



مميزات الدورة

- بناء معرض أعمال قوي بمشاريع حقيقية
- وصول مدى الحياة لمحتويات الدورة
- تحديثات مستمرة على الدورة مجانًا
- شهادة معتمدة من أكاديمية حسوب
- إرشادات من المدربين على مدار الساعة
- من الصفر دون الحاجة لخبرة مسبقة

اشترك الآن

1. أساسيات بايثون

تمتلك لغة بـايثون طيفًـا واسـعًا من البـنى البرمجيـة، والـدوال القياسـية، ومـيزات رائعـة لبيئـات التطـوير التفاعلية؛ لكن تجاهل أغلب ما ذكرته آنفًا وابدأ بتعلم ما تحتاج إليه لكتابة برامجك.

لكنك ستحتاج إلى تعلم المفاهيم الأساسية للبرمجة قبل أن تفعـل أي شـيء، وقـد تظن أن هـذه المفـاهيم تافهة أو مملة لكنها أساسية لتتحكم بحاسوبك كما تشاء.

في هذا الفصل أمثلة عديدة أنصحك أن تكتبهـا في الصـدفة التفاعليـة الـتي تسـمى أيضًـا REPL (اختصـار للعبارة Read-Evaluate-Print Loop أي حلقة قراءة-تقدير القيمة-طباعة)، التي تسـمح لـك بتنفيـذ تعليمـات بايثون كل تعليمة على حدة مباشرةً وتظهر لك الناتج.

الصدفة التفاعلية رائعة لتتعلم التعليمات الأساسية في بايثون، لذا أؤكد عليك أن تجربها أثناء قراءتـك لهـذا الفصل، وبالتالي ستتذكر المفاهيم المشروحة فيه تذكرًا أفضل لأنك تراها عمليًا أمامك بدل قراءتها فحسب.

ولكي تتمكن من متابعة ما نقوم به في هذا الفصل وما نقدمه، سنوضح أولًا كيفية تهيئة بيئـة عمـل بـايثون Python، لكي يسهل عليك تطبيق كل المفاهيم التي تصادفها مباشرةً بنفسك.

1.1 تهيئة بيئة العمل في بايثون Python

سنوضح هنا كيفية الوصول إلى لغة بايثون وتهيئة بيئة استخدامها على حاسوبك.

1.1.1 تنزيل وتثبيت بايثون

يمكنك تنزيل بـايثون لنظـام تشـغيل ويـنـدوز أو مـاك أو لينكس مجانًـا من الموقـع الرسـمي للغـة بـايثون، وستعمل معك جميع التطبيقات المشروحة في هذه السلسلة معك.

> **تحذير**: احرص على تنزيل النسخة الثالثة من بايثون (مثل 3.10.4) فالبرامج المشروحة في هذه السلسلة مكتوبة لتعمل على بايثون 3 ولن تعمل كما يجب على بايثون 2.

ستجد في موقع بايثون نسختين إحداهما 64-بت والأخرى 32-بت لكل نظام تشغيل، لذا عليك اختيار الملف الصحيح لتنزله وتثبته. إذا اشتريت حاسوبك بعد عام 2007 فمن المرجح أنك تستعمل نظام 64-بت، وإلا فعليك استخدام نسخة 32-بت.

إذا كنت تستعمل نظام ويندوز فنزِّل مثبت بايثون (الذي ينتهي باللاحقة `.exe`) وشغِّله، بعد ذلك اتبع ما يرشدك إليه المثبِّت على الشاشة. أما على نظام macOS فنزِّل ملف `.pkg` ثم شغله واتبع الإرشادات الموجودة على الشاشة.

أما إذا كنت تستعمل توزيعة أوبنتو (أو ما يشبهها من التوزيعات المبنية على دبيان) فيمكنك تثبيت بايثون من سطر الأوامر بتنفيذ الأمر الآتي:

```
sudo apt install python3 python3-pip idle3
```

1.1.2 تنزيل وتثبيت محرر Mu

صحيحٌ أن مفسِّر بايثون هو البرنامج الذي يشغل شيفرات بايثون التي تكتبها، لكن برمجية Mu editor هي مكان إدخالك للشيفرات، بنفس الطريقة التي تكتب فيها بمعالج النصوص المفضل لديك. يمكنك تنزيل محرر Mu من موقعه.

نزِّل المثبِّت الخاص بنظامك إن كنت تستعمل ويندوز أو ماك، بعدها ابدأ عملية التثبيت بفتح المثبت. أما على لينكس فستحتاج إلى تثبيت محرر Mu كحزمة بايثون، وفي تلك الحالة سيكفيك اتباع الإرشادات الموجودة في الموقع.

1.1.3 تشغيل محرر Mu

بعد انتهاء تثبيت Mu، يمكنك تشغيله:

- في نظام ويندوز بالضغط على قائمة ابدأ ثم البحث عن Mu.
- في نظام MacOS بفتح Finder ثم Applications ثم الضغط على أيقونة mu-editor.
- في لينكس عليك تشغيل الطرفية Terminal ثم كتابة mu-editor.

وحين تشغيلك لمحرر Mu لأول مرة فستظهر لك نافذة تخيّرك بين عدة خيارات وعليك اختيار Python 3، يمكنك في أي وقت تغيير النمط بالضغط على زر Mode في أعلى نافذة المحرر.

1.1.4 تشغيل IDLE

نستعمل في هذه السلسلة Mu كمحرر وصدفة تفاعلية interactive shell، لكنـك تسـتطيع اسـتخدام أي محرر تشاء لكتابة شيفرات بايثون، وتثبت بيئة التطوير والتعليم المدمجـة Integrated Development and Learning Environment (اختصارًا IDLE) مع بايثون تلقائيًا، ويمكنها أن تعمل كمحرر ثانوي لك إن لم يعمـل يثبت عندك محرر Mu. لنشغل IDLE:

- في نظام ويندوز بالضغط على قائمة ابدأ ثم البحث عن IDLE.
- في نظام macOS بفتح Finder ثم Applications ثم الضغط على أيقونة Python.
- في لينكس عليك تشغيل الطرفية Terminal ثم كتابة `idle3`.

1.1.5 الصَدَفة التفاعلية Interactive Shell

عندما تشغل Mu ستظهر أمامك نافذة تحرير الملف، ويمكنك فتح الصَدَفة التفاعلية من خلال الضغط على زر REPL.

الصَدَفة Shell هي برنامج يسمح لك بإعطاء تعليمات للحاسوب، كما تفعل حينما تفتح الطرفيـة Terminal في لينكس أو ماك، أو موجه الأوامر في ويندوز. تسـمح لـك الصـدفة التفاعليـة في بـايثون بإدخـال التعليمـات لينفذها مفسِّر بايثون مباشرةً.

تجد الصدفة التفاعلية في محرر Mu في إطار في الجزء السفلي من النافذة فيها النص الآتي:

```
Jupyter QtConsole 4.7.7
Python 3.6.9 (default, Mar 15 2022, 13:55:28)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.16.3 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

إن كنت تستعمل IDLE، فالصدفة التفاعلية هي أول ما ستراه أمامك، ويفترض أن تكون فارغة عدا الآتي:

```
Python 3.6.9 (default, Mar 15 2022, 13:55:28)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license()" for more
information.
>>>
```

الأسطر التي تبدأ بالعبارة `In [1]:` أو `>>>` تسمى محثًا prompt (أي أنها عبارة «تحثّ» المستخدم على كتابة أمر وراءها). أمثلة هذه السلسلة ستستخدم `>>>` للإشارة إلى محث الصدفة التفاعلية لأنه أكثر شيوعًا. إذا شغلت مفسر بايثون من سطر الأوامر فستجد أمامك المحث `>>>` أيضًا. يستخدم محرر Jupyter Notebook المحث `In [1]:` وشاع بسببه.

لنجرب مثالًا بسيطًا، أدخل السطر الآتي في الصدفة التفاعلية بعد المحث:

```
>>> print('Hello, world!')
```

بعد كتابة السطر السابق فاضغط على زر Enter، ويجب أن يظهر ما يلي في الصدفة:

```
>>> print('Hello, world!')
Hello, world!
```

لقد أعطيت الحاسوب أمرًا ونفذه لك. مبارك!

1.1.6 تثبيت وحدات خارجية

بعض البرامج التي سنكتبها ستستورد وحدات modules، وتأتي بعض هذه الوحدات مع بايثون لكن هنالك وحدات أخرى خارجية طورها مبرمجون ليسوا من فريق تطوير لغة بايثون.

يوضح الملحق 1 بالتفصيل كيفية استخدام البرنامج pip في ويندوز أو pip3 في ماك ولينكس لتثبيت وحدات خارجية.

1.2 أين تجد المساعدة

يميل المطورون إلى التعلم بالبحث في الإنترنت عن إجابات عن أسئلتهم، وهذه الطريقة مختلفة كثيرًا عمّا اعتاد كثيرون على فعله من حضور دروس لمدرس يلقي محاضرةً ويجيب عن أسئلة الطلاب.

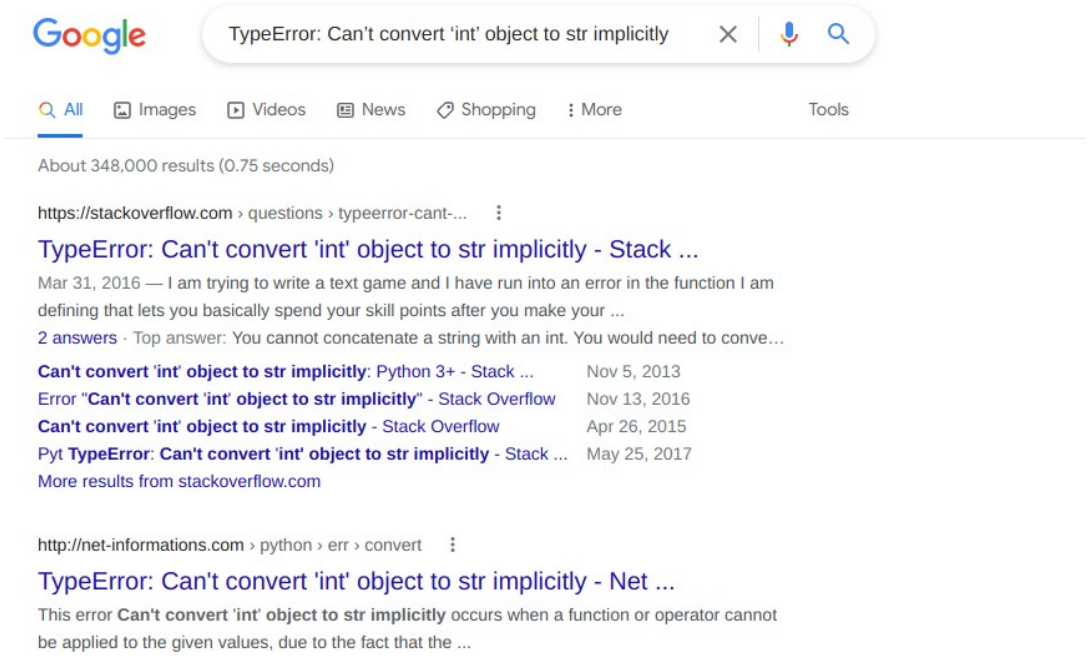
ما يميز استخدام الإنترنت للتعلم أنك ستجد مجتمعات كاملة فيها أشخاص يستطيعون الإجابة عن أسئلتك، بل ومن المرجح أن تكون أسئلتك مجاب عنها مسبقًا وتنتظرك الإجابات عنها لتقرأها.

إذا واجهت رسالة خطأ أو حدث خطبٌ ما أثناء محاولتك تشغيل شيفرتك، فلن تكون أول شخص يواجه هذه المشكلة، وأؤكد لك أن العثور على حلها أسهل مما تظن بكثير.

لنسبب خطأً عمدًا في برنامجنا للتجربة: أدخل `3 + '42'` في الصدفة التفاعلية، لا حاجة إلى شرح ما الذي تفعله هذه الشيفرة الآن لأننا سنتعلمها لاحقًا، لكن سيظهر لك الخطأ الآتي:

```
>>> '42' + 3
❶ Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    '42' + 3
❷ TypeError: Can't convert 'int' object to str implicitly
>>>
```

رسالة الخطأ ❷ تظهر لأن بايثون لا تستطيع فهم التعليمة التي أعطيتها إياها، والجزء الأول ❶ من رسالة الخطأ يبين لنا التعليمة الخطأ ورقم سطرها وما المشكلة التي واجهتها بايثون معها. إذا لم تكن متأكدًا من معنى رسالة الخطأ فابحث عنها في الويب. أدخل "TypeError: Can't convert 'int' object to str implicitly" (بما فيها علامات الاقتباس) في محرك البحث المفضل لديك وستجد عشرات الصفحات التي توضح لك ما هذا الخطأ وما مسبباته كما في الصورة الآتية:



الشكل 1: نتائج بحث جوجل حول رسالة خطأ برمجي

ستجد أن أحدهم كان له نفس سؤالك وأتى مطور من أولي الخبرة وأجابه. اعلم أنه لا يمكن لشخص واحد أن يعرف كل شيء عن البرمجة، لذا يكون البحث عن إجابات الأسئلة التقنية جزءًا من يومك كمطور برمجيات.

1.3 طرح أسئلة برمجية ذكية

إذا لم تجد إجابةً عن سؤالك في الإنترنت، فيمكنك أن تجرب سؤال الخبراء في أحد المنتديات أو المواقع المخصصة مثل Stack Overflow أو مجتمع Learn Programming في Reddit. لكن أبقِ في ذهنك أن هنالك طرائق ذكية لطرح الأسئلة البرمجة لتساعد الآخرين على مساعدتك. فبدايةً احرص على قراءة قسم الأسئلة الشائعة FAQ في تلك المواقع لتعرف الطريقة الصحيحة لطرح السؤال.

حين طرحك لسؤال برمجي فتذكر ما يلي:

- اشرح ما تحاول فعله، وليس ما فعلته. هذا يسمح لمن يريد مساعدته أن يخبرك إن كنت على الطريق الصحيح لحل المشكلة.
- حدد النقطة التي يحدث الخطأ عندها، هل تحدث مثلًا حين بدء تشغيل البرنامج كل مرة أم حين وقوع حدث معين؟ وما هو هذا الحدث الذي يحدث ويسبب ظهور الخطأ.
- انسخ والصق رسالة الخطأ كاملة على مواقع تسمح لك بمشاركة الشيفرات مع الآخرين بسرعة مثل pastebin.com أو gits.github.com، فعبرها تستطيع مشاركة نصوص طويلة دون أن تفقد تنسيق النص، وبعد ذلك ضع رابط URL التابع للشيفرة التي شاركتها في مشاركتك أو سؤالك.
- اشرح ما حاولت فعله لحل المشكلة، وبهذا تخبر الآخرين أنك بذلت جهدًا لمحاولة حل المشكلة بنفسك.
- ضع إصدار بايثون الذي تستخدمه (إذ هنالك اختلافات جوهرية بين الإصدار الثاني من مفسر بايثون والإصدار الثالث). اذكر أيضًا نوع نظام تشغيلك وإصداره.
- إذا حدث الخطأ بعد إجراء تعديل على شيفرتك فاذكر ما هو التعديل الذي أجريته.

أرجو منك أيضًا أن تتبع آداب طرح النقاشات في الإنترنت، فاحرص على سلامة السؤال من الأخطاء الإملائية أو اللغوية، وأن تكون لغته واضحة للجميع، وإذا طرحت سؤالك بالإنكليزية فلا تضعه بأحرف كبيرة، ولا تقدم مطالب غير معقولة وغير منطقية ممن يمد إليك يد المساعدة.

يرى أغلبية مستخدم الحاسوب أن حاسوبهم هو صندوق سحري بدل رؤيته كأداة يمكنهم استخدامها كيفما يشاؤون؛ وبتعلمك البرمجة ستصل إلى أقوى أدوات المتاحة في عالمنا الحالي وهي القدرة الحاسوبية الهائلة المتوافرة أمامك! وتذكر أن تستمتع بوقتك أثناء البرمجة وتعلمها، فالبرمجة ليست كالجراحة العصبية فلا بأس أن تخطئ وتجرب كما تشاء.

نفترض في هذه السلسلة أنك لا تعرف شيئًا عن البرمجة وستتعلم فيه الكثير، لكن إن كانت لديك أسئلة خارج سياقه فتذكر أن تبحث عنها أو أن تطرحها على الخبراء بأسلوب واضح فالبحث عن الإجابات من أهم الأدوات التي عليك احترافها لتعلم البرمجة.

1.4 إدخال التعابير البرمجية في الصدفة التفاعلية

يمكنك تشغيل الصدفة التفاعلية بفتح محرر Mu، الذي يفترض أنك ثبتته وفق التعليمات المذكورة في التمهيد، افتح قائمة ابدأ في ويندوز أو مجلد التطبيقات في ماك وشغِّل محرر Mu، ثم اضغط على زر New ثم احفظ الملف الفارغ باسم مثل `blank.py`، وحينا تحاول تشغيل الملف باستخدام الزر Run أو الضغط على زر F5 في لوحة مفاتيحك فستفتح لك الصدفة التفاعلية في الجزء السفلي من نافذة المحرر، وهنا سترى المحث الآتي أمامك:

```
>>>
```

أدخل `2 + 2` في سطر الأوامر لكي تجري عملية جمع بسيطة، يجب أن تبدو الطرفية التفاعلية كما يلي:

```
>>> 2 + 2
4
>>>
```

نسمي `2 + 2` في بايثون بالتعبير البرمجي expression، وهي أبسط أنواع التعليمات البرمجية في اللغة، وتتألف التعابير البرمجية عادةً من قيم (مثل `2`) وعوامل (operators مثل `+`)، وتقدر قيمتها evaluate إلى قيمة واحدة؛ وهذا يعني أنك تستطيع استخدام التعابير في أي مكان من شيفرات بايثون تستطيع فيه استخدام قيمة ما.

تقدر قيمة التعبير `2 + 2` في المثال السابق إلى قيمة واحدة هي `4`، واعلم أن قيمةً واحدةً لا عامل فيها مثل 4 تعد تعبيرًا برمجيًا في بايثون أيضًا، كما هو واضح هنا:

```
>>> 2
2
```

1.4.1 لا ضير من الأخطاء

سيفشل تشغيل البرامج التي تحتوي على شيفرات لا يفهمها الحاسوب، مما يؤدي إلى إنهيار البرنامج وظهور رسالة خطأ، ورسائل الخطأ لا تسبب مشاكل في حاسوبك، لذا لا تخف من ارتكاب الأخطاء في الشيفرات البرمجية حين تجربتك، وانهيار البرنامج crash يعني أن البرنامج توقف عن التنفيذ بشكل غير متوقع.

إذا أردت أن تتعرف على المزيد من المعلومات حول الخطأ الذي ظهر لك، فعليك البحث في الإنترنت عنه، أو العودة إلى توثيق لغة بايثون.

يمكنك استخدام عدة عوامل في تعابير بايثون، والجدول الآتي يستعرض جميع العوامل الحسابية باللغة.

العامل	العملية	مثال	الناتج
**	القوة (أو الأس)	2 ** 3	8
%	باقي القسمة	22 % 8	6
//	عامل قسمة الأعداد الصحيحة	22 // 8	2
/	القسمة	22 / 8	2.75
*	الضرب	3 * 5	15
-	الطرح	5 - 2	3
+	الجمع	2 + 2	4

الجدول 1: العوامل الرياضية من أعلاها إلى أدناها أولوية وأسبقية

ترتيب أولوية العمليات في بايثون (تسمى أيضًا «أسبقية») مشابهة لأولويتها في الرياضيات، فتقدر قيمة المعامل ** أولًا، ثم تأتي المعاملات * و / و // و % بالترتيب حسب التعبير من اليسار إلى اليمين، ثم يأتي المعاملان + و - بالترتيب أيضًا من اليسار إلى اليمين.

يمكنك استخدام الأقواس ( ) لتغيير الترتيب إن احتجت إلى ذلك.

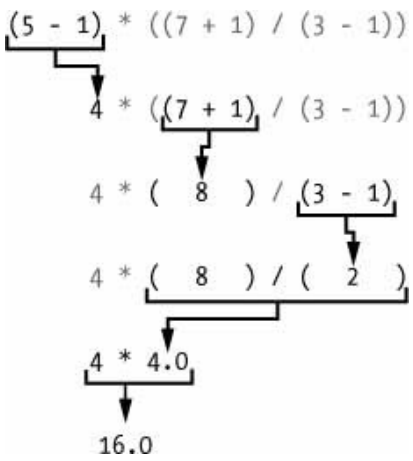
لا تلعب المسافات الفارغة أي معنى بين العوامل في بايثون (عدا المسافة البادئة في أول السطر) لكن من المتعارف عليه استخدام فراغ واحد بينها.

أدخل التعابير الآتية في الصدفة التفاعلية:

```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565878 * 578453
28093077826734
>>> 2 ** 8
256
>>> 23 / 7
3.2857142857142856
>>> 23 // 7
3
>>> 23 % 7
2
```

```
>>> 2  + 2
4
>>> (5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

كل ما عليك فعله هو كتابة التعبير البرمجي، وستتولى بايثون العمليات الحسابية وتعيد لك قيمةً واحدةً هي الناتج، كما هو موضح في الرسم الآتي:



الشكل 2: العمليات الرياضية بايثون

اعلم أن قواعد وضع العوامل والقيم مع بعضها بعضًا لتشكيل التعابير البرمجية من أساس لغة بايثون، ومثَلها كمثل القواعد النحوية التي تساعدنا في التواصل:

- هذه الجملة صحيحة قاعديًا في اللغة العربية.
- جملة صحيحة في اللغة العربية قاعديًا.

لاحظ أن الجملة الثانية صعبة الفهم لأنها لا تتبع القواعد الأساسية لبنية الجملة العربية، وبالمثل إذا أدخلت تعليمة بايثون غير صحيحة فلن تتمكن بايثون من فهمها وسيظهر خطأ SyntaxError كما هو ظاهر في الآتي:

```
>>> 5 +
 File "<stdin>", line 1
  5 +
  ^
SyntaxError: invalid syntax
>>> 42 + 5 + * 2
 File "<stdin>", line 1
```

```
42 + 5 + * 2
     ^
SyntaxError: invalid syntax
```

يمكنك دومًا اختبار صحة تعليمة ما بإدخالها في الصدفة التفاعلية، وأؤكد لك أنها لن تسبب ضررًا بحاسوبك، فأسوأ ما يمكن هو ظهور رسالة خطأ. صدقني حينما أخبرك أن المطورين المحترفين يرون رسائل الخطأ في كل يوم.

1.5 أنواع البيانات العددية والعشرية والنصية

أذكرك أن التعابير هي قيم تجمعها المعاملات، وتكون نتيجتها هي قيمة واحدة دومًا.

نوع البيانات data type هو تصنيف للقيم، فكل قيمة يكون لها نوع بيانات واحد فقط، وسنذكر في الجدول الآتي أكثر أنواع البيانات شيوعًا.

القيمة `-2` و `30` هي أعداد صحيحة integer أو اختصارًا int، والتي تشير إلى أنها أعداد كاملة دون فواصل ويمكن أن تكون موجبة أو سالبة. أما الأرقام مع فاصلة عشرية مثل `3.14` فهي تسمى «الأرقام ذات الفاصلة العائمة floating-point numbers» أو اختصارًا floats. انتبه إلى أن القيمة `42` هي عدد صحيح int بينما `42.0` هي قيمة ذات فاصلة عائمة float.

نوع البيانات	أمثلة
أرقام صحيحة	-2, -1, 0, 1, 2, 3, 4, 5
أرقام ذات فاصلة عائمة	-1.25, -1.0, -0.5, 0.0, 0.5, 1.0, 1.25
سلاسل نصية	'a', 'aa', 'aaa', 'Hello!', '11 cats'

الجدول 2: أنواع البيانات الشائعة

قد تحتوي برامج بايثون على نصوص أيضًا، وتسمى بالسلاسل النصية strings أو اختصارًا strs. احرص على إحاطة السلاسل النصية التي تضعها في برامج بعلامة اقتباس مفردة ' كما في `'مرحبًا'` أو `'مع السلامة'` لكي تعرف بايثون أين تبدأ السلسلة النصية وأين تنتهي.

يمكنك أيضًا إنشاء سلاسل نصية فارغة بكتابة `''`. سنتعلم السلاسل النصية تفصيليًا في الفصل الرابع.

إذا رأيت رسالة الخطأ SyntaxError: EOL while scanning string literal فمن المرجح أنك نسيت علامة الاقتباس المفردة في نهاية السلسلة النصية، كما في المثال الآتي:

```
>>> 'Hello, world!
SyntaxError: EOL while scanning string literal
```

1.6 ضم السلاسل النصية وتكرارها

قد يتغير معنى العوامل اعتمادًا على أنواع بيانات القيم التي تحيط بها، فمثلًا العامل + هو عامل الجمع حينما يحاط بقيمتين عدديتين سواءً كانتا أعدادًا صحيحةً أو ذات فاصلة عائمة؛ لكن حين استخدام العامل + بين قيمتين نصيتين فهو يلمهما على بعضهما ويضمهما ويسمى عامل string concatenation. أدخل ما يلي إلى الصدفة التفاعلية:

```
>>> 'Hello' + 'World'
'HelloWorld'
```

ينتج من التعبير السابق قيمة واحدة وهي سلسلة نصية تجمع النص الموجود في السلسلتين النصيتين المدخلتين؛ لكنك إذا جربت استخدام العامل + على سلسلة نصية ورقم صحيح فلن تعرف بايثون ماذا عليها أن تفعل هنا، وستظهر لك رسالة خطأ:

```
>>> 'Hello' + 42
Traceback (most recent call last):
 File "<pyshell#0>", line 1, in <module>
  'Hello' + 42
TypeError: can only concatenate str (not "int") to str
```

رسالة الخطأ can only concatenate str (not "int") to str تعني أن بايثون تظن أنك تحاول ضم عدد صحيح `42` إلى سلسلة نصية `'Hello'`، فإذا كنت تريد فعل ذلك فعليك أن تحول نوع البيانات العددي إلى نصي يدويًا لأن بايثون لن تفعل ذلك تلقائيًا بالنيابة عنك. سنتعلم تحويل أنواع البيانات في قسم «فهم مكونات تطبيقك الأول» حينما نتحدث عن الدوال `str()` و `int()` و `float()`.

العامل * يضبط عددين صحيحين أو ذوي فاصلة عائلة مع بعضها بعضًا، لكن حين استخدامه مع سلسلة نصية وعدد صحيح فإنه يكررها ويسمى عامل string replication. لترى كيف يعمل، أدخل سلسلةً نصية متبوعةً بعامل * ثم رقم في الصدفة الفاعلية:

```
>>> 'Hello' * 5
'HelloHelloHelloHelloHello'
```

نتيجة التعبير السابق هي قيمة نصية واحدة تُكرَّر فيها السلسلة النصية الأصلية عددًا محددًا من المرات قيمته هي قيمة العدد الصحيح. يفيدنا عامل تكرار السلاسل النصية في بعض الأحيان لكن فائدته لا تقارن بفائدة عامل الضم الذي نستعمله كثيرًا.

يمكن أن يستعمل عامل * مع قيمتين عدديتين لإجراء عملية ضرب، أو مع سلسلة نصية واحدة وعدد صحيح واحد لإجراء عملية تكرار؛ وإلا فستظهر لنا بايثون رسالة خطأ كما يلي:

```
>>> 'Hello' * 'World'
Traceback (most recent call last):
 File "<pyshell#32>", line 1, in <module>
  'Hello' * 'World'
TypeError: can't multiply sequence by non-int of type 'str'
>>> 'Hello' * 5.0
Traceback (most recent call last):
 File "<pyshell#33>", line 1, in <module>
  'Hello' * 5.0
TypeError: can't multiply sequence by non-int of type 'float'
```

من المنطقي ألا تسمح لنا بايثون بهذه التعابير، إذ لا يمكنك ضرب كلمتين ببعضهما، ومن الصعب أن تستعمل عددًا عشريًا لتكرار عبارة نصية.

1.7 تخزين القيم في متغيرات

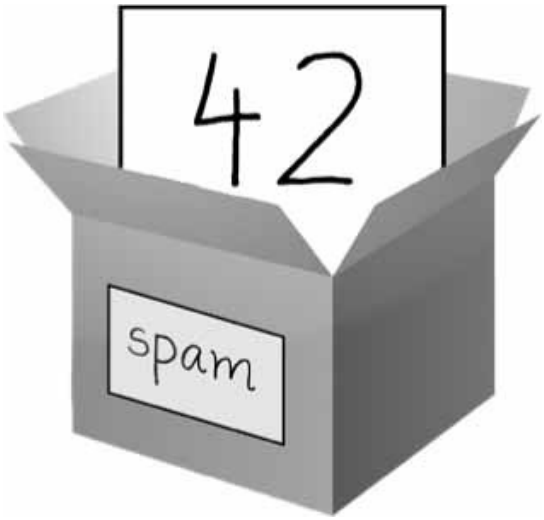
المتغير variable هو ما يشبه الصندوق في ذاكرة الحاسوب الذي يسمح لك بتخزين قيمة واحدة فيه، فإذا أردت تخزين قيمة ناتج أحد التعبيرات البرمجية في برنامجك لاستعمالها لاحقًا فخزنها في متغير.

1.7.1 عبارات الإسناد

ستستخدم عبارة إسناد assignment statement لتخزين القيم في متغيرات، وتتألف عبارة الإسناد من اسم المتغير وعلامة المساواة (وتسمى أيضًا عامل الإسناد) والقيمة التي نرغب بتخزينها.

إذا كتبت مثلًا عبارة الإسناد `spam = 42` فمعناه أن المتغير الذي اسمه `spam` سيخزِّن القيمة `42` داخله.

تخيل أن المتغير هو صندوق له لافتة أو اسم، يمكنك أن تضع القيم داخله كما في الشكل التالي:



الشكل 3: العبارة spam = 42 تخبر البرنامج أن
المتغير يحوي القيمة العددية 42 داخله

على سبيل المثال، أدخِل ما يلي في الصدفة التفاعلية:

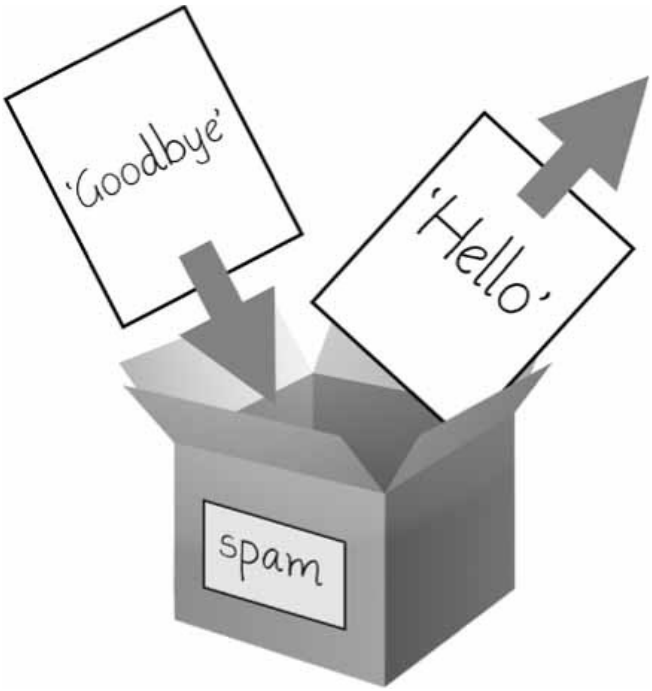
```
❶ >>> spam = 40
  >>> spam
    40
  >>> eggs = 2
❷ >>> spam + eggs
    42
  >>> spam + eggs + spam
    82
❸ >>> spam = spam + 2
  >>> spam
    42
```

سيُهيِّئ المتغير `intialize` أو يُنشَأ في أول مرة تخزن فيه قيمة ❶، وبعد ذلك يمكنك استخدامه في التعابير البرمجية مع غيره من المتغيرات والقيم ❷، وحين إسناد قيمة جديدة إلى المتغير ❸ فستنسى القيمة القديمة، ولهذا السب كان ناتج قيمة المتغير `spam` في نهاية البرنامج هو `42` بدلًا من `40`، وهذا ما يسمى بإعادة كتابة قيمة المتغير overwrite.

أدخل الشيفرة الآتية في الصدقة التفاعلية لتجربة إعادة الكتابة فوق سلسلة نصية:

```
>>> spam = 'Hello'
>>> spam
'Hello'
>>> spam = 'Goodbye'
>>> spam
'Goodbye'
```

وكما في الصندوق في الشكل التالي، فإن الصندوق spam يخزن القيمة `'Hello'` إلى غاية تبديلك لها إلى `'Goodbye'`.



الشكل 4: تنسى القيمة القديمة حين إسناد قيمة جديدة إلى المتغير

1.7.2 أسماء المتغيرات

يصف الاسم الجيد للمتغير البيانات التي يحتويها، فتخيل أنك انتقلت إلى منزل جديد ووضعت لافتات على صناديقك وكتبت عليها «أشياء»، وبهذا لن تعرف ما يحتويه الصندوق إلى أن تنظر داخله. وستجد أن أغلبية أمثلة هذا الكتاب (وتوثيق لغة بايثون) تستعمل أسماء عامة للمتغيرات مثل `spam` و `eggs` و `olive`، لكن احرص على استخدام أسماء واضحة ودالة على محتويات المتغير في برامج لتزيد من مقروئية شيفرتك.

وصحيحٌ أنك تستطيع تسمية متغيراتك بأي اسم، لكن هنالك بعض القيود التي تفرضها لغة بايثون، فالجدول الآتي يحتوي على أمثلة عن أسماء المتغيرات.

يمكنك أن تسمي متغيراتك بأي اسم طالما التزمت بالشروط الثلاثة الآتية:

- أن يكون كلمةً واحدةً دون فراغات.
- أن يستعمل الأرقام والأحرف والشرطة السفلية _ فقط.
- ألا يبدأ برقم.

أسماء متغيرات صالحة	أسماء متغيرات غير صالحة
current_balance	current-balance لا يسمح باستخدام الشرطات
currentBalance	current balance لا يسمح باستخدام الفراغات
account4	4account لا يمكن أن يبدأ برقم
TOTAL_SUM	TOTAL_$UM لا يسمح باستخدام المحارف الخاصة مثل $
hello	'hello' لا يسمح باستخدام محارف خاصة مثل '

الجدول 3: أسماء صالحة وغير صالحة للمتغيرات

أسماء المتغيرات حساسة لحالة الأحرف، وهذا يعني أن `spam` و `SPAM` و `Spam` و `sPaM` هي أربعة متغيرات مختلفة، وصحيحٌ أن الاسم `Spam` صالح لتسمية المتغيرات في بايثون، لكن من المتعارف عليه أن نبدأ متغيراتنا بأحرف صغيرة.

سنستخدم طريقة التسمية «سنام الجمل camel case» في أمثلة هذا الكتاب بدلًا من الشرطات السفلية، أي أن المتغيرات ستكون `lookLikeThis` بدلًا من `look_like_this`. قد يشير المبرمجون الخبراء إلى أن الدليل الرسمي لتنسيق شيفرات بايثون المسمى PEP 8 يقول أن علينا استخدام الشرطات السفلية، لكنني شخصيًا أفضل كتابة المتغيرات بطريقة سنام الجمل، وأقتبس من فقرة «A Foolish Consistency Is the Hobgoblin of Little Minds» في دليل PEP 8 نفسه:

> «من المهم التوافق مع دليل تنسيق الشيفرات، لكن أهم شيء هو معرفة متى يمكنك أن تختلف معه. فبعض الأحيان ليس من المناسب التوافق مع نمط التنسيق، واحكم بنفسك على تلك الحالات».

1.8 برنامجك الأول

صحيحٌ أن الصدفة التفاعلية جيدة لتكتب تعليمات بايثون كل واحدةً على حدة، لكن لكتابة برامج كاملة متكاملة فعليك أن تكتب التعليمات البرمجية في محرر شيفرات.

محررات الشيفرات تشبه كثيرًا محررات النصوص العادية مثل المفكرة أو TextMate لكنها تحتوي على ميزات مخصصة لتسهيل كتابة الشيفرات البرمجية.

لفتح نافذة تحرير ملف جديد في محرر Mu، اضغط على زر New في الصف العلوي من النافذة.

يفترض أن تظهر نافذة فيها مؤشر كتابة ينتظر منك المدخلات، لكن هذه النافذة تختلف عن الصدفة التفاعلية التي كانت تشغل تعليمات بايثون أولًا بأول حين الضغط على زر Enter. إذ يسمح لك محرر الشيفرات بإدخال تعليمات برمجية متعددة ثم حفظ الملف وتشغيل البرنامج. يمكنك معرفة الفرق بينهما بسهولة، إذ تحتوي نافذة الصدفة التفاعلية على المحث >>> فيها، بينما لا يحتويه محرر الشيفرات.

حان الوقت لكتابة أول برنامج لك! حينما تظهر لك نافذة المحرر أدخل فيها الأسطر الآتية:

```
❶ # يرحب البرنامج بالمستخدم ويسأله عن عمره
❷ print('Hello, world!')
  print('What is your name?')     # اسأل عن الاسم
❸ myName = input()
❹ print('It is good to meet you, ' + myName)
❺ print('The length of your name is:')
  print(len(myName))
❻ print('What is your age?')     # اسأل عن العمر
  myAge = input()
  print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

بعد أن تدخل الشيفرة المصدرية السابقة فعليك حفظها لكي لا تكتبها كل مرة تشغل فيها محرر Mu. اضغط على زر Save واكتب اسم الملف `hello.py` ثم اضغط على Save لحفظه.

أنصحك أن تحفظ برامجك التي تكتبها بين الحين والآخر أثناء كتابتها وتطويرها، فلو حدث خلل في حاسوبك أو أغلقت محرر الشيفرات خطأً فلن تخسر ما كتبته، ويمكنك استعمال اختصار الحفظ الذي هو `Ctrl+S` في ويندوز ولينكس و `⌘+S` في نظام ماك.

بعد حفظ الملف يمكنك تشغيل برنامجك، اضغط على زر F5 بلوحة المفاتيح ويجب أن يبدأ تشغيل البرنامج في الصدفة التفاعلية.

تذكر أن عليك الضغط على زر F5 من نافذة المحرر وليس من نافذة الصدفة التفاعلية.

أدخل اسمك حينما يسألك البرنامج عليه، وسيبدو ناتج تنفيذ البرنامج كما يلي:

```
Hello, world!
What is your name?
Abdullatif
It is good to meet you, Abdullatif
The length of your name is:
10
What is your age?
5
You will be 6 in a year.
>>>
```

عندما لا يبقى أي شيفرات لتنفَّذ فسينتهي برنامج بايثون terminate، أي أنه يتوقـف عن العمـل، ويمكننـا القول بتعبير تقني أنه يخرج exit.

يمكنك إغلاق محرر النصوص بالضغط على زر X في أعلى النافـذة، ولإعـادة فتح برنـامج سـابق اضـغط على File ثم Open من القائمـة العلويـة، وسـتظهر نافـذة اختيـار الملفـات الـتي سـتختار منهـا الملـف `hello.py` وتضغط على زر Open؛ يفترض أن ترى أمامك الملف `hello.py` الذي كتبته وحفظته سابقًا.

1.9 فهم مكونات تطبيقك الأول

لنأخذ جولةً سريعة على تعليمات بايثون بعد فتحك لتطبيقك الأول في محرر الشـيفرات، وذلـك بـالنظر إلى ما يفعله كل سطر من الشيفرة.

1.9.1 التعليقات

يسمى السطر الآتي تعليقًا comment:

```
❶ # يرحب البرنامج بالمستخدم ويسأله عن عمره
```

تتجاهل لغة بايثون التعليقات، ويمكنـك اسـتخدامها لكتابـة ملاحظـات أو لتـذكير نفسـك مـا الـذي تحـاول شيفرتك فعله. تكون أي نصوص مذكورة بعد علامة المربع # جزءًا من التعليق.

يضع المطورون في بعض الأحيان رمز # في بداية أحد الأسطر البرمجية لتعطيله مؤقتًا أثناء تجربة البرنـامج، وهذا يسمى بتعليق الشيفرة commenting out، ويمكن أن تستفيد من هذا حينما تحاول معرفـة لم لا يعمـل برنامجك، ثم تزيل رمز # عندما تريد إعادة تفعيل السطر البرمجي.

تتجاهل بايثون أيضًا السطر الفارغ بعد التعليق، ويمكنك إضافة الأسطر الفارغة إلى برنامجك كيفمـا تشـاء، وهذا يسهِّل قراءة برنامجك كما لو كنت تكتب فقرات في كتاب.

1.9.2 الدالة print()

تظهر الدالة `print()` قيمة السلسلة النصية الموجودة بين قوسين على الشاشة:

```
❷ print('Hello, world!')

  print('What is your name?') # اسأل عن الاسم
```

السطر `print('Hello, world!')` يعني اطبع النص الموجود في السلسلة النصية `'Hello, world!'`، فحين تنفيذ بايثون لهذا السطر فأنت تطلب منها أن تستدعي الدالة `print()` وأن تمرِّر pass قيمة السلسلة النصية إلى تلك الدالة. القيمة التي تمرر إلى استدعاء دالة function call تسمى بالوسيط argument. لاحظ أن علامات الاقتباس لا تظهر على الشاشة، فهي إشارة متى تبدأ وتنتهي السلسلة النصية، وليست جزءًا من النص.

> **ملاحظة**: يمكنك أن تستخدم هذه الدالة لتطبع سطرًا فارغًا على الشاشة؛ فكل ما عليك فعله هو استدعاء print() دون أي شيء بين قوسين.

حينما تكتب اسم الدالة ويليه قوسَي الفتح والإغلاق فأنت تستدعي دالةً اسمها هو الاسم الذي يسبق القوسين. سنشرح الدوال تفصيليًا في الفصل الثالث.

1.9.3 الدالة input()

تنتظر الدالة `input()` أن يدخل المستخدم نصًا عبر لوحة المفاتيح ثم يضغط على زر Enter:

```
❸ myName = input()
```

نتيجة استدعاء هذه الدالة هي سلسلة نصية تطابق ما أدخله المستخدم، ثم ستُسند السلسلة النصية الناتجة عن هذا الاستدعاء إلى المتغير `myName`.

يمكنك أن تعدّ عملية استدعاء الدالة `input()` على أنها تعبير برمجي نتيجته هي قيمة السلسلة النصية التي أدخلها المستخدم، فإذا أدخل المستخدم `'Ahmed'` فيمكنك أن تقول أن التعبير البرمجي أصبح أشبه بالتعبير `myName = 'Ahmed'`.

إذا استدعيت الدالة `input()` وظهرت لك رسالة خطأ مثل NameError: name 'Ahmed' is not defined، فهذا يعني أنك تشغل الشيفرة عبر الإصدار الثاني من بايثون وليس الثالث.

1.9.4 طباعة اسم المستخدم

يحتوي الاستدعاء التالي للدالة `print()` على التعبير `'It is good to meet you, ' + myName` بين القوسين:

```
❹ print('It is good to meet you, ' + myName)
```

تذكر أن بايثون تقدر قيمة التعابير البرمجية وتنتهي بقيمة واحدة. فإذا احتوى المتغير `myName` على القيمة `'Ahmed'` في السطر ❸، فستقدر قيمة التعبير السابق إلى `'It is good to meet you, Ahmed'`، ثم ستمرر هذه السلسلة النصية إلى الدالة `print()` التي تطبعها على الشاشة.

1.9.5 الدالة len()

يمكنك أن تمرر سلسلةً نصيةً إلى الدالة `len()` أو متغيرًا يحتوي على سلسلةٍ نصية، وستنتج الدالة عددًا صحيحًا يمثل عدد المحارف في السلسلة النصية:

```
❺ print('The length of your name is:')

  print(len(myName))
```

أجرب إدخال ما يلي إلى الصدفة التفاعلية:

```
>>> len('hello')
5
>>> len('I like to drink good coffee')
27
>>> len('')
0
```

وكما في الأمثلة السابقة، ستكون نتيجة `len(myName)` هي رقم صحيح، ثم ستمرر إلى الدالة `print()` لطباعتها.

تذكر أن الدالة `print()` تسمح لك بطباعة أعداد أو سلاسل نصية، لكن لاحظ رسالة الخطأ التي تظهر حينما تحاول كتابة ما يلي في الصدفة التفاعلية:

```
>>> print('I am ' + 29 + ' years old.')
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
```

```
  print('I am ' + 29 + ' years old.')
TypeError: can only concatenate str (not "int") to str
```

لا يأتي الخطأ من دالة `print()` بل من التعبير الذي حاولت تمريره إلى الدالة، وستحصل على رسالة الخطأ نفسها إذا كتبت التعبير البرمجي بمفرده في الصدفة التفاعلية:

```
>>> 'I am ' + 29 + ' years old.'
Traceback (most recent call last):
 File "<pyshell#7>", line 1, in <module>
  'I am ' + 29 + ' years old.'
TypeError: can only concatenate str (not "int") to str
```

تعطي بايثون خطأً لأن العامل `+` يمكن أن يستعمل لجمع رقمين مع بعضهما أو ضم سلسلتين نصيتين، لكنه لا يستطيع إضافة عدد إلى سلسلة نصية لأن ذلك ليس مسموحًا به قاعديًا في بايثون.

يمكنك حل هذه المشكلة بتحويل الرقم إلى سلسلة نصية، وهذا ما سنتعلمه في القسم التالي.

1.9.6 الدوال str() و int() و float()

إذا أردت ضم عدد صحيح مثل `29` إلى سلسلة نصية لتمريره إلى الدالة `print()` مثلًا، فما ستحتاج إليه هو السلسلة النصية `'29'` التي هي النسخة النصية من العدد `29`.

الدالة `str()` تقبل أن يُمرَّر إليها عدد صحيح ثم تنتج لنا سلسلةً نصيةً تمثل هذا العدد كما يلي:

```
>>> str(29)
'29'
>>> print('I am ' + str(29) + ' years old.')
I am 29 years old.
```

ولأن ناتج التعبير `str(29)` هو `'29'` فسيكون ناتج التعبير `'I am ' + str(29) + ' years old.'` هو `'I am ' + '29' + ' years old.'` الذي بدوره سينتج `'I am 29 years old.'`، وهذه هي القيمة التي ستمرر إلى الدالة `print()`.

الدوال `str()` و `int()` و `float()` ستحول القيم التي تمررها إليها إلى سلسلة نصية وعدد صحيح وعدد ذي فاصلة عائلة على التوالي وبالترتيب.

جرب تحول بعض القيم في الصدفة التفاعلية وانظر ماذا سيحدث:

```
>>> str(0)
'0'
>>> str(-3.14)
'-3.14'
>>> int('42')
42
>>> int('-99')
-99
>>> int(1.25)
1
>>> int(1.99)
3.14
>>> float('3.14')
3.14
>>> float(10)
10.0
```

نستدعي في المثال السابق الدوال `str()` و `int()` و `float()` ونمرر إليهم مجموعةً من القيم لها أنـواع بيانات مختلفة، وسنحصل على نسخة نصية أو عددية من تلك القيم.

تفيد الدالة `str()` حينما يكون لدينا عدد صحيح أو ذو فاصلة عائمة ونريد ضـمه إلى سلسـلة نصـية، بينمـا تفيد الدالة `int()` حينما يكون لدينا رقم مخزن على شكل سلسلة نصية ونريد إجـراء بعض العمليـات الحسـابية عليه، فمثلًا تعيد الدالة `input()` سلسلةً نصيةً دومًا حتى لو أدخل المسـتخدم رقمًـا، فجـرب إدخـال `spam = input()` في الصدفة التفاعلية وكتابة رقم ما ثم طباعة قيمة المتغير `spam`:

```
>>> spam = input()
101
>>> spam
'101'
```

لاحظ أن القيمة المخزنة في المتغير `spam` ليست العدد `101` بل السلسلة النصية `'101'`، فلو أردت إجـراء أي عملية رياضية على القيمة المخزنـة في `spam` فعليـك اسـتخدام الدالـة `int()` للحصـول على عـدد صـحيح. سنعيد تخزين القيمة العددية للمتغير `spam` في المتغير `spam` نفسه:

```
>>> spam = int(spam)
>>> spam
101
```

يمكنك الآن التعامل مع المتغير `spam` كأي عدد صحيح:

```
>>> spam * 10 / 5
202.0
```

لاحظ أنك إذا مررت قيمةً إلى الدالة `int()` لا يمكن أن تحول إلى عدد صحيح، فستظهر لك رسالة خطأ:

```
>>> int('99.99')
Traceback (most recent call last):
 File "<pyshell#18>", line 1, in <module>
  int('99.99')
ValueError: invalid literal for int() with base 10: '99.99'
>>> int('twelve')
Traceback (most recent call last):
 File "<pyshell#19>", line 1, in <module>
  int('twelve')
ValueError: invalid literal for int() with base 10: 'twelve'
```

تفيد أيضًا الدالة `int()` بتحويل عدد عشري إلى عدد صحيح (مع تقريبه إلى أصغر عدد صحيح يمثله):

```
>>> int(7.7)
7
>>> int(7.7) + 1
8
```

لقد استخدمنا الدالتين `int()` و `str()` في آخر ثلاثة أسطر من برنامجك للحصول على قيمة نوع البيانات:

```
❻ print('What is your age?') # اسأل عن أعمارهم
  myAge = input()
  print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

ا. مساواة النصوص والأرقام

صحيحٌ أن القيمة النصية لعددٍ ما تختلف اختلافًا تامًا عن العدد الصحيح أو العدد ذي الفاصلة العائمــة، لكن يمكن أن يساوي العددُ الصحيح العددَ ذا الفاصلة العائمة:

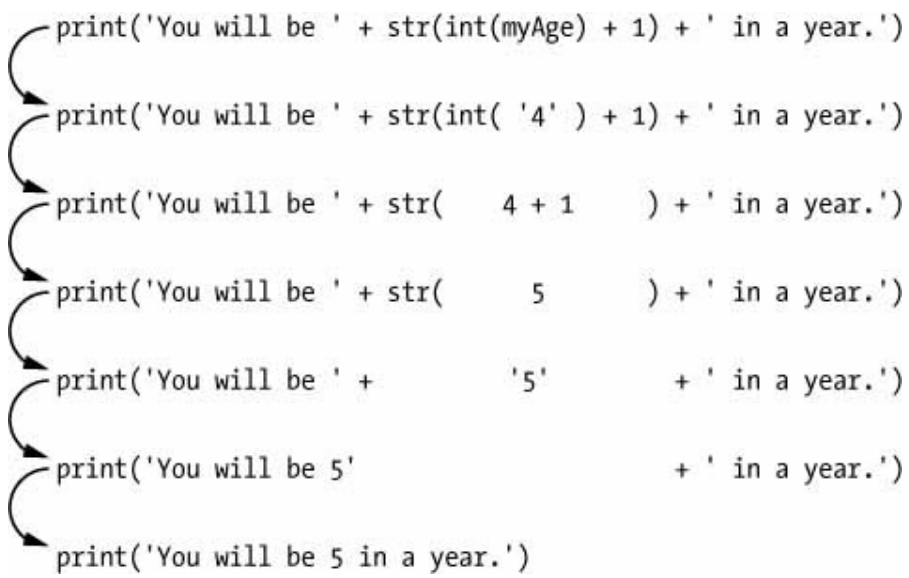
```
>>> 42 == '42'
False
>>> 42 == 42.0
True
>>> 42.0 == 0042.000
True
```

تفعل بايثون ذلك لأن السلاسل النصية هي نصوص بينما الأعداد الصحيحة والأعداد ذات الفاصلة هي أعداد في نهاية المطاف.

المتغير `myAge` يحتوي على القيمة المعادة من `input()`، ولمّا كانت القيمة المعادة من الدالة `input()` هي سلسلة نصية دومًا حتى لو أدخل المستخدم رقمًا، فعلينا استخدام `int(myAge)` لإعادة قيمة عددية صحيحة من السلسلة النصية الموجودة في `myAge`، ثم سنزيد مقدار هذا العدد بواحد في التعبير `int(myAge) +1`.

سنمرر بعد ذلك نتيجة التعبير السابق إلى الدالة `str()` على الشكل `str(int(myAge) +1)`، ثم سنضم القيمة النصية للتعبير السابق مع السلسلتين النصيتين `'You will be '` و `' in a year.'` وذلك للحصول على قيمة نصية واحدة، ثم ستمرر هذه القيمة النصية إلى الدالة `print()` لطباعتها على الشاشة.

لنقل مثلًا أن المستخدم أدخل السلسلة النصية `'4'` قيمةً للمتغير `myAge` عبر الدالة `input()`. ستحول السلسلة النصية `'4'` إلى عدد صحيح `4`، ثم سيضاف `1` إليها فتصبح `5`، ثم تحولها الدالة `str()` إلى سلسلة نصية مجددًا لإضافتها إلى السلاسل النصية الأخرى، وبالتالي ستنتج لدينا النسخة النهائية التي ستطبع على الشاشة، كما هو ظاهر في الشكل التالي:



الشكل 5: النسخة النهائية المطبوعة على الشاشة

1.10 أسئلة للتدريب

1. حدد العوامل والقيم مما يلي كلًا على حدة.
   - *
   - 'hello'
   - -88.8
   - -
   - /
   - +
   - 5
2. أي القيمتين التاليتين متغيرٌ وأيهما سلسلة نصية؟

```
spam
'spam'
```

3. اذكر ثلاثة أنواع للبيانات.
4. مم يتألف التعبير البرمجية؟ وماذا تفعل جميع التعابير البرمجية؟
5. تعرفنا في هذا الفصل على عبارات الإسناد مثل `spam = 10`، ما الفرق بين التعبير البرمجي والعبارة؟
6. ما قيمة المتغير `olive` بعد تنفيذ الشيفرة الآتية:

```
olive = 20
olive + 1
```

7. ماذا سينتج من التعبيرين الآتيين:

```
'spam' + 'spamspam'
'spam' * 3
```

8. لماذا `eggs` هو اسم صالح للمتغيرات بينما `100` ليس كذلك؟
9. ما هي الدوال الثلاث التي يمكن أن تستخدم للحصول على قيمة عددية أو ذات فاصلة أو نصية؟
10. لماذا سيسبب التعبير الآتي خطأً؟ وكيف نحله؟

```
'I have eaten ' + 99 + ' burritos.'
```

1.11 تدريب إضافي

ابحث في الإنترنت في توثيق بايثون عن الدالة `len()`، وستجدها في صفحة بعنوان «Built-in Functions»، اقرأ سريعًا قائمة الدوال المشروحة في الصفحة وابحث عن الدالة `round()`، وجربها في الصدفة التفاعلية وتعرف عليها.

1.12 الخلاصة

يمكنك أن تحسب العمليات الحسابية بالآلة الحاسبة أو تضم السلاسل النصية عبر معالج النصوص في حاسوبك، ويمكنك تكرار السلاسل النصية بسهولة بنسخها ولصقها مرارًا وتكرارًا. لكن التعابير البرمجية -وما تحتويه من مكونات مثل العوامل والقيم واستدعاءات الدوال- هي اللبنة الأساسية لبناء البرامج، وبعد أن تتعلم هذه العناصر الأساسية فستتمكن من إعطاء أوامر لبايثون لتنفذ لك أمورًا معقدة على مجموعة كبير من البيانات.

من المهم أن تتذكر من هذا الفصل أنواع العوامل المختلفة (`+` و `-` و `*` و `/` و `//` و `%` و `**` للعوامل الحسابية، و `+` و `*` للسلاسل النصية) وأنواع البيانات المختلفة (الأعداد الصحيحة integers والأعداد ذات الفاصلة العائمة floating-point والسلاسل النصية string).

شرحنا أيضًا بعض الدوال، مثل `print()` و `input()` التي تتعامل مع النصوص البسيطة لطباعتها على الشاشة أو لإدخالها من لوحة المفاتيح، واستعملنا الدالة `len()` للحصول على القيمة العددية لسلسلة نصية، وساعدتنا الدوال `str()` و `int()` و `float()` في تحويل القيم المُمرَّرة إليها إلى سلاسل نصية أو أعداد صحيحة أو أعداد ذات فاصلة على التوالي.

سنتعلم في الفصل القادم كيفية إخبار بايثون بمتى تنفذ الشيفرة ومتى تتجاوزها بذكاء، وكيفية تكرار جزء من الشيفرة اعتمادًا على شرط محدد، وهذا ما نسميه «بنى التحكم» مما يسمح لنا بكتابة برامج تتصرف تصرفات ذكية.

دورة تطوير التطبيقات
باستخدام لغة بايثون

احترف البرمجة وتطوير التطبيقات مع أكاديمية حسوب
والتحق بسوق العمل فور انتهائك من الدورة

التحق بالدورة الآن

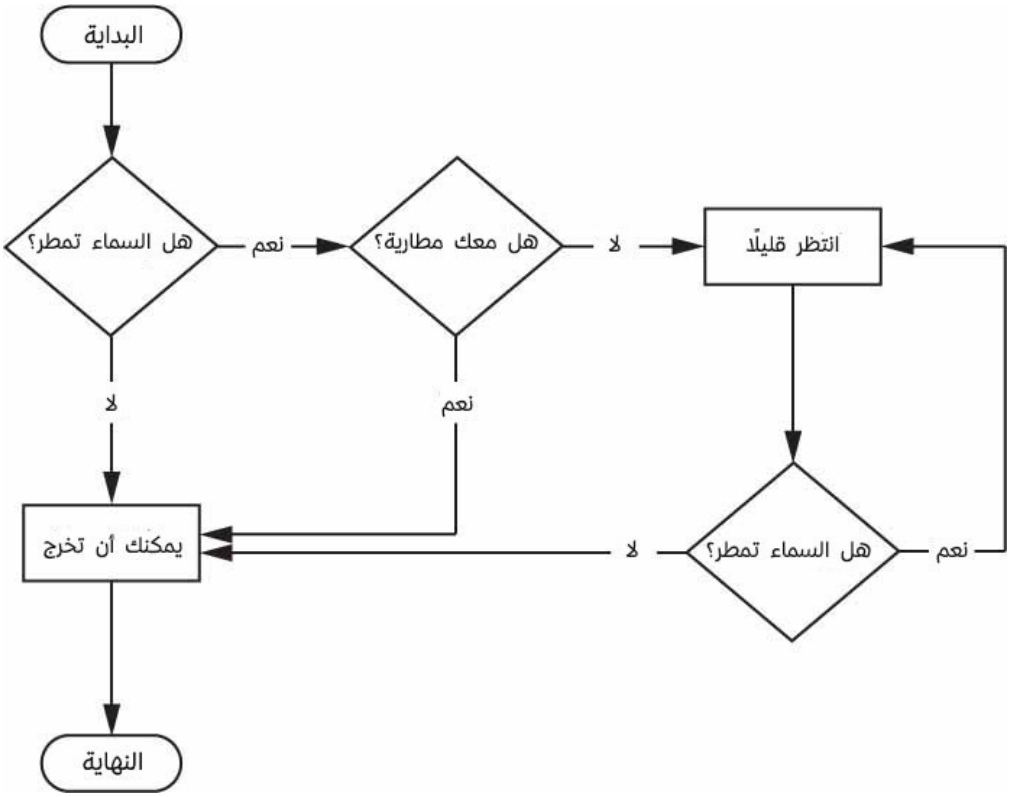
2. بنى التحكم في بايثون

أصبحت تعرف أن هنالك تعليمات وأن البرامج تتألف من سلسلة من التعليمات، لكن قوة البرمجة الحقيقية لا تقع في تنفيذ مجموعة من التعليمات واحدةً تلو الأخرى وكأنك تتبضع قائمة التسوق. فاعتمادًا على نتيجة بعض التعبيرات البرمجية يستطيع البرنامج أن يتخطى تنفيذ مجموعة من التعليمات أو تكررها أو يختار بعضها لتنفيذه.

من النادر أن تجد برامج تنفذ من أول سطر حتى آخر سطر بالترتيب، لذا تكون هنالك حاجة إلى بنى التحكم flow control لتقرير ما هي التعليمات البرمجية التي ستنفذ ووفق أي شروط.

يمكن بسهولة تحويل بنى التحكم البرمجية إلى رموز في مخططات التدفق flowcharts التي نرسمها، لذا سأوفر لك نسخة منها تحاكي الشيفرات التي نكبها.

الشكل التالي يظهر مخططًا تدفقيًا يساعد في اتخاذ القرار عمّا سنفعله إذا كان الجو ماطرًا. تتبع الأسهم من البداية إلى النهاية:



الشكل 6: مخطط تدفقي يخبرك ما تفعل إن كانت السماء تمطر

من الشائع أن يكون في المخططات التدفقية أكثر من طريق من البداية إلى النهاية، والأمر سيان بالنسبة إلى شيفرات تطبيقات الحاسوب.

تُمثَّل نقاط التفرع في المخططات التدفقية باستخدام شكل المعيّن diamond، في حين تُمثَّل بقية الخطوات بمستطيلات عادية، وتكون البداية والنهاية على شكل مستطيل بحواف مدورة.

قبل أن تتعلم عن عبارات بنى التحكم، فعليك أن تعرف كيفية تمثيل خيارات «نعم» و «لا»، بعدها ستحتاج إلى فهم كيفية كتابة نقاط التفرع بلغة بايثون؛ ولهذا الغرض سنحتاج إلى تعلم القيم المنطقية أو البوليانية Boolean وعوامل المقارنة والعوامل المنطقية.

2.1 القيم المنطقية Boolean

يكون لأنواع البيانات التي تعلمناها سابقًا من سلاسل نصية وأرقام صحيحة وأرقام ذات فاصلة، عدد لا متناهي من القيم التي يمكن أن تأخذها. لكن للقيم المنطقية أو البوليانية تملك نوعين من القيم فقط: `True` و`False` (نقول عنها أنها قيم بوليانية Boolean نسبةً إلى العالم الرياضي جورج بولي)، وحين إدخالها في شيفرة بايثون فستلاحظ عدم وجود علامات الاقتباس التي تحيط بالسلاسل النصية، وتبدأ دومًا بالحرف الكبير T أو F وتكون بقية الكلمة بأحرف صغيرة.

أدخل ما يلي في الصدفة التفاعلية، لاحظ الأخطاء التي ستظهر لك في بعض التعليمات:

```
❶ >>> spam = True
  >>> spam
  True
❷ >>> true
  Traceback (most recent call last):
    File "<pyshell#2>", line 1, in <module>
      true
  NameError: name 'true' is not defined
❸ >>> True = 2 + 2
  SyntaxError: can't assign to keyword
```

وكأي قيمة أخرى، يمكن للقيم المنطقية أن تستعمل في التعابير البرمجية ويمكن أن تخـزن في المتغـيرات ❶، وإن لم تستعمل حالة الأحرف الصحيحة ❷ أو جربت استخدام `True` أو `False` لأسماء المتغيرات ❸ فستظهر لك رسالة خطأ.

2.2 عوامل المقارنة

تقارن عوامل المقارنة comparison operators بين قيمتين وتكون النتيجة هي قيمـة منطقيـة بوليانيـة واحدة، الجدول الآتي يستعرض عوامل المقارنة:

العامل	المعنى
==	يساوي
=!	لا يساوي
>	أصغر
<	أكبر
=>	أصغر أو يساوي
=<	أكبر أو يساوي

الجدول 4: عوامل المقارنة

تكون نتيجة استخدام هذه العوامل هي `True` أو `False` اعتمادًا على القيم التي تعطيها لها.

لنجرب الآن بعض تلك العوامل ولنبدأ بالعاملين == و !=:

```
>>> 42 == 42
True
>>> 42 == 99
False
>>> 2 != 3
True
>>> 2 != 2
False
```

وكما قد تتوقع، ستكون نتيجة عامل «يساوي» == هي True حينما تساوت القيمتـان على يمينـه ويسـاره، وعامل «لا يساوي» ستكون نتيجته True حينما تختلف القيمتان على يمينه ويساره.

يمكننا استخدام العاملين == و != على أي نوع من أنواع البيانات:

```
>>> 'hello' == 'hello'
True
>>> 'hello' == 'Hello'
False
>>> 'dog' != 'cat'
True
>>> True == True
True
>>> True != False
True
>>> 42 == 42.0
True
>>> 42 == '42'
False
```

لاحظ أن القيم العددية سواءً كانت صحيحة `int` أو ذات فاصلة `float` لا تتساوي مع القيمة النصية. فالتعبير `42 == '42'` ❶ نتيجته هي `Flase` لأن بايثون تعدّ الرقم `42` مختلفًا عن السلسلة النصية `'42'`.

أما المعاملات `<` و `>` و `<=` و `>=` في لا تعمل إلا مع القيم العددية:

```
>>> 42 < 100
True
>>> 42 > 100
False
>>> 42 < 42
False
>>> eggCount = 42
❶ >>> eggCount <= 42
True
>>> myAge = 29
❷ >>> myAge >= 10
True
```

2.2.1 الفرق بين عامل `=` و `==`

قد تلاحظ أن عامل المساواة `==` يحتوي على إشارتي يساوي، بينما عامل الإسناد فيه إشارة يساوي واحدة. ومن السهل الخلط بينهما بالنسبة إلى المبتدئين، لذا تذكر أن:

- عامل المساواة `==` يتأكد إن كانت القيمتان عن يمينه ويساره متساويتين.
- عامل الإسناد `=` يضع التي على اليمين في المتغير الذي على اليسار.

قد يساعدك في التذكر أن عامل المساواة `==` فيه حرفان، مثل معامل عدم المساواة `!=` تمامًا.

ستستخدم عوامل المقارنة كثيرًا لمقارنة قيمة أحد المتغيرات مع قيمة أخرى، كما في `eggCount <= 42` ❶ و `myAge >= 10` ❷، فلو كنتَ تعرف قيمة المتغير كيف ستكون قبل أن تشغل برنامجك فلا حاجة إلى المقارنة كلها، فليس من المنطقي أن تكتب `'dog' != 'cat'` في شيفرتك بل تكتب `True` مباشرةً. سترى أمثلة كثيرة عن ذلك أثناء تعلمك لبنى التحكم.

2.3 العوامل المنطقية البوليانية

تستخدم العوامـل المنطقيـة الثلاثـة and و or و not لمقارنـة القيم المنطقيـة، وكمـا في عوامـل المقارنـة ستكون نتيجة هذه التعابير هي قيمة منطقية، ولنبدأ شرح هذه العوامل بالتفصيل بدءًا من العامل and و or.

2.3.1 العوامل المنطقية الثنائية

يعمل العاملان and و or على قيمتين أو تعبيرين منطقيين، لهذا يسميان بالعوامـل المنطقيـة الثنائيـة binary Boolean operators.

ينتج العامل nad القيمة True إذا كانت كلا القيمتان المنطقيتان تساوي True، وإلا فالنتيجة هي False.
أدخل التعابير البرمجية الآتية في الصدفة التفاعلية لترى أثر هذا العامل عمليًا:

```
>>> True and True
True
>>> True and False
False
```

جداول الحقيقة truth table هو جدول في الجـبر المنطقي البوليـاني يوضـح نـاتج كـل شـكل من أشـكال التعابير المنطقية.

جدول الحقيقة الآتي يوضح ناتج كل عملية ممكنة مع العامل and:

التعبير	النتيجة
True and True	True
True and False	False
False and True	False
False and False	False

الجدول 5: جدول الحقيقة للعامل and

وفي المقابل تكون نتيجة العامل or هي True إذا كان أحد القيمتين المنطقيتين يساوي True.

أما إذا كانت كلتاهما Flase فنتيجة التعبير هي False:

```
>>> False or True
True
>>> False or False
False
```

يمكنك أن تعرف ناتج كل تعبير ممكن مع العامل or من جدول الحقيقة الخاص به.

التعبير	النتيجة
True or True	True
True or False	True
False or True	True
False or False	False

الجدول 6: جدول الحقيقة للعامل or

2.3.2 العامل not

وعلى النقيض من العاملين and و or، يستخدم العامل not على قيمة أو تعبير منطقي واحد، ولهذا هو عامل أحادي unary operator. ما يفعله العامل not هو عكس القيمة المنطقية الحالية:

```
>>> not True
False
❶ >>> not not not not True
True
```

وكما في نفي النفي في حديثنا العادي، يمكننا أن نجعل العامل not متشعبًا ❶، لكن لا يوجد سبب لفعل ذلك في البرامج العملية. الجدول الآتي هو جدول الحقيقة للعامل not:

التعبير	النتيجة
not True	False
not False	True

الجدول 7: جدول الحقيقة للعامل not

2.4 المزج بين العوامل المنطقية وعوامل المقارنة

لما كانت نتيجة استخدام عوامل المقارنة هي قيمة منطقية، فيمكننا استخدامها في تعابير برمجية مع العوامل المنطقية.

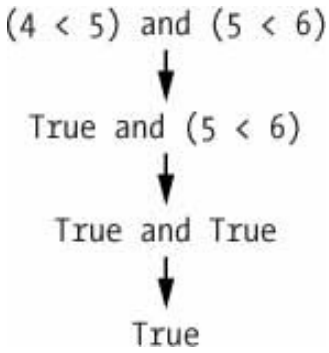
تذكر أن العوامل المنطقية and و or و not هي عوامل منطقية لأنها تعمل على القيم المنطقية True وFalse؛ بينما التعابير التي تحتوي على عوامل مقارنة مثل 4 < 5 ليست قيمًا منطقية بحد ذاتها لكنها تعابير تُنتِج قيمًا منطقية.

جرب إدخال بعض التعابير المنطقية التي فيها عوامل مقارنة في الصدفة التفاعلية:

```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 < 6)
False
>>> (1 == 2) or (2 == 2)
True
```

سيقدر الحاسوب قيمة التعبير على اليسار أولًا، ثم قيمة التعبير الذي على اليمين، ثم بعد أن يعـرف مـا هي القيمة المنطقية لكلٍ منها فسيقدر قيمة التعبير البرمجي كاملًا ويخرج قيمة منطقية وحيدة.

يمكنك أن تتخيل أن عملية تقدير قيمة التعبير البرمجي `(4 < 5) and (5 < 6)` تشبه ما يلي:



الشكل 7: تقدير قيمة التعبير البرمجي

يمكنك أن تستعمل أكثر من عامل منطقي في تعبير واحد، بالإضافة إلى عوامل المقارنة:

```
>>> 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
True
```

سيكون للعوامل المنطقية ترتيب لتقدير القيمة كمـا في العوامـل الرياضـية، فبعـد تقـدير قيمـة العمليـات الحسابية وعمليات المقارنة، ستعمل بايثون على عوامل `not` أولًا، ثم عوامل `and` ثم عوامل `or`.

2.5 عناصر بنى التحكم

تبدأ عبارات بنى التحكم بجزء يسمى عادةً بالشرط condition ويكون متبوعًا دومًا بكتلة برمجية اشتراطية clause، وقبل أن نتعلم عن بنى التحكم في بايثون فسنشرح ما هو الشرط وما هي الكتلة البرمجية.

2.5.1 الشروط

تعدّ جميع التعابير المنطقية البوليانية التي رأيتها حتى الآن شروطًا conditions، فالشروط هي تعابير برمجية، لكنها تطلق في سياق الحديث عن بنى التحكم. يكون للشروط دومًا نتيجة منطقية إما `True` أو `False`، وتقرر بنى التحكم ما الذي يجب فعله اعتمادًا على الشرط إن كان `True` أو `False`، وجميع بنى التحكم تقريبًا تستخدم الشروط.

2.5.2 الكتل البرمجة

تجمِّع أسطر الشيفرات البرمجية في بايثون على شكل كتل blocks، ويمكنك أن تعرف متى تبدأ الكتلة ومتى تنتهي من المسافة البادئة indent للأسطر البرمجية. وهنالك ثلاث قواعد للكتل البرمجية في بايثون:

- تبدأ الكتلة البرمجية حين زيادة المسافة البادئة.
- يمكن أن تحتوي الكتل البرمجية على كتل أخرى.
- تنتهي الكتل حينما تقل المسافة البادئة إلى الصفر أو إلى المسافة البادئة للكتلة الأب.

من الأسهل فهم الكتل البرمجية بالنظر إلى بعض الشيفرات التي لها مسافة بادئة، لذا لنعثر على الكتل البرمجية في البرنامج البسيط الآتي:

```
name = 'Ahmed'
password = 'swordfish'
if name == 'Ahmed':
    ❶ print('Hello, Ahmed')
    if password == 'swordfish':
        ❷ print('Access granted.')
    else:
        ❸ print('Wrong password.')
```

تبدأ أول كتلة من الشيفرات ❶ في السطر `print('Hello, Ahmed')` وتضم إليها جميع الأسطر البرمجية التي تليها، وداخل هذه الكتلة هنالك كتلة أخرى ❷ التي تحتوي سطرًا واحدًا داخلها فيه `print('Access granted.')`، أما الكتلة الثالثة ❸ والأخيرة ففيها `print('Wrong password.')`.

2.5.3 تنفيذ البرنامج

تبدأ بايثون بتنفيذ مثالنا السابق `hello.py` من أول البرنامج حتى نهايته سطرًا بسطر، وعملية تنفيذ البرنامج (التي تسمى program execution أو اختصارًا execution) هي اصطلاح يشير إلى التعليمة البرمجية التي يجري تنفيذها حاليًا، فلو كانت شيفرة برنامجك مطبوعةً على ورقة وتشير بإصبعك إلى السطر الذي يجري تنفيذه حاليًا فتشير إصبعك هنا إلى خط سير تنفيذ البرنامج.

لا تنفذ جميع البرامج من الأعلى إلى الأسفل، فلو كنت تستعمل إصبعك من أجل اتباع خط سير أحد البرامج التي فيها بنى تحكم فسترى أنك تنتقل من مكانٍ إلى آخر في الشيفرة المصدرية، وأنك قد تتخطى أجزاء من الشيفرة كليًا.

2.6 عبارات بنى التحكم

حان الوقت الآن لنتحدث عن أهم جزء من بنى التحكم: عبارات بنى التحكم نفسها. تمثل العبارات في مخططات التدفق -مثل التي رأيتها في الشكل 6- على شكل معيّن، وتشير إلى القرارات التي يتخذها برنامجك.

2.6.1 عبارة if

أكثر نوع شائع من بنى التحكم هو العبارة الشرطية `if`، ففيها ستُنفَّذ الكتلة البرمجية التي تلي `if` إذا كان الشرط محققًا أي `True`، وسيتخطاها البرنامج إن لم يكن الشرط محققًا أي `False`.

فإذا أردنا قراءة عبارة `if` البرمجة باللغة العربية فما علينا سوى قول: «إذا كان الشرط محققًا، فنفذ الكتلة البرمجية الآتية».

تتألف عبارة `if` في بايثون مما يلي:

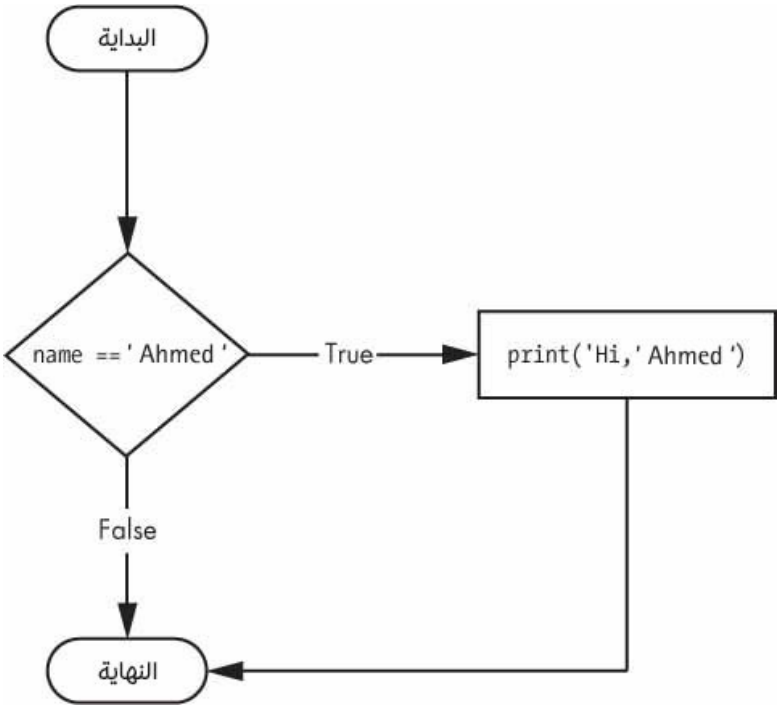
- الكلمة المفتاحية `if`
- الشرط، وهو التعبير الذي تكون نتيجته هي `True` أو `False`
- نقطتان رأسيتان `:`
- كتلة برمجية تلي ما سبق، تكون فيها الأسطر مسبوقة بمسافة بادئة

لنقل مثلًا أنك تريد كتابة شيفرة تتحقق إن كان اسم المستخدم هـو Ahmed، وعلى فـرض أننـا قـد أسـندنا سابقًا قيمةً ما إلى المتغير name:

```
if name == 'Ahmed':
  print('Hi, Ahmed.')
```

تنتهي جميع عبارات بنى التحكم بنقطتين رأسيتين : متبوعة بكتلة برمجية، والكتلة البرمجية في مثالنـا هي التي تحتوي على print('Hi, Ahmed.').

يوضح الشكل التالي المخطط التدفقي للشيفرة السابقة:



الشكل 8: المخطط التدفقي

2.6.2 عبارات else

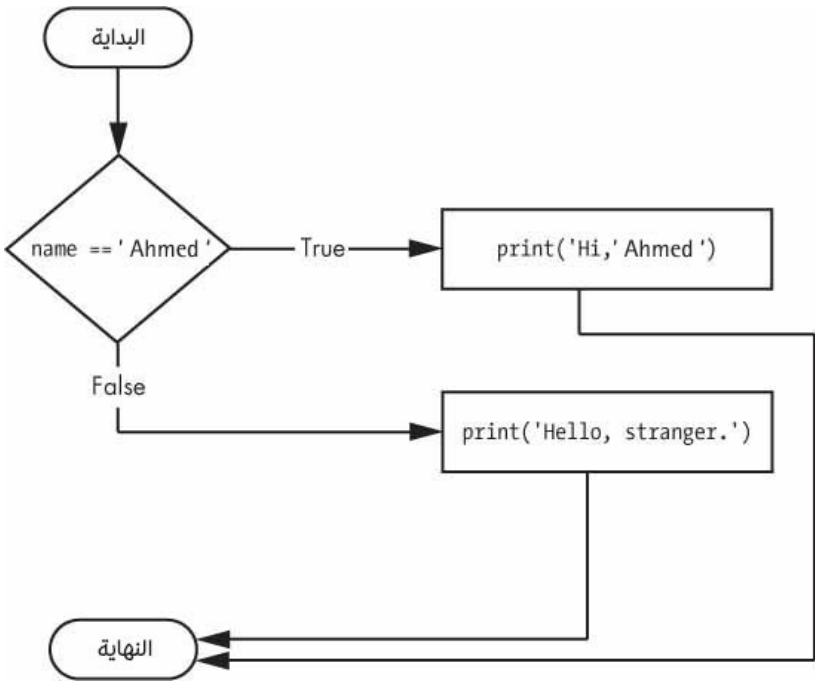
يمكن اختياريًا أن يأتي بعد كتلة if العبارة else، وتنفذ كتلة else في حال كان شرط عبارة if غير محقق False. أي بالعربية يمكننا أن نقول «إذا كان الشرط محققًا، فنفذ الكتلة البرمجية الآتية، وإلا فنفذ هذه الكتلة».

لا تملك عبارة else شرطًا، وتتألف else مما يلي:

- الكلمة المفتاحية else
- نقطتان رأسيتان :
- كتلة برمجية تلي ما سبق، تكون فيها الأسطر مسبوقة بمسافة بادئة

وبالعودة إلى مثالنا السابق للترحيب بالمستخدم، فسنضيف العبارة `else` لطباعة تحية مختلفة لأي شخص ليس اسمه أحمد:

```
if name == 'Ahmed':
  print('Hi, Ahmed.')
else:
  print('Hello, stranger.')
```



الشكل 9: يبين المخطط التدفقي للبرنامج السابق

2.6.3 عبارات elif

تعرفنا سابقًا على عبارة `if` و `else` التي يجب أن تنفذ إحداهما، لكن ماذا لو كنّا نريد وجود أكثر من احتمال أو أكثر من شرط؟ تعمل العبارة `elif` كأنها «وإلا إذا كان كذا» `else if`، وتأتي بعد عبارة `if` أو `elif` أخرى.

توفر عبارة `elif` شرطًا بديلًا يمكن التحقق مما إذا كان محققًا إن كانت الشروط التي تسبقه غير محققة.

تتألف عبارة `elif` في بايثون مما يلي:

- الكلمة المفتاحية `elif`
- الشرط، وهو التعبير الذي تكون نتيجته هي `True` أو `False`
- نقطتان رأسيتان `:`

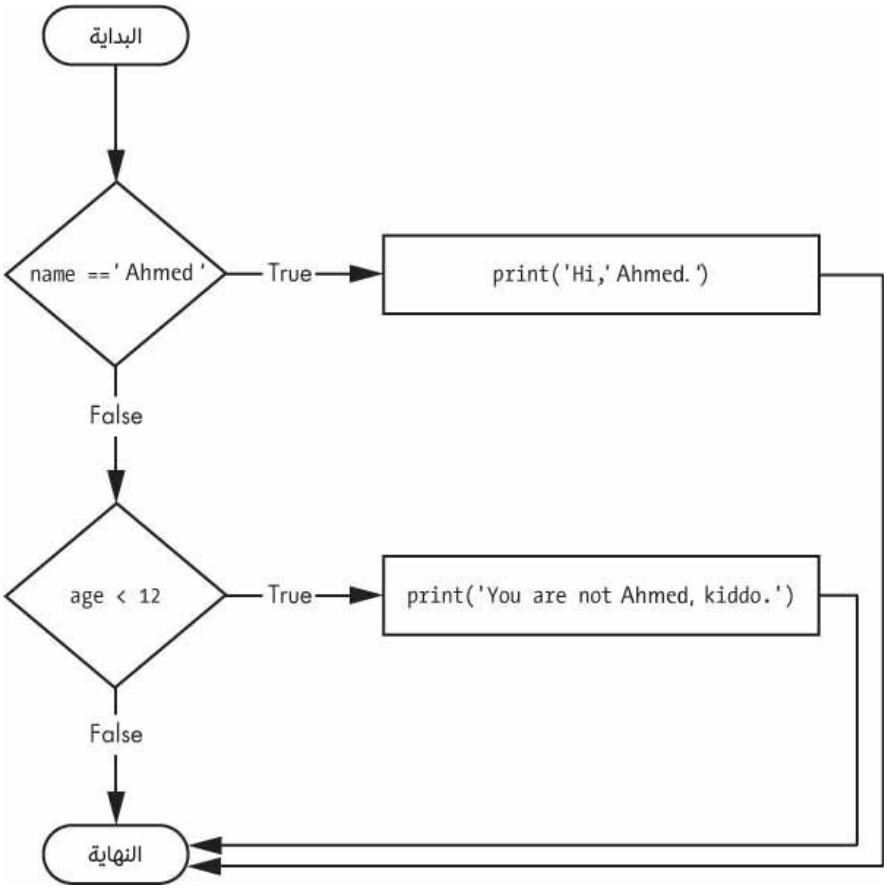
- كتلة برمجية تلي ما سبق، تكون فيها الأسطر مسبوقة بمسافة بادئة

لنضف عبارة `elif` إلى برنامجنا الذي نتحقق فيه من اسم المستخدم:

```
if name == 'Ahmed':
  print('Hi, Ahmed.')
elif age < 12:
  print('You are not Ahmed, kiddo.')
```

سنتحقق هذه المرة من عمر المستخدم، فإن كان عمره أقل من 12 فسيقول له «أنت لست أحمد يا غلام!».

يمكننا تمثيل البرنامج بمخطط التدفق الآتي:



الشكل 10: مخطط التدفق للعبارة elif

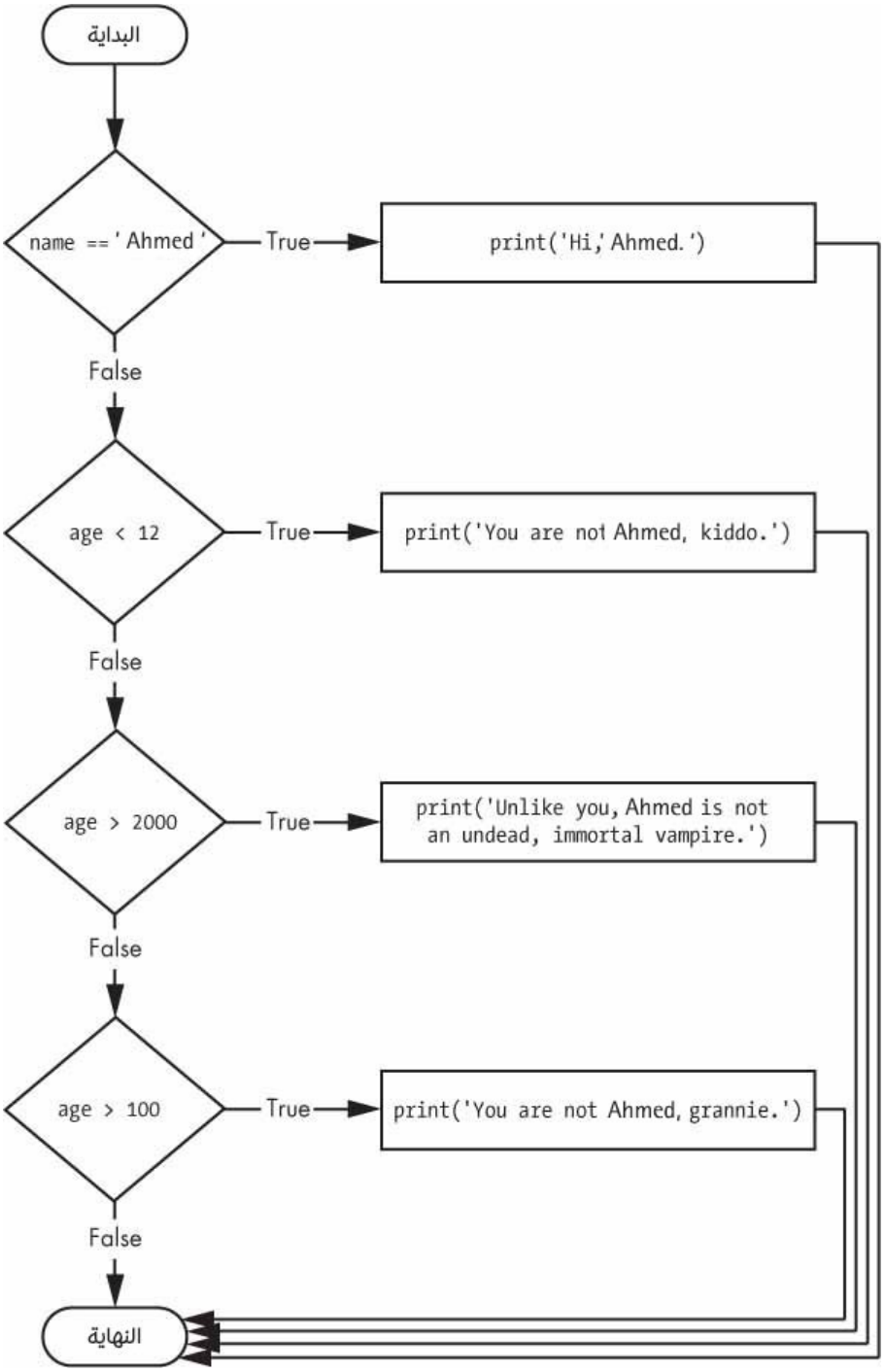
ستقفَّذ الكتلة البرمجية التي تلي `elif` إن كان عمر المستخدم `age < 12` وكان الشرط `name == 'Ahmed'` غير محقق `False`. لكن إن كان كلا الشرطين غير محقق فسيتجاوز البرنامج تنفيذ الكتلتين البرمجيتين، وليس من الضروري أي ينفذ أحد الكتل البرمجية، فقد تنفذ عبارة واحدة أو لا تنفذ أي عبارة.

بعد أن يتحقق شرط إحدى العبارات الشرطية فسيتجاوز البرنامج بقية عبارات `elif` كلها.

على سبيل المثال، أنشِئ الملف `vampire.py` وضع فيه الشيفرة الآتية:

```
age = 3000
if name == Ahmed:
  print('Hi, Ahmed.')
elif age < 12:
  print('You are not Ahmed, kiddo.')
elif age > 2000:
  print('Unlike you, Ahmed is not an vampire.')
elif age > 100:
  print('You are not Ahmed, grannie.')
```

أضفنا هنا عبارتَي `elif` لبرنامج الترحيب بالمستخدم، وأضفنا جوابين مختلفين بناءً على العمر age. يظهـر المخطط التدفقي الآتي سير عمل البرنامج:



الشكل 11: المخطط التدفقي لعبارات elif متعددة في برنامج vampire.py

لترتيب عبارات `elif` أهمية كبيرة، وللتعرف على أهمية ترتيبها فلنحاول إضافة علّة لبرنامجنا.

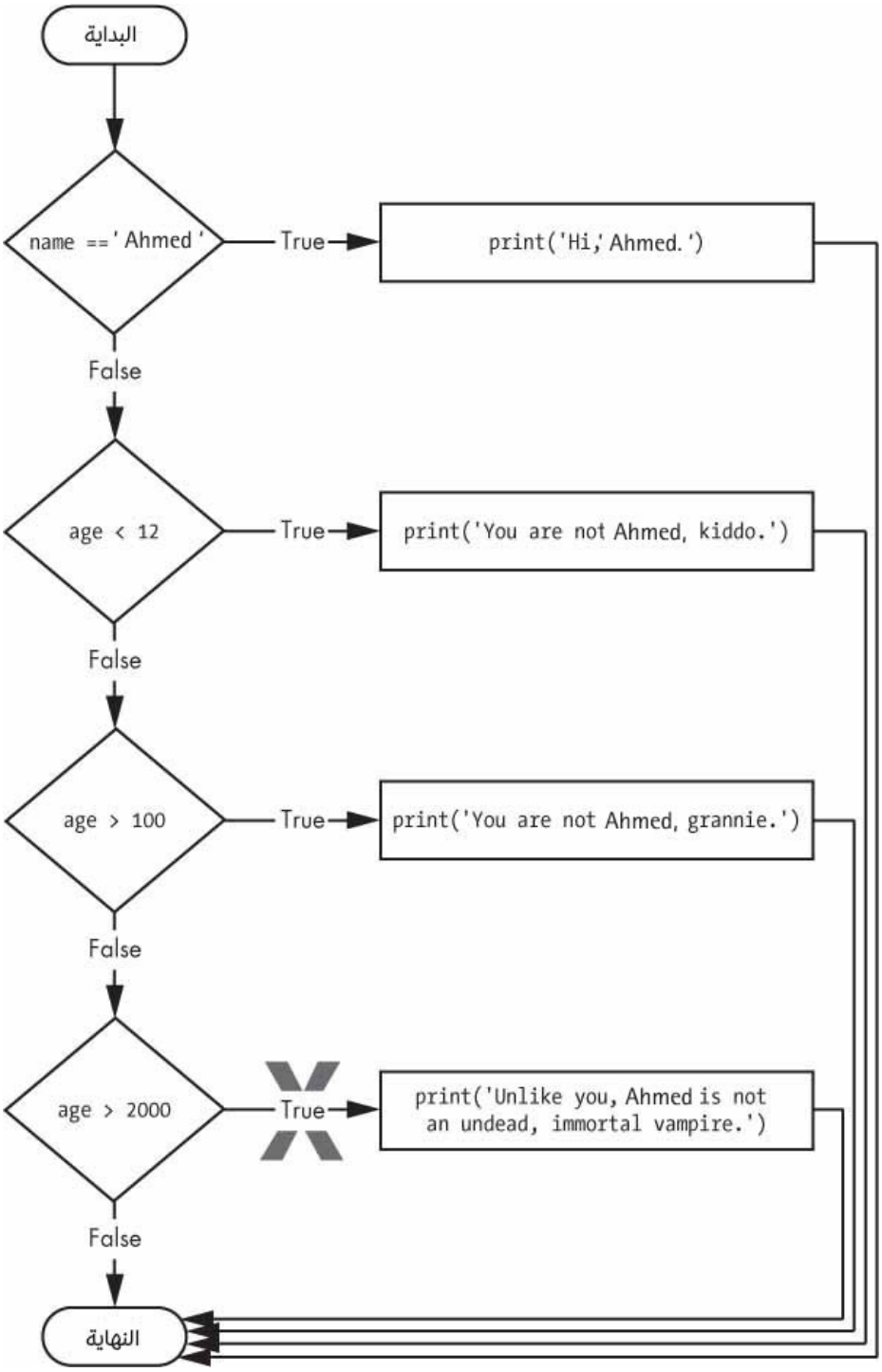
تذكر أن البرنامج سيتخطى عبارات `elif` بعد تحقيق شرط إحداها، لذا إذا غيرنا ترتيب الشيفرة إلى مـا يلي وحفظناه باسم `vampire2.py`:

```
name = 'Abdullatif'
age = 3000
if name == 'Ahmed':
  print('Hi, Ahmed.')
elif age < 12:
  print('You are not Ahmed, kiddo.')
❶ elif age > 100:
  print('You are not Ahmed, grannie.')
elif age > 2000:
  print('Unlike you, Ahmed is not an undead, vampire.')
```

إذا جربنا البرنامج وكانت قيمة المتغير `age` تساوي 3000 مثلًا، فقد تتوقع أن برنـامج سـيطبع العبـارة الـتي تقول أن عمره أكثر من 2000 سنة، لكن ولمّا كانت الشرط `age > 100` محققًا `True` (وذلـك لأن 3000 أكـبر من 100 بالفعل) ❶، فستعرض عبارة الترحيب بالعجوز وستخطى تنفيذ البرنامج جميـع عبـارات `elif` الأخـرى. لذا من المهم الانتباه إلى ترتيب عبارات `elif`.

المخطط التدفقي الآتي يظهر سير تنفيذ الشيفرة السابقة.

لاحظ كيف جرى تبديل المعين الذي يحتوي على `age > 100` و `age > 2000`.



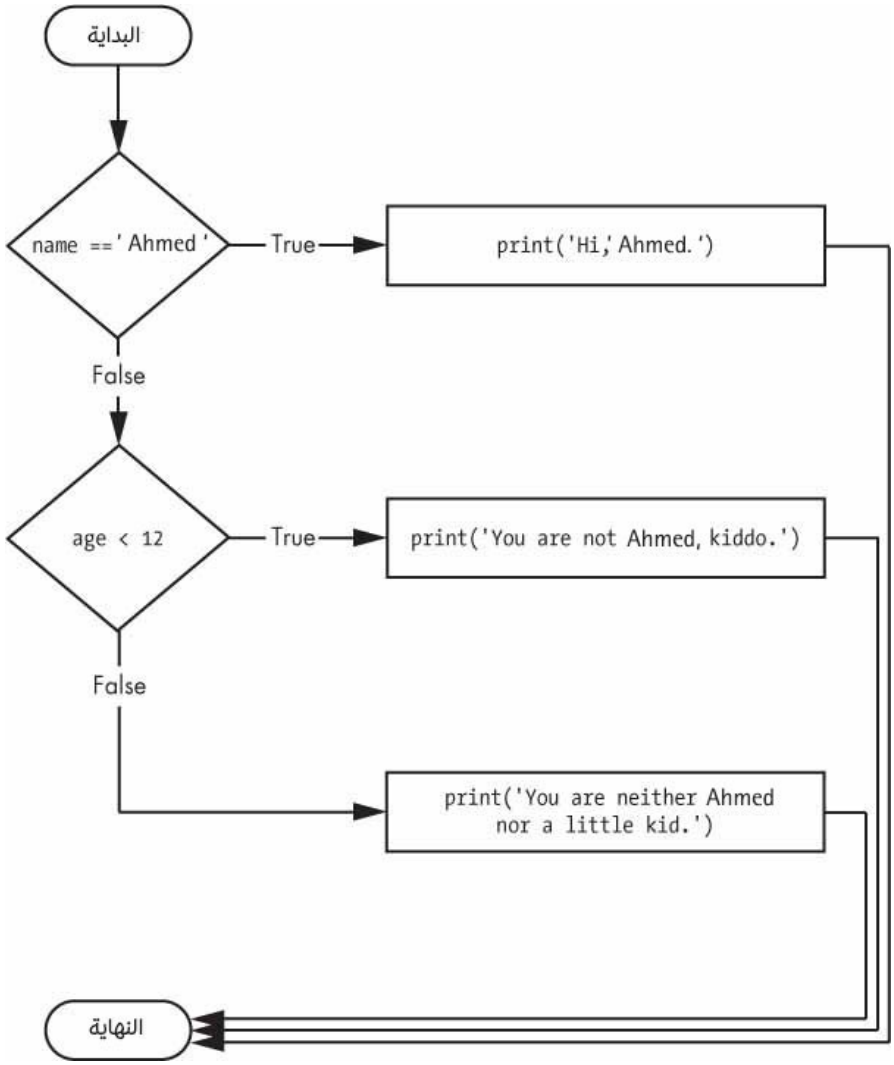
الشكل 12: المخطط التدفقي لتنفيذ برنامج vampire2.py

لاحظ أن الفرع الذي عليه إشارة × لا ينفذ أبدًا، لكنه إذا كان العمر age أكبر من 2000 فهو بكل تأكيد أكبر من 100 أيضًا.

يمكنك أن تضع عبارة `else` بعد آخر عبارة `elif` اختياريًا، وفي هذه الحالة سنضمن تنفيذ كتلة برمجية واحدة فقط لا غير، أي في حال كانت جميع شروط `if` و `elif` هي `False` فستنفَّذ كتلة `else`. لنعد كتابة مثال الترحيب بالمستخدم لنستعمل فيه `if` و `elif` و `else`:

```
name = 'Abdullatif'
age = 3000
if name == 'Ahmed':
  print('Hi, Ahmed.')
elif age < 12:
  print('You are not Ahmed, kiddo.')
else:
  print('You are neither Ahmed nor a little kid.')
```

يظهر الشكل التالي المخطط التدفقي للمثال السابق، الذي سنسميه `littleKid.py`.



الشكل 13: المخطط التدفقي لبرنامج liitleKid.py

يمكننا وصف هذا النمط من بنى التحكم باللغة العربية: «إذا كان أول شرط محققًا فافعل كذا، وإلا إن كان الشرط الثاني محققًا فافعل كذا، وإلا فافعل كذا». من المهم أن تنتبه إلى ترتيب عبارات `if` و`elif` و `else` حين استخدامها لكي تتجنب العلل المنطقية في برامجك. وتذكر أن هنالك عبارة `if` وحيدة فقط لا غير، وأي عبارات `elif` يجب أن تأتي بعدها؛ وإذا أردت ضمان تنفيذ إحدى الكتل البرمجية فأنهِ بنى التحكم بعبارة `else`.

2.6.4 حلقة التكرار while

يمكنك أن تعيد تنفيذ إحدى الكتل البرمجية مرارًا وتكرارًا باستخدام عبارة `while`. وستُنفَّذ الشيفرة الموجودة في الكتلة التي تلي شرط `while` طالما كان الشرط محققًا `True`.

تتألف عبارة `while` في بايثون مما يلي:

- الكلمة المفتاحية `while`
- الشرط، وهو التعبير الذي تكون نتيجته هي `True` أو `False`
- نقطتان رأسيتان `:`
- كتلة برمجية تلي ما سبق، تكون فيها الأسطر مسبوقة بمسافة بادئة

يمكنك ملاحظة أن عبارة `while` شبيهة جدًا بعبارة `if`، والفرق بينهما هو في السلوك. فبعد الانتهاء من الكتلة التي تلي شرط `if` سيكمل البرنامج تنفيذ الشيفرة التي تليها، لكن في نهاية كتلة `while` فسيعود تنفيذ البرنامج إلى بداية عبارة `while`. نسمي عبارة `while` عادةً «حلقة التكرار while» أو «حلقة while».

للنظر إلى الفرق بين العبارة الشرطية `if` وحلقة `while` لهما نفس الشرط وتفعلان نفس الفعل في الكتلة التي تليهما. هذه هي الشيفرة التي نستعمل العبارة الشرطية`if` فيها:

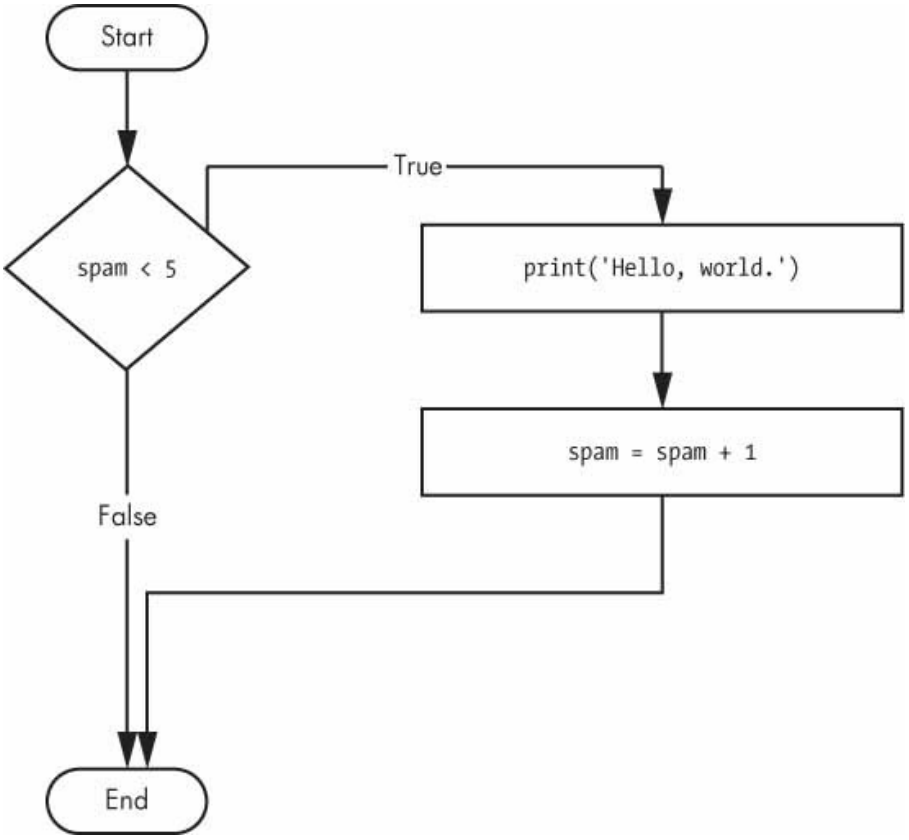
```
spam = 0
if spam < 5:
  print('Hello, world.')
  spam = spam + 1
```

وهذه هي الشيفرة التي نستعمل الحلقة `while` فيها:

```
spam = 0
while spam < 5:
  print('Hello, world.')
  spam = spam + 1
```

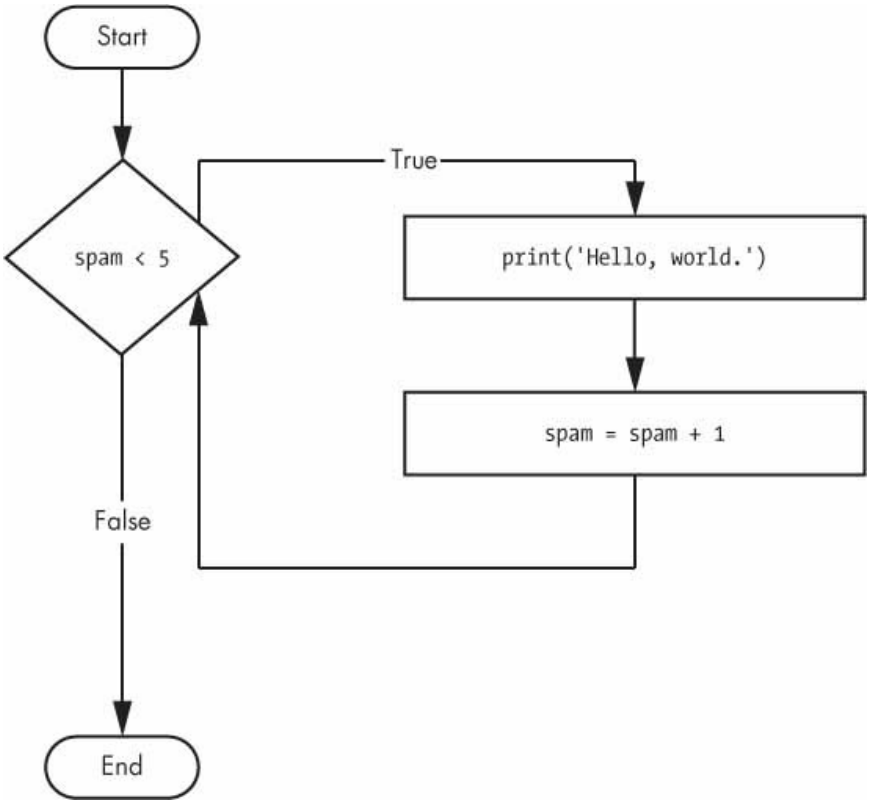
تشبه هذه التعابير بعضها بعضًا، إذ تتحقق `if` و `while` من أن قيمة المتغير `spam` أصغر من 5، ثم تعرض رسالةً ترحيبيةً وتزيد قيمة المتغير `spam` بمقدار 1. لكن حينما تشغل البرنامجين السابقين فستجد نتيجةً مختلفةً لكلٍ منهما، ففي البرنامج الذي فيه `if` سترى عبارة الترحيب مرة واحدة، بينما تكرر العبارة الترحيبية 5 مرات في برنامج `while`!

لننظر إلى المخططين التدفقيين للبرنامجين السابقين في الشكلين الآتيين لنفهم ما حدث:



الشكل 14: المخطط التدفقي للبرنامج الذي يحتوي على العبارة الشرطية if

وفي الآتي المخطط التدفقي للبرنامج الذي يحتوي على العبارة while:



الشكل 15: المخطط التدفقي للبرنامج الذي يحتوي على العبارة الشرطية while

تتحقق العبارة الشرطية `if` من الشرط وتطبع `"Hello, World"` مرةً واحدةً بعد تحقق الشرط، ثم ينتهي تنفيذ البرنامج. بينما البرنامج الذي فيه حلقة `while` فسينفذ 5 مرات لأن قيمة العدد `spam` تزيد مـرة في نهايـة كل حلقة تكرار، وهذا يعني أن الحلقة ستنفذ 5 مرات قبل أن يصبح الشرط `spam < 5` غير محقق `False`.

ستجري عملية التحقق من شرط حلقة `while` في بداية كل دورة iteration، أي في بداية كل تنفيـذ لحلقـة `while`؛ وإذا كان الشرط محققًا `True` فستنفذ الكتلة، ثم بعد ذلك يعاد التحقق من الشرط مجددًا.

لاحظ أنه إذا كان شرط حلقة `while` غـير محقـق `False` من أول مـرة فلن تنفَّـذ الحلقـة أبـدًا وسـيتخطاها البرنامج.

ا. حلقة while المزعجة

هذا برنامج بسيط يطلب منك باستمرار أن تدخل `"your name"` في سطر الأوامر حين سؤاله عن اسمك.

أنشِئ ملفًا جديدًا من File ثم New وأدخل الشيفرة الآتية واحفظها في الملف `yourName.py`:

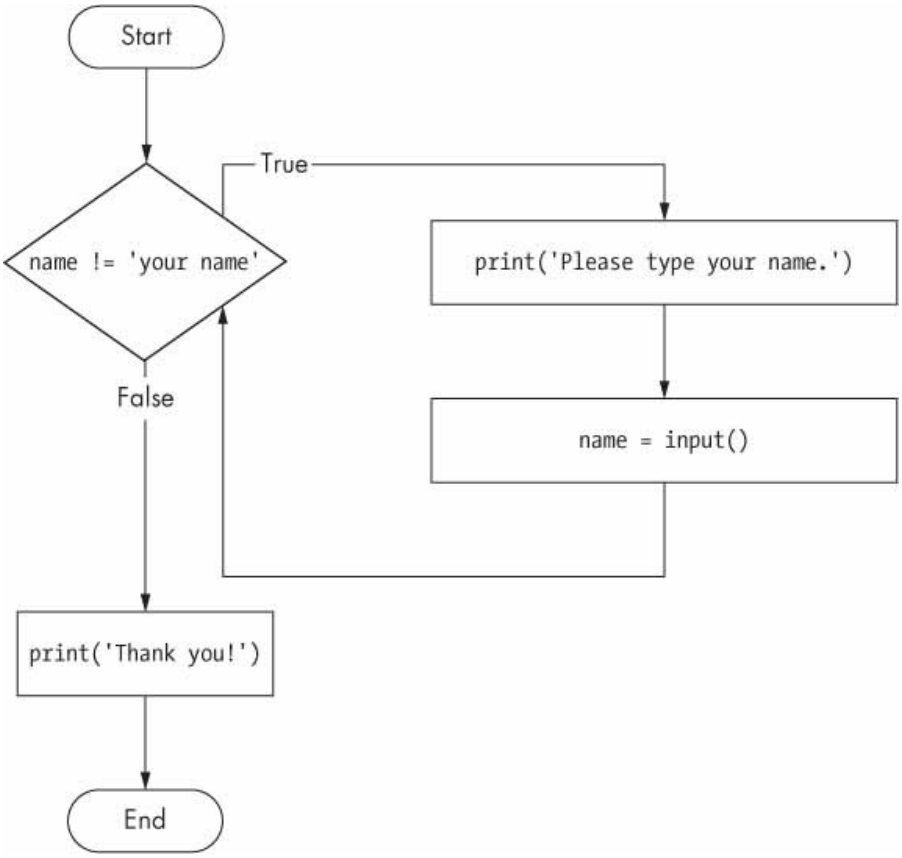
```
❶ name = ''

❷ while name != 'your name':
```

```
    print('Please type your name.')
❸ name = input()
❹ print('Thank you!')
```

يضبط البرنامجُ المتغيرَ name إلى سلسلة نصية فارغة ❶، وبهذا يكون الشرط name != 'your name' محققًا True وبالتالي سيبدأ تنفيذ حلقة while ❷.

سيسأل البرنامجُ المستخدمَ عن اسمه، ثم يأخذ المدخلات ويخزنها في المتغير name ❸، ولمّا كنّا قد وصلنا إلى نهاية حلقة التكرار فسنعود إلى بداية حلقة while ونتحقق من الشرط، وإن لم يكن الاسم name مساويًا إلى السلسلة النصية 'your name' فيسكون الشرط محققًا True وسيعاد تنفيذ حلقة while مجددًا. كن حينما يدخل المستخدم الكلمتين your name فسيصبح شرط حلقة while غير محقق False، وبالتالي بدلًا من إعادة تكرار الحلقة فسيكمل برنامجنا تنفيذ بقية البرنامج ❹.

المخطط التدفقي في الشكل التالي يوضح آلية عمل البرنامج yourName.py:



الشكل 16: المخطط التدفقي للبرنامج yourName.py

لنجرب الآن البرنامج، بالضغط على زر F5 لتشغيله، ثم كتابة أي عبارة سوى "your name" عدة مرات قبل أن نرضخ للضغط الذي يمارسه البرنامج تجاهنا ونكتب "your name".

```
Please type your name.
Abdullatif
Please type your name.
Hsoub
Please type your name.
%#@#%*(^&!!!
Please type your name.
your name
Thank you!
```

إذا لم ندخل "your name" أبدًا فلن يصبح شرط `while` غير محقق `False` وبالتالي سيستمر تنفيذ البرنامج إلى الأبد. وفي حالة برنامجنا كانت لدينا الدالة `input()` التي تمكننا من إكمال سير البرنامج حينما ندخل العبارة الصحيحة وبالتالي تتغير حالة الشرط، لكن قد لا يتغير الشرط في بعض البرامج مما يسبب مشكلة، لذا هنالك حاجة لتعلم طريقة للخروج من حلقة `while`.

2.6.5 العبارة break

هنالك طريقة نجعل فيها برنامجنا يخرج من حلقة `while` قبل انتهاء تنفيذها. فإذا وصل التنفيذ إلى عبارة `break` فسيخرج البرنامج من حلقة `while` مباشرةً. وتتألف عبارة `break` في بايثون من الكلمة المحجوزة `break` فقط.

أليس ذلك بسيطًا؟ لنكتب برنامجًا يشبه البرنامج السابق لكنه يستخدم العبارة `break` للخروج من حلقة التكرار، أدخِل الشيفرة الآتية واحفظها في ملف باسم `yourName2.py`:

```
❶ while True:
    print('Please type your name.')
  ❷ name = input()
  ❸ if name == 'your name':
      ❹ break
❺ print('Thank you!')
```

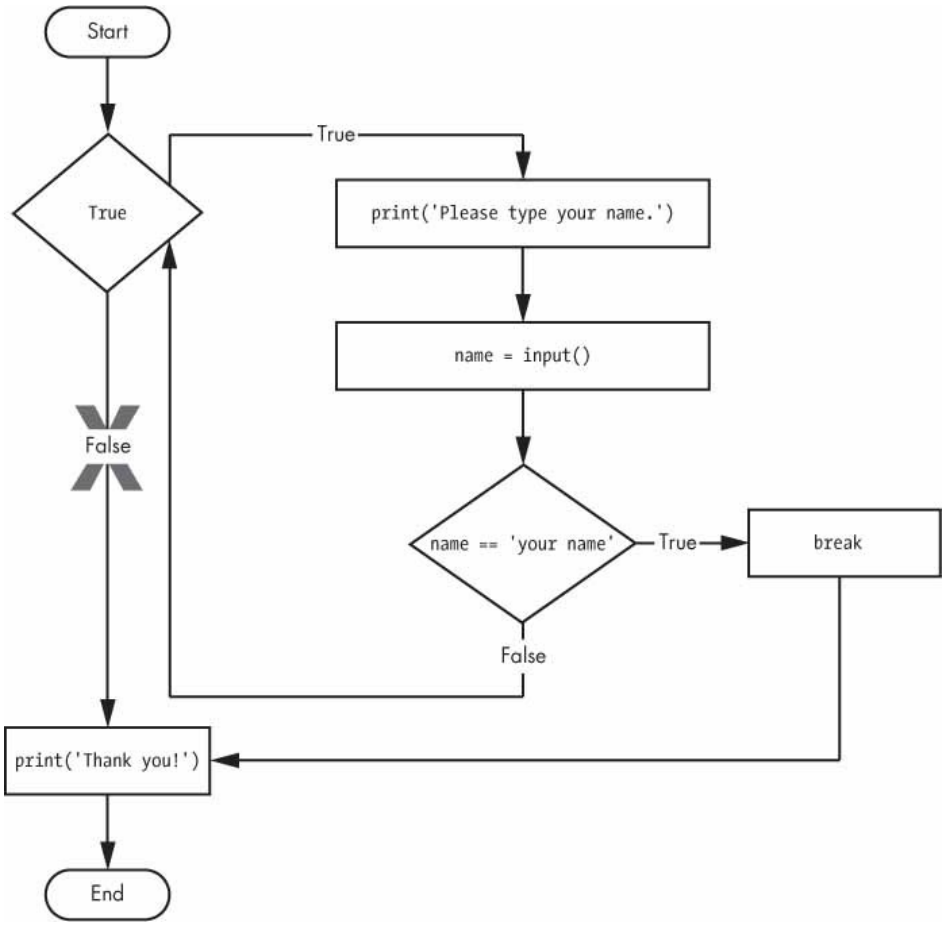
ننشِئ في أول سطر ❶ حلقة تكرار لا نهائية، وذلك بجعل شرط حلقة `while` محققًـا دومًـا `True`، وبالتأكيـد إذا وضعنا `True` شرطًا لحلقة `while` فستكون قيمته هي `True` دومًا.

بعد أن يبدأ تنفيذ حلقة التكـرار فلن يخـرج البرنـامج منهـا إلا إذا اسـتخدام عبـارة `break`، لاحـظ أن حلقـات التكرار اللانهائية التي لا ينتهي تنفيذها أبدًا هي علّة منطقية في البرامج.

وكما في المثال السابق، سيطلب البرنامج من المستخدم أن يدخل your name ❷، وسنتحقق إن كان المتغير `name` يساوي `'your name'` باستخدام البنية الشرطية `if` ❸، وإذا كان الشرط محققًا `True` فستنفذ العبارة `break` ❹، وسينتقل التنفيذ إلى خارج الحلقة وستطبع رسالة الشكر ❺.

إذا لم يكن شرط العبارة `if` محققًا فهذا يؤدي إلى دورة جديدة لحلقة `while`، وسيتحقق البرنامج من شـرط تنفيذ حلقة `while` ❶، ولمّا كان الشرط محققًا `True` دومًا، فستنفذ حلقة التكرار مجددًا وتسأل المستخدم أن يدخل your name.

يوضح المخطط التدفقي بالشكل التالي، آلية عمل البرنامج `yourName2.py`:



الشكل 17: المخطط التدفقي للبرنامج yourName2.py مع حلقة تكرار لا نهائية

لاحظ أن المسار × لا ينفذ منطقيًا أبدًا لأن شرط الحلقة هو `True` دومًا.

2.6.6 هل وقعت في حلقة تكرار لا نهائية؟

إذا شغلت تطبيقًا يحتوي على علة تؤدي إلى حلقة تكرار لا نهائية، فاضغط على Ctrl+C أو أعد تشغيل الصدفة من Shell ثم Restart Shell؛ مما يرسل إشارة KeyboardInterrupt إلى برنامجك تؤدي إلى إيقاف تشغيله مباشرةً. يمكنك التجربة بإنشاء برنامج بسيط باسم infiniteLoop.py:

```
while True:
  print('Hello, world!')
```

سيطبع البرنامج السابق العبارة Hello, World! إلى اللانهاية لأن شرط حلقة while محقق True دومًا. قد تستفيد من استخدام الاختصار Ctrl+C لإنهاء تنفيذ البرامج حتى دون أن تكون عالقًا في حلقة تكرار لا نهائية.

2.6.7 عبارة continue

وكما في عبارة break، نستعمل العبارة continue داخل حلقات التكرار، وحينما يصل التنفيذ إلى عبارة continue فسينتقل تنفيذ البرنامج إلى بداية حلقة التكرار مباشرةً ويعيد التحقق من شرط الحلقة، أي نفس ما يحدث حين الوصول إلى نهاية دورة حلقة التكرار.

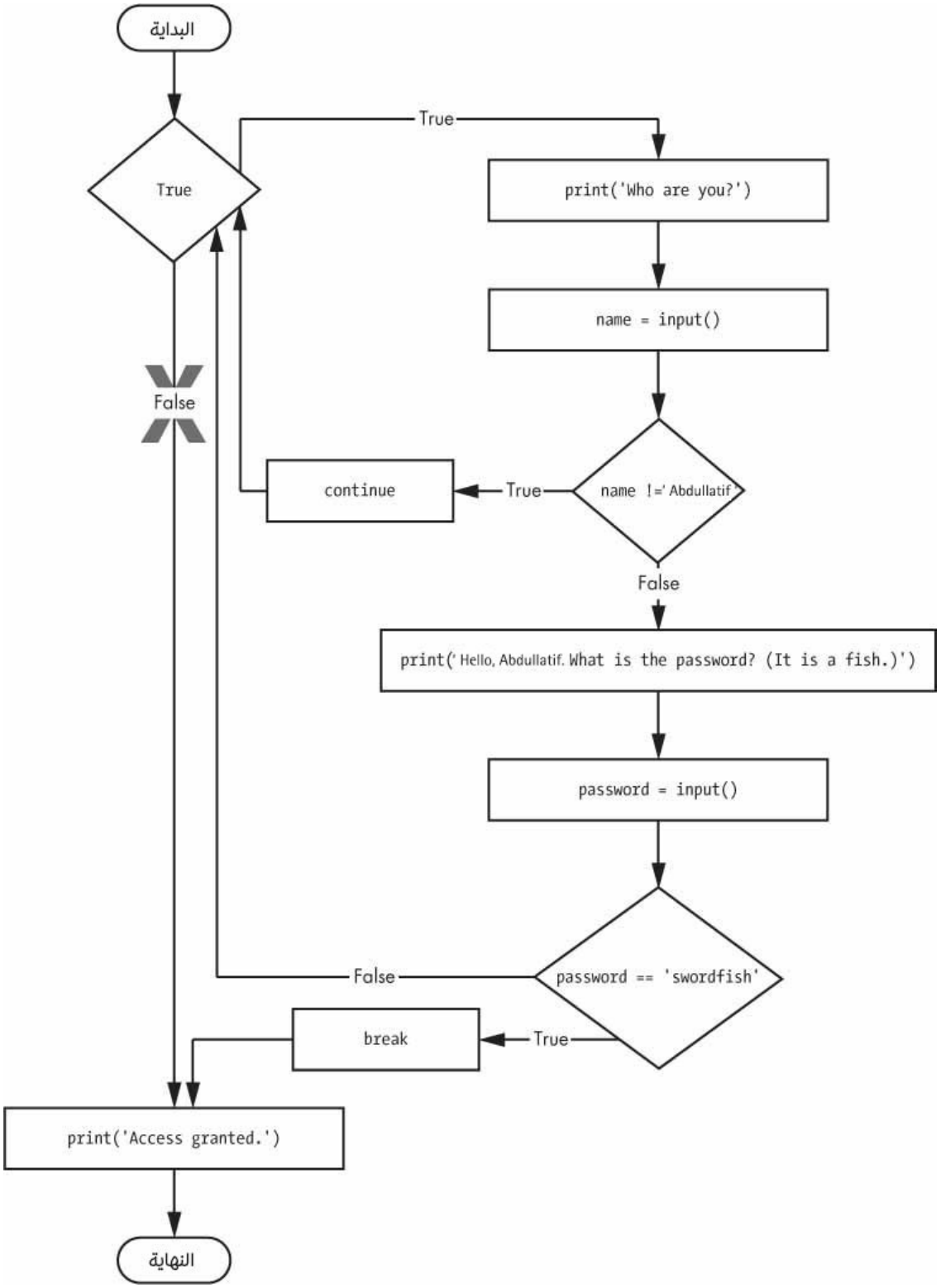
لنستخدم continue لكتابة برنامج يسأل عن اسم المستخدم وكلمة المرور، أدخل ما يلي في ملف جديد واحفظه باسم swordfish.py:

```
 while True:
   print('Who are you?')
   name = input()
❶  if name != 'Abdullatif':
❷    continue
   print('Hello, Abdullatif. What is the password? (It is a fish.)')
❸  password = input()
   if password == 'swordfish':
❹    break
❺ print('Access granted.')
```

إذا أدخل المستخدم أي اسم باستثناء Abdullatif ❶ فستنقل العبارة `continue` ❷ التنفيذ إلى بداية حلقة التكرار، وحين إعادة التحقق من شرط الدخول إلى الحلقة فسيكون محققًا دومًا لأنه `True`.

بعد تجاوز المستخدم الشرط الموجود في `if` فسنسأله عن كلمة المرور ❸، وإذا أدخل كلمة المرور swordfish فستنفذ عبارة `break` ❹ وبالتالي نخرج من حلقة التكرار كليًا وستطبع العبارة Access granted❺، وإذا لم تكن كلمة المرور صحيحةً فسنصل إلى نهاية دورة حلقة التكرار ثم نعود إلى بدايتها ونتحقق من الشرط مجددًا الذي هو `True` دومًا...

المخطط التدفقي في الشكل التالي يوضح آلية عمل البرنامج `swordfish.py`:



الشكل 18: المخطط التدفقي للبرنامج swordfish.py

لاحظ أن المسار × لا ينفذ منطقيًا أبدًا لأن شرط الحلقة هو `True` دومًا.

شغل البرنامج السابق وجرب بعض المدخلات، ولن يسألك البرنامج عن كلمة المرور حتى تدعي أنك Abdullatif، وسيطبع لك رسالة أن الوصول مسموح لك إن أدخلت كلمة المرور الصحيحة:

```
Who are you?
I'm fine, thanks. Who are you?
Who are you?
Abdullatif
Hello, Abdullatif. What is the password? (It is a fish.)
Mary
Who are you?
Abdullatif
Hello, Abdullatif. What is the password? (It is a fish.)
swordfish
Access granted.
```

ا. القيم التي تكافئ True والقيم التي تكافئ False

ستعدّ الشروط في بايثون بعض القيم في أنواع البيانات المختلفة على أنها مكافئة للقيمة `True` وأخرى للقيمة `False`. فلو استخدمنا القيم `0` و `0.0` و '' (سلسلة نصية فارغة) في الشروط فستكافئ `False`، بينما ستكافئ أي قيمة أخرى `True`. ألقِ نظرةً هنا:

```
name = ''
❶ while not name:
  print('Enter your name:')
  name = input()
print('How many guests will you have?')
numOfGuests = int(input())
❷ if numOfGuests:
  ❸ print('Be sure to have enough room for all your guests.')
print('Done')
```

يبدأ البرنامج بتهيئة المتغير `name` مع قيمة نصية فارغة، لذا سيكون شرط حلقة `while` محققًا `True` ❶، وسيتحقق أيضًا إذا أدخل المستخدم سلسلة نصية فارغة أثناء سؤاله عن الاسم `name` (بالضغط مباشرةً على زر Enter دون كتابة شيء).

سيبدأ تنفيذ الحلقة بطلب إدخال الاسم وعدد الضيوف، وإذا كان عدد الضيوف `numOfGuests` ليس صفرًا 0 ❷ فسيكون الشرط محققًا `True` وسيطبع البرنامج تذكيرًا للمستخدم ❸.

كـــان بإمكاننـــا كتابـــة `name != ''` بـــدلًا من `not name`، و `numOfGuests != 0` بـــدلًا من `numOfGuests`، لكن استخدام القيم التي تكافئ `True` أو `False` في شروط سيجعل مقروئية شيفرتك أفضل.

2.6.8 حلقات تكرار for والدالة range()

ستعمل حلقة `while` لطالما كان الشـرط محققًـا `True` (ومن هنـا أتى اسـمها while)، لكن مـاذا لـو أردنـا تنفيذ كتلة من الشيفرات لعدد محدد من المرات؟ يمكننا فعل ذلك عبر حلقة التكرار `for` والدالة `range()`.

ستبدو عبارة `for` كالآتي: `for i in range(5):` وستتضمن ما يلي:

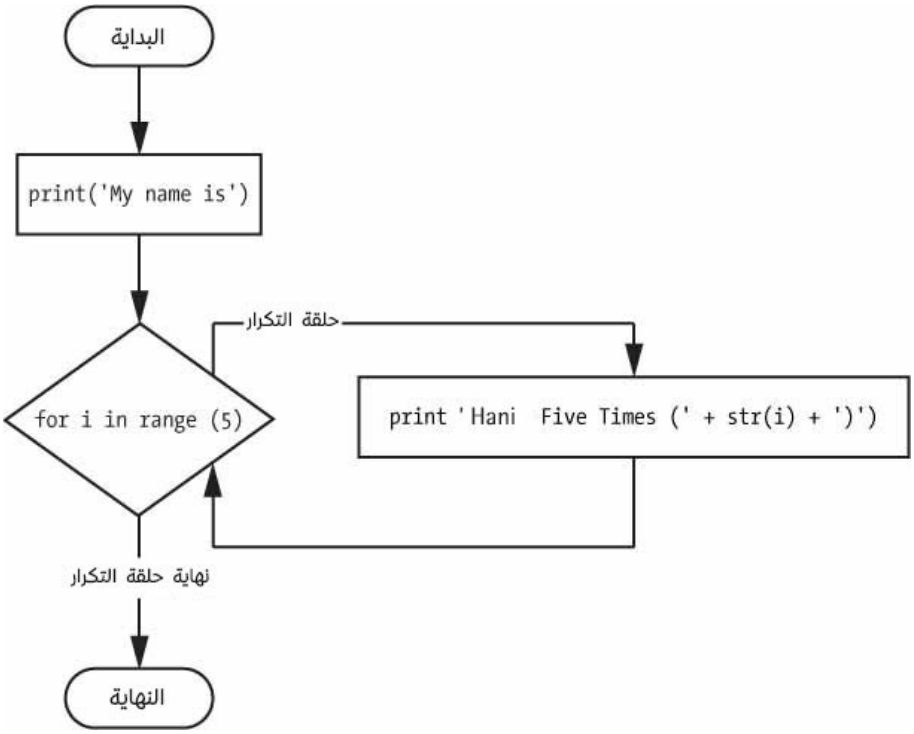
- الكلمة المحجوزة `for`
- اسم المتغير
- الكلمة المحجوزة `in`
- استدعاء للدالة `range()` مع تمرير 3 أعداد صحيحة كحد أقصى إليها
- نقطتان رأسيتان `:`
- كتلة برمجية تلي ما سبق، تكون فيها الأسطر مسبوقة بمسافة بادئة

لننشئ برنامجًا ولنسمه `fiveTimes.py` الذي يساعدنا على معاينة حلقة `for` عمليًا:

```
print('My name is')
for i in range(5):
  print('Hani Five Times (' + str(i) + ')')
```

سترى أن الكتلة البرمجية لحلقة التكرار `for` تنفذ 5 مرات، وستكون قيمة المتغير `i` في أول مرة تعمل فيها الحلقة هو 0، وستطبع `print()` العبارة `Hani Five Times (0)` ثم بعد أن تنتهي بايثون من تنفيـذ أول دورة في حلقة التكرار فسيعود التنفيذ إلى بداية الحلقة وستزيد قيمة المتغير `i` بمقدار 1، ولهذا سيؤدي استدعاء الدالة `range(5)` إلى حدوث 5 تكرارات داخل حلقة `for`؛ إذ سـيبدأ المتغـير `i` من القيمـة `0` ثم `1` ثم `2` ثم `3` ثم `4`. وعمومًا، ستزداد قيمة المتغير حتى تصل إلى الرقم الممرر للدالة `range()` لكن دون تضمينه في النتائج.

يوضح الآتي المخطط التدفقي لهذا البرنامج:



الشكل 19: المخطط التدفقي للبرنامج

إذا شغلت البرنامج فسترى العبارة Hani Five Times متبوعةً بقيمة المتغير `i` في دورة حلقة `for` الحالية:

```
My name is
Hani Five Times (0)
Hani Five Times (1)
Hani Five Times (2)
Hani Five Times (3)
Hani Five Times (4)
```

ملاحظة: يمكنك استخدام `break` و `continue` داخل حلقات `for` أيضًا، وستؤدي العبارة `continue` إلى تخطي الدورة الحالية لحلقة التكرار والانتقال إلى الدورة الآتية، كما لو أن تنفيذ البرنامج وصل إلى نهاية دورة حلقة `for` الحالية ثم عاد إلى بداية الحلقة. يمكنك أن تستعمل العبارتين `break` و `continue` داخل حلقات `while` و `for` فقط، وإذا حاولت أن تجرب استخدامها في مكانٍ آخر فستعطيك بايثون رسالة خطأ.

لنأخذ قصة عالم الرياضيات الشهير كارل فريدريش غاوس (قد تعرفه باسم غـاوس أو Gauss)، أراد مدرسـه حينما كان صغيرًا أن يعطيه وظيفة صعبة وطلب منه جمع الأرقام من 0 إلى 100، وخطـرت ببـال الفـتى غـاوس طريقة ذكية للإجابة عن ذاك السؤال بثوانٍ معدودة، لكن لنكتب الآن برنامج بايثون يحسب لنا النـاتج ويسـتعمل الحلقة `for`:

```
❶ total = 0

❷ for num in range(101):

  ❸ total = total + num

❹ print(total)
```

يفترض أن يكون الناتج 5,050. نضبط قيمة المتغير `total` إلى 0 ❶ في بداية البرنامج، ثم نبدأ حلقة `for`❷ التي ننفذ فيها `total = total + num` ❸ مئة مرة، وبعد أن تنتهي دورات التكرار المئة فسنكون قد جمعنا الأعداد الصحيحة من 0 إلى 100 في المتغير `total`، ثم نطبق قيمة `total` إلى الشاشة ❹. سيعمل برنامجنا بأجزاء من الثانية ويخبرنا بالناتج النهائي.

(اكتشف غاوس حينما أعطاه المدرس هذه الوظيفة حلها بطريقة ذكية: هنالك خمسون زوجًا من الأرقام التي يكون مجموعها 101 مثل 1 + 100 و 2 + 99 و 3 + 98 وهلمَّ جرًا، حتى يصل إلى 50 + 51. ولمّا كان جداء 50 × 101 هو 5,050 فسيكون مجموع جميع الأرقام من 0 إلى 100 هو 5,050. يا له من فتى ذكي!)

2.6.9 كتابة حلقة while تكافئ حلقة for

يمكنك عمليًا أن تكتب حلقة `while` لتكافئ في عملها حلقة `for`، وستجد أن كتابة حلقات `for` مختصرة أكثر. لنعد كتابة المثال `iveTimes.py` ليستعمل الحلقة `while`:

```
print('My name is')
i = 0
while i < 5:
  print('Hani Five Times (' + str(i) + ')')

  i = i + 1
```

إذا شغلت البرنامج فمن المفترض أن يكون الناتج مماثلًا تمامًا للمثال `fiveTimes.py` الذي يستعمل حلقة `for`. اعلم أنك تستطيع كتابة أمور كثيرة في البرمجة بأكثر من طريقة، لكن عليك اختيار الأداة الأنسب لأداء المهمة التي تريدها.

2.6.10 وسائط الدالة range(): البداية والنهاية والخطوة

يمكن أن تستدعى بعض الدوال مع عدّة وسائط arguments مفصولة بفاصلة، والدالة `range()` هي إحدى هذه الدوال. يسمح لك تمرير وسائط للدالة `range()` أن تغير سلوكها، فمثلًا يمكنك تحديد الرقم الذي يجب أن تبدأ منه الدالة `range()`:

```
for i in range(12, 16):
  print(i)
```

يمثِّل الوسيط 1 من أين يجب لحلقة `for` البدء، ويمثل الوسيط 2 أين يجب أن تتوقف (دون تضمين رقم):

```
12
13
14
15
```

يمكننا أيضًا استدعاء الدالة `range()` مع ثلاثة وسائط، ويكون أول وسيطين هما البداية والنهاية، أما الوسيط الثالث فسيكون «الخطوة step»، الذي يشير إلى مقدار زيادة قيمة المتغير عند كل دورة:

```
for i in range(0, 10, 2):
  print(i)
```

أي أن استدعاء `range(0, 10, 2)` سيعد من الصفر إلى الثمانية وبخطوة 2:

```
0
2
4
6
8
```

الدالة `range()` مرنة ويمكنك استخدامها لتوليد أي سلسلة أرقام. على سبيل المثال، يمكنك استخدام رقم سالب كوسيط لقيمة الخطوة مما يؤدي إلى العد عكسيًا تنازليًا:

```
for i in range(5, -1, -1):
  print(i)
```

ستنتج حلقة `for` السابقة الناتج الآتي:

```
5
4
3
2
1
0
```

نجد أن المجال `range(5, -1, -1)` مع حلقة التكرار `for` سيؤدي إلى طباعة 5 أعداد تنازليًا من 5 إلى غاية 0.

2.7 استيراد الوحدات

يمكن لجميع برامج بايثون أن تستدعي مجموعةً من الدوال الأساسية نسميها الدوال المضمنة في اللغة أو «الدوال المضمنة built-in functions»، والتي تتضمن الدوال التي تعرفت عليها سابقًا مثل `print()` و`input()` و `len()`.

تأتي بايثون أيضًا مع مجموعة من الوحدات الأساسية modules التي نسميها بالمكتبة القياسية standard library. تتألف كل وحدة من عدد برامج بايثون التي فيها مجموعة من الدوال التي يمكنك استخدامها في برامجك. فمثلًا تحتوي الوحدة `math` على مجموعة من الدوال المتعلقة بالعمليات الرياضية، بينما تضم الوحدة random مجموعة من الدوال التي تجري عمليات على الأرقام المولدة عشوائيًا، وهكذا.

قبل أن نستخدم الدوال الموجودة في إحدى تلك الوحدات في برامجنا، يجب علينا أولًا استيراد تلك الوحدة باستخدام العبارة `import`. وتتألف عبارة `import` مما يلي:

- الكلمة المحجوزة `import`
- اسم الوحدة التي نريد استيرادها
- واختياريًا أسماء وحدات أخرى نريد استيرادها على أن نفصل بينها بفاصلة

بعد أن تستورد إحدى الوحدات فيمكنك أن تستعمل جميع الدوال الرائعة الموجودة فيها، ولنضرب مثالًا الوحدة `random` التي تمنحنا وصولًا إلى الدالة `random.randint()` حين استيرادها.

احفظ الشيفرة الآتية في ملف باسم `printRandom.py`:

```
import random
for i in range(5):
  print(random.randint(1, 10))
```

سيطبع البرنامج السابق ناتجًا يشبه الناتج الآتي حين تشغيله:

```
4
1
8
4
1
```

ستكون نتيجة استدعاء الدالة `random.randint()` هي عدد صحيح عشوائي يقع بين العددين الذين مررتهما إلى الدالة كوسيطين. ولمّا كانت الدالة `randint()` موجودةً في الوحدة `random`، فعليك أن تكتب الكلمة `random` أولًا قبل اسم الدالة لتخبر بايثون أنك تريد استخدام الدالة الموجودة في الوحدة `random`، ونفصل بين اسم الوحدة واسم الدالة بنقطة.

هذا مثال لعبارة استيراد تستورد أربع وحدات في آنٍ واحد:

```
import random, sys, os, math
```

يمكننا الآن استخدام أي دالة موجودة في الوحدات الأربع السابقة، وسنتعلم المزيد عن تلك الوحدات لاحقًا في هذا الكتاب.

2.7.1 عبارة from import

هنالك شكل بديل لعبارة `import` يتضمن الكلمة المحجوزة `from`، والتي نتبعها باسم الوحدة، ثم الكلمة المحجوزة `import` ثم نجمة `*`؛ فمثلًا `from random import *`.

وحين استيراد الوحدات بهذا الشكل فلا حاجة إلى وضع السابقة `random.` قبل أسماء الدوال التي نريد استدعاءها؛ لكن في المقابل من الأفضل كتابة الاسم الكامل للدالة مع السابقة التي تشير إلى اسم الوحدة لزيادة مقروئية الشيفرة البرمجية.

2.8 إنهاء تنفيذ البرنامج حينما نشاء باستخدام الدالة sys.exit()

آخر مفهوم من مفاهيم بنى التحكم التي سنشرحها في هذا الفصل هو آلية إنهاء تنفيذ البرنامج.

ينتهي تنفيذ البرامج دومًا حينما يصل التنفيذ إلى نهاية الملف، لكن يمكننا إنهاء تنفيذ البرنامج قبل آخر تعليمة برمجية باستخدام الدالة `sys.exit()`. ولمّا كانت هذه الدالة جزءًا من الوحدة `sys` فعلينا أن نستورد تلك الوحدة في بداية البرنامج قبل استخدامها.

افتح محرر الشيفرات واكتب ما يلي واحفظه `exitExample.py`:

```
import sys

while True:
  print('Type exit to exit.')
  response = input()
  if response == 'exit':
```

```
    sys.exit()

print('You typed ' + response + '.')
```

شغل البرنامج السابق وستجد أنك دخلت في حلقة تكرار لا نهائية دون وجود عبارة `break` داخلها، والطريقة الوحيدة لكي ينتهي تنفيذ البرنامج هي الوصول إلى استدعاء الدالة `sys.exit()`، ولأن قيمة المتغير `response` تساوي ما يدخله المستخدم عبر `input()`، فإذا أدخلت `exit` فستتحقق العبارة الشرطية `if` وسينتهي تنفيذ البرنامج.

2.9 برنامج قصير: احزر الرقم

جميع الأمثلة السابقة كانت بسيطة جدًا لكنها مناسبة لاستيعاب المفاهيم البرمجية الأساسية، لكن لنبني الآن برنامجًا متكاملًا نوظِّف فيه المعلومات التي تعلمناها. سنكتب في هذا القسم لعبة «احزر الرقم»، والتي ستبدو كما يلي حين تشغيلها:

```
I am thinking of a number between 1 and 20.
Take a guess.
10
Your guess is too low.
Take a guess.
15
Your guess is too low.
Take a guess.
17
Your guess is too high.
Take a guess.
16
Good job! You guessed my number in 4 guesses!
```

افتح محرر الشيفرات وأدخل الشيفرة الآتية واحفظها في ملف باسم `guessTheNumber.py`:

```
# هذه لعبة لتخمين الرقم

import random
secretNumber = random.randint(1, 20)
print('I am thinking of a number between 1 and 20.')
```

```
# اطلب من اللاعب أن يحزر الرقم 6 مرات
for guessesTaken in range(1, 7):
  print('Take a guess.')
  guess = int(input())

  if guess < secretNumber:
    print('Your guess is too low.')
  elif guess > secretNumber:
    print('Your guess is too high.')
  else:
    break     # هذا الشرط هو التخمين الصحيح
if guess == secretNumber:
  print('Good job! You guessed my number in ' + str(guessesTaken) +
'guesses!')
else:
  print('Nope. The number I was thinking of was ' + str(secretNumber))
```

لنقسم البرنامج إلى أقسام صغيرة ونناقشها كلًا على حدة، بدءًا من أعلى الملف:

```
# هذه لعبة لتخمين الرقم
import random
secretNumber = random.randint(1, 20)
```

يبدأ البرنامج بتعليق في أول سطر فيه يشرح ماذا يفعل البرنامج، ثم يستورد الوحدة `random` لكي نستطيع استخدام الدالة `random.randint()` لتوليد رقم ليحزره اللاعب، وسنخزن الرقم العشوائي النـاتج في المتغـير `secretNumber`.

```
print('I am thinking of a number between 1 and 20.')

# اطلب من اللاعب أن يحزر الرقم 6 مرات
for guessesTaken in range(1, 7):
```

```
    print('Take a guess.')
    guess = int(input())
```

سيخبر البرنامج اللاعب أنه قد «فكر» في رقم سري ويملك اللاعب ست محاولات ليحزره، وستكون الشيفرة التي تسمح للاعب بإدخال رقم والتحقق منه موجودة في حلقة `for` التي ستنفذ ست مرات كحد أقصى.

أول ما تفعله حلقة التكرار هو طلب كتابة تخمين للرقم السري عبر `input()`، ولأن الدالة `input()` تعيد سلسلةً نصية وليس رقمًا صحيحًا فنمرر الناتج إلى الدالة `int()`، وثم يخزن الرقم الذي أدخله المستخدم في المتغير `guess`.

```
    if guess < secretNumber:
      print('Your guess is too low.')
    elif guess > secretNumber:
      print('Your guess is too high.')
```

تتحقق الشروط السابقة إن كان تخمين المستخدم أصغر أو أكبر من الرقم السري، وفي كلتي الحالتين سيوفر البرنامج تلميحًا للاعب.

```
    else:
      break     # هذا الشرط هو التخمين الصحيح
```

إذا لم يكن تخمين اللاعب أصغر ولا أكبر من الرقم السري فهذا يعني أنه يساويه، وفي هذه الحالة سيخرج البرنامج من حلقة `for` عبر `break`.

```
if guess == secretNumber:
  print('Good job! You guessed my number in ' + str(guessesTaken) + '
guesses!')
else:
  print('Nope. The number I was thinking of was ' + str(secretNumber))
```

بعد نهاية حلقة `for` سيتحقق البرنامج عبر عبارة `if…else` أن اللاعب قد خمَّن الرقم السري الصحيح، ويعرض رسالة مناسبة لكل حالة.

تذكر أن الشيفرات التي تقع بعد حلقة التكرار ستنفذ بعد انتهاء تنفيذ حلقة التكرار سواءً بإكمالها 6 مرات وعدم تخمين اللاعب للرقم الصحيح، أو بالخروج منها عبر `break` حين تخمين الرقم السري الصحيح.

سـيعرض برنامجنـا رسـالتين نصـيتين فيهمـا متغـير يحمـل قيمـةً عدديـةً صـحيحةً `guessesTaken` و`secretNumber`، ولأننا نحاول ضم عدد صحيح إلى سلسلة نصية فيجب أن نمرر الرقم إلى الدالة `str()` أولًا، ثم نضم السلاسل النصية عبر العامل `+` لتمريرها إلى الدالة `print()` لطباعتها على الشاشة.

2.10 برنامج قصير: حجرة ورقة مقص

لنستخدم المفاهيم البرمجية التي تعلمناها لإنشاء لعبة «حجرة ورقة مقص» الشهيرة.

سيكون ناتج تشغيل اللعبة كما يلي:

```
ROCK, PAPER, SCISSORS
Wins, 0 Losses, 0 Ties
Enter your move: (r)ock (p)aper (s)cissors or (q)uit
p
PAPER versus...
PAPER
It is a tie!
Wins, 1 Losses, 1 Ties
Enter your move: (r)ock (p)aper (s)cissors or (q)uit
s
SCISSORS versus...
PAPER
You win!
Wins, 1 Losses, 1 Ties
Enter your move: (r)ock (p)aper (s)cissors or (q)uit
q
```

افتح نافذةً جديدةً في محرر الشيفرات وأدخل الشيفرة الآتية واحفظها في ملف باسم `rpsGame.py`:

```
import random, sys

print('ROCK, PAPER, SCISSORS')
```

ستتبّع هذه المتغيرات عدد مرات الربح والخسارة والتعادل.

```
wins = 0
losses = 0
ties = 0
```

```
while True: # دورة اللعبة
  print('%s Wins, %s Losses, %s Ties' % (wins, losses, ties))
  while True: # مدخلات المستخدم
    print('Enter your move: (r)ock (p)aper (s)cissors or (q)uit')
    playerMove = input()
    if playerMove == 'q':
      sys.exit() # الخروج من البرنامج
    if playerMove == 'r' or playerMove == 'p' or playerMove == 's':
      break # الخروج من حلقة مدخلات المستخدم
    print('Type one of r, p, s, or q.')
  # عرض ما اختاره اللاعب
  if playerMove == 'r':
    print('ROCK versus...')
  elif playerMove == 'p':
    print('PAPER versus...')
  elif playerMove == 's':
    print('SCISSORS versus...')
  # عرض ما اختاره الحاسوب
  randomNumber = random.randint(1, 3)
  if randomNumber == 1:
    computerMove = 'r'
    print('ROCK')
  elif randomNumber == 2:
    computerMove = 'p'
    print('PAPER')
```

```
    elif randomNumber == 3:
      computerMove = 's'
      print('SCISSORS')
    # عرض النتيجة الربح/الخسارة/التعادل
    if playerMove == computerMove:
      print('It is a tie!')
      ties = ties + 1
    elif playerMove == 'r' and computerMove == 's':
      print('You win!')
      wins = wins + 1
    elif playerMove == 'p' and computerMove == 'r':
      print('You win!')
      wins = wins + 1
    elif playerMove == 's' and computerMove == 'p':
      print('You win!')
      wins = wins + 1
    elif playerMove == 'r' and computerMove == 'p':
      print('You lose!')
      losses = losses + 1
    elif playerMove == 'p' and computerMove == 's':
      print('You lose!')
      losses = losses + 1
    elif playerMove == 's' and computerMove == 'r':
      print('You lose!')
      losses = losses + 1
```

لنمعن النظر إلى الشيفرة من بدايتها:

```
import random, sys

print('ROCK, PAPER, SCISSORS')

# ستتبّع هذه المتغيرات عدد مرات الربح والخسارة والتعادل
wins = 0
losses = 0
ties = 0
```

سنستورد في البداية الوحدتين `random` و `sys` لكي نستطيع استدعاء الدالتين `random.randint()` و`sys.exit()`، ثم سنهيئ ثلاثة متغيرات لكي نتتبع عدد مرات ربح أو خسارة أو تعادل اللاعب.

```
while True: # دورة اللعبة
  print('%s Wins, %s Losses, %s Ties' % (wins, losses, ties))
  while True: # مدخلات المستخدم
    print('Enter your move: (r)ock (p)aper (s)cissors or (q)uit')
    playerMove = input()
    if playerMove == 'q':
      sys.exit() # الخروج من البرنامج
    if playerMove == 'r' or playerMove == 'p' or playerMove == 's':
      break # الخروج من حلقة مدخلات المستخدم
    print('Type one of r, p, s, or q.')
```

يستعمل برنامجنا حلقة `while` داخل حلقة `while`، أول حلقة هي حلقة اللعبة الأساسية، وتمثل دورًا من لعبة «حجرة ورقة مقص» في كل مرة تنفذ فيها تلك الحلقة.

أما حلقة التكرار الثانية فهي تطلب من اللاعب مدخلات، وستبقى تعمل حتى يدخل المستخدم r أو p أو s أو q التي ترمز إلى حجرة rock وورقة paper ومقص scissors على التوالي وبالترتيب، أما q فتعني أن اللاعب يريد إنهاء اللعبة quit، وفي تلك الحالة ستستدعى الدالة `sys.exit()` وسنتهي تنفيذ البرنامج.

إذا أدخل المستخدم r أو p أو s فسنخرج من حلقة التكرار الثانية عبر `break`، وإلا فسيعود التنفيذ إلى بداية حلقة التكرار ويذكر البرنامج اللاعب أن عليه إدخال r أو p أو s أو q.

```
# عرض ما اختاره اللاعب
if playerMove == 'r':
  print('ROCK versus...')
elif playerMove == 'p':
  print('PAPER versus...')
elif playerMove == 's':
  print('SCISSORS versus...')
```

يظهر البرنامج هنا ما اختاره اللاعب.

```
# عرض ما اختاره الحاسوب
randomNumber = random.randint(1, 3)
if randomNumber == 1:
  computerMove = 'r'
  print('ROCK')
elif randomNumber == 2:
  computerMove = 'p'
  print('PAPER')
elif randomNumber == 3:
  computerMove = 's'
  print('SCISSORS')
```

أما هنا فيظهر ما اختاره الحاسوب عشوائيًا. ولمّا كانت الدالة `random.randint()` تعيـد رقمًـا عشـوائيًا، فسنحتاج إلى تحويل الرقم الصحيح المخزن في المتغير `randomNumber` إلى حجرة أو ورقة أو مقص عبر البنيـة الشرطية `if` و `elif`. بالتالي سيخزن البرنامج ما اختاره الحاسوب في المتغير `computerMove` ثم يطبع رسالة نصية فيها الحركة المختارة.

```
    # عرض النتيجة الربح/الخسارة/التعادل
    if playerMove == computerMove:
      print('It is a tie!')
      ties = ties + 1
    elif playerMove == 'r' and computerMove == 's':
      print('You win!')
      wins = wins + 1
    elif playerMove == 'p' and computerMove == 'r':
      print('You win!')
      wins = wins + 1
    elif playerMove == 's' and computerMove == 'p':
      print('You win!')
      wins = wins + 1
    elif playerMove == 'r' and computerMove == 'p':
      print('You lose!')
      losses = losses + 1
    elif playerMove == 'p' and computerMove == 's':
      print('You lose!')
      losses = losses + 1
    elif playerMove == 's' and computerMove == 'r':
      print('You lose!')
      losses = losses + 1
```

وفي النهايـة، سـيوازن البرنـامج بين السلسـلتين النصـيتين الموجـودتين في المتغـيرين `playerMove` و`computerMove` ويظهر الناتج على الشاشة، وسـيزيد قيمـة أحـد المتغـيرات `wins` أو `losses` أو `ties` بمـا يناسب الحالة.

بعد أن يصل تنفيذ البرنامج إلى النهاية، سيعود لبداية حلقة التكرار الرئيسية ونلعب دورًا جديدًا من اللعبة.

2.11 أسئلة للتدريب

1. ما هي القيمتان التي يقبلها نوع البيانات المنطقي أو البولياني؟
2. ما هي العوامل المنطقية أو البولية الثلاثة؟
3. اكتب جدول الحقيقة لكل عامل منطقي (أي كل تركيب ممكن من القيم المنطقي مع كل معامل وما نتيجة كل تركيب)
4. ما هي نتيجة التعابير الآتية:

```
(5 > 4) and (3 == 5)
not (5 > 4)
(5 > 4) or (3 == 5)
not ((5 > 4) or (3 == 5))
(True and True) and (True == False)
(not False) or (not True)
```

5. ما هي عوامل المقارنة الستة؟
6. ما هو الفرق بين عامل المساواة وعامل الإسناد؟
7. اشرح ما هو الشرط وكيف تستخدمه.
8. أشر إلى الكتل البرمجية الثلاثة في الشيفرة الآتية:

```
spam = 0
if spam == 10:
print('eggs')
if spam > 5:
print('olive')
else:
print('steak')
print('spam')
print('spam')
```

9. اكتب شيفرة تطبع Hello إذا كانت القيمة `1` مخزنة في المتغير `spam`، و `World` إذا كانت القيمة `2` في المتغير `spam`، و `Greetings` فيما عدا ذلك.

10. ماذا عليك أن تضغط على لوحة المفاتيح إذا علقتَ في حلقة تكرار لا نهائية؟

11. ما الفرق بين `break` و `continue`؟

12. ما الفرق بين `range(10)` و `range(0, 10)` و `range(0, 10, 1)` في حلقة التكرار `for`؟

13. اكتب برنامجًا قصيرًا يطبع الأرقام من 1 إلى 10 باستخدام حلقة `for`. ثم اكتب برنامجًا مكافئًا لـه يطبـع الأرقام من 1 إلى 10 باستخدام حلقة `while`.

14. إذا كانت لديك دالة باسم `olive()` داخل وحدة باسم `spam`، كيف ستستدعي تلـك الدالـة في برنـامج خارجي بعد استيرادك للوحدة `spam`؟

2.12 تدريب إضافي

ابحث عن الدالتين `round()` و `abs()` في الإنترنت، وتعلم ماذا يفعلان، ثم جربهما في الصدفة التفاعلية.

2.13 الخلاصة

بعد أن تعلمنا كيفية كتابة شروط أو عبـارات تكـون نتيجتهـا `True` أو `False`، أصـبح بإمكاننـا كتابـة بـرامج تستطيع اتخاذ قرارات تحدد ما هي الشيفرات التي سينفذها البرنامج وأيها سيتخطاها.

أصـبح تسـتطيع تنفيـذ الشـيفرات مـرارًا وتكـرارًا عـبر حلقـات التكـرار، وستسـتفيد من العبـارتين `break` و`continue` للخروج من حلقة التكرار أو العودة إلى بدايتها مباشرةً.

تسمح لنا بنى التحكم بكتابة برامج أكثر ذكاءً، وسنتعلم كتابة شيفرات أفضل عبر نوع جديـد من بـنى التحكم الذي سنتعرف عليه في فصلنا القادم، ألا وهو الدوال functions.

والآن هل تستطيع تجربة الآتي؟

كتابة شيفرة تطبع Hello إذا كـانت القيمـة `1` مخزنـة في المتغـير `spam`، و `World` إذا كـانت القيمـة `2` في المتغير `spam`، و `Greetings` فيما عدا ذلك. كتابة برنامج قصـير يطبـع الأرقـام من 1 إلى 10 باسـتخدام حلقـة `for`. ثم اكتب برنامجًا مكافئًا له يطبع الأرقام من 1 إلى 10 باستخدام حلقة `while`.

دورة علوم الحاسوب

مميزات الدورة

- بناء معرض أعمال قوي بمشاريع حقيقية
- وصول مدى الحياة لمحتويات الدورة
- تحديثات مستمرة على الدورة مجانًا
- شهادة معتمدة من أكاديمية حسوب
- إرشادات من المدربين على مدار الساعة
- من الصفر دون الحاجة لخبرة مسبقة

اشترك الآن

3. الدوال في لغة بايثون

تعرفت في الفصول السابقة على الدوال `print()` و `input()` و `len()`، وأصبحت تعرف بتوفير بـايثون عدّة دوال مضمنة في اللغة built-in functions مثلها، لكنك تستطيع كتابة دوال خاصة بك أيضًا.

يمكنك القول أن الدالة هي برنامج صغير داخل برنامجك، ولكي نفهم سويةً كيف تعمل الـدوال فنحتـاج إلى إنشاء واحدة. أدخل الشيفرة الآتية في المحرر لديك وسم الملف باسم `helloFunc.py`:

```
❶ def hello():
❷   print('Howdy!')
    print('Howdy!!!')
    print('Hello there.')

❸ hello()
  hello()
  hello()
```

أول سطر هو عبارة `def` التي تعرّف دالـة `hello()`، والشـيفرة الـتي تلي عبـارة `def` هي جسـم الدالـة ❷، الذي يحتوي على الشيفرة التي ستنفذ حين استدعاء call الدالة، وليس حين تعريف الدالة.

الأسطر الثلاثة التي تحتوي على `hello()` ❸ هي استدعاءات الدالة، وعملية استدعاء الدالة هي كتابة اسمها متبوعًا بقوسين، وقد تمرر إليها بعض الوسائط arguments بين القوسين.

حينما يصل تنفيذ البرنامج إلى استدعاء هذه الدوال، فسينتقل التنفيذ إلى أول سطر في الدالة ويبدأ بتنفيذ الشيفرات الموجودة فيها حتى يصل إلى نهايتها، وحينها يعود التنفيذ إلى السطر البرمجي الذي استدعى الدالة، ثم يكمل البرنامج عمله كالمعتاد.

ولأننا استدعينا الدالة `hello()` ثلاث مرات، فإن الشيفرة الموجودة داخل الدالة `hello()` ستنفذ ثلاث مرات، وحينما تشغِّل البرنامج السابق سيظهر لديك:

```
Howdy!
Howdy!!!
Hello there.
Howdy!
Howdy!!!
Hello there.
Howdy!
Howdy!!!
Hello there.
```

أحد الأغراض الأساسية من تعريف الدوال هو تجميع الشيفرات مع بعضها والتي يمكن أن تنفذ عدة مرات، فلو لم تكن لدينا الدالة السابقة فسنحتاج إلى نسخ ولصق الشيفرة عدة مرات كما يلي:

```
print('Howdy!')
print('Howdy!!!')
print('Hello there.')
print('Howdy!')
print('Howdy!!!')
print('Hello there.')
print('Howdy!')
print('Howdy!!!')
print('Hello there.')
```

هنالك قاعدة عامة تقول أن عليك تفادي تكرار الشيفرات نفسها قدر الإمكان، لأنك إذا قررت مستقبلًا إجراء تغيير عليها -كأنك وجدت علّة وحللتها مثلًا- فعليك أن تجري هذا التغيير في كل مكان نسخت إليه تلك الدالة.

وكلما زادت خبرتك البرمجة وجدتَ نفسك تقلل من تكرار الشيفرات، مما يجعل برامجك أقصر وأسهل للقراءة وأسهل في التعديل والتحديث.

3.1 عبارة def مع معاملات

تذكر حينما كنت تستدعي الدالة `print()` أو `len()` كنت تمرر إليها قيمًا تسمى بالوسائط arguments، وذلك بكتابتها بين القوسين.

يمكنك أيضًا أن تعرف دوالك التي تقبل وسائط، وجرب هذا المثال في ملف باسم `helloFunc2.py`:

```
❶ def hello(name):
❷   print('Hello, ' + name)

❸ hello('Alice')
  hello('Bob')
```

ستبدو المخرجات كما يلي حين تشغيل البرنامج السابق:

```
Hello, Alice
Hello, Bob
```

لاحظ أن تعريفنا للدالة `hello()` في هذا المثال يحتوي على معامل باسم `name` ❶. المعاملات parameters هي المتغيرات التي تحتوي على قيمة الوسائط arguments. أي حينما نستدعي دالةً مع وسائط arguments فإن قيمة تلك الوسائط ستكون مخزنة في المعاملات parameters.

حين استدعاء الدالة `hello()` أول مرة سنمرر لها القيمة `'Alice'` ❸ وسينتقل التنفيذ لداخل الدالة، وستُضبَط قيمة المعامل `name` إلى القيمة `'Alice'` تلقائيًا، ومن ثم ستطبع هذه القيمة باستخدام الدالة `print()` ❷.

أحد الأمور التي من المهم تذكرها حول المعاملات هي أن القيمة المخزنة فيها ستنسى حين إنتهاء تنفيذ الدالة، فلو كتبت `print(name)` بعد استدعاء `hello('Bob')` في البرنامج السابق، فسيظهر لك الخطأ NameError لعدم وجود متغير باسم `name`، وذلك لأن برنامجنا سيحذف المتغير `name` بعد إنتهاء استدعاء الدالة `hello('Bob')` ولذا سنشير في `print(name)` إلى المتغير `name` الذي لن يكون موجودًا.

هذا يشبه كثيرًا كيفية حذف قيمة المتغيرات في برامجنا السابقة من الذاكرة حين انتهاء تنفيذها. وسنتحدث لاحقًا عن سبب حدوث ذلك بالتفصيل ضمن هذا الفصل، عند تحدثنا عما يسمى: النطاق المحلي local scope.

3.2 التعريف والاستدعاء والتمرير والوسائط والمعاملات!

كثرت علينا المصطلحات الجديدة كالتعريف define والاستدعاء call والتمرير pass والوسائط arguments والمعاملات parameters. لننظر سويةً إلى الشيفرة الآتية لمراجعتها:

```
❶ def sayHello(name):
    print('Hello, ' + name)
❷ sayHello('Abdullatif')
```

تعريف الدالة يعني إنشاءها، وكما في عبارة الإسناد `spam = 42` التي تنشِئ المتغير `spam` سنستعمل العبارة `def` لتعريف الدالة `sayHello()` ❶. السطر الذي فيه `sayHello('Abdullatif')` ❷ يستدعي الدالة التي أنشأناها، مما ينقل تنفيذ البرنامج إلى بداية الشيفرة الموجودة داخل الدالة، وسطر الاستدعاء السابق يمرر السلسلة النصية `'Abdullatif'` إلى الدالة، والقيمة التي تمرر إلى الدالة تسمى وسيطًا، وسيُسند الوسيط `'Abdullatif'` إلى المتغير المحلي المسمى `name`، وتسمى المتغيرات التي تحمل قيمة الوسائط الممررة إلى الدالة بالمعاملات.

من السهل الخلط بين المصطلحات السابقة، لكن حاول أن تستوعبها جيدًا لكي تفهم بقية الفصل بسهولة.

3.3 القيم المعادة وعبارة return

عندما تستدعي الدالة `len()` وتمرر إليها وسيطًا مثل `'Hello'` فستكون القيمة الناتجة من الدالة هي الرقم 5، الذي يمثل طول السلسلة النصية التي مررتها إليها. يمكننا القول عمومًا أن الناتج إحدى الدوال يسمى بالقيمة المعادة return value من تلك الدالة.

حينما تنشِئ دالةً باستخدام العبارة `def` فيمكنك أن تحدد ما هي القيمة المعادة من الدالة باستخدام العبارة `return`. تتألف عبارة `return` من:

- الكلمة المحجوزة `return`
- قيمة أو تعبير برمجي يجب أن تعيدها الدالة

حين استخدام تعبير برمجي مع عبارة `return` فستكون القيمة المعادة هي ناتج ذاك التعبير. مثلًا البرنامج الآتي يعرف دالةً تعيد سلسلةً نصيةً مختلفةً اعتمادًا على الرقم المُمرَّر إليها كوسيط.

احفظ الشيفرة الآتية في ملف باسم `magic8Ball.py`:

```
❶ import random


❷ def getAnswer(answerNumber):
  ❸ if answerNumber == 1:
      return 'It is certain'
    elif answerNumber == 2:
      return 'It is decidedly so'
    elif answerNumber == 3:
      return 'Yes'
    elif answerNumber == 4:
      return 'Reply hazy try again'
    elif answerNumber == 5:
      return 'Ask again later'
    elif answerNumber == 6:
      return 'Concentrate and ask again'
    elif answerNumber == 7:
      return 'My reply is no'
    elif answerNumber == 8:
      return 'Outlook not so good'
    elif answerNumber == 9:
      return 'Very doubtful'

❹ r = random.randint(1, 9)
❺ fortune = getAnswer(r)
❻ print(fortune)
```

حين يبدأ تنفيذ البرنامج السابق ستسورد بايثون الوحدة `random` ❶، ثم ستعرف الدالة `getAnswer()` ❷، ولأننا عرفنا الدالة ولم نستدعها فلن تنفذ الشيفرة الموجودة داخلها، بل سينتقل التنفيذ مباشرةً إلى استدعاء الدالة `random.randint()` التي مررنا إليها وسيطين هما `1` و `9` ❹ وسيعاد منها عدد عشوائي بين 1 و 9 (بما في ذلك الرقمين 1 و 9) وتخزن هذه القيمة في المتغير `r`.

تستدعى الدالة`getAnswer()` مع تمرير الوسيط `r` ❺، وينتقل التنفيذ إلى بداية الدالة `getAnswer()` ❸، وستخزن قيمة الوسيط `r` في المعامل `answerNumber`، ثم اعتمادًا على القيمة الموجودة في `answerNumber` فستعيد الدالة إحدى السلاسل النصية المعرفة مسبقًا، ومن ثم سيعود التنفيذ إلى النقطة التي استدعيت فيها الدالة `getAnswer()` ❺ وستسند السلسلة النصية المعادة إلى المتغير ذي الاسم `fortune`، الذي سيمرر بدوره إلى الدالة `print()` ❻ ويطبع على الشاشة.

لاحظ أنك تستطيع تمرير القيم المعادة من الدوال كوسيط مباشرةً إلى دوال أخرى، لذا يمكنك أن تختصر الأسطر الثلاثة الآتية:

```
r = random.randint(1, 9)
fortune = getAnswer(r)
print(fortune)
```

إلى السطر المكافئ:

```
print(getAnswer(random.randint(1, 9)))
```

تذكر أن التعابير البرمجية تتألف من قيم وعوامل، ويمكن استخدام استدعاءات الدوال في تعبير برمجي لأن الاستدعاء سيؤول إلى قيمة معادة من تلك الدالة.

3.4 القيمة None

هنالك قيمة في بايثون تسمى `None` وهي تمثل عدم وجود قيمة. والقيمة `None` هي القيمة الوحيدة لنوع البيانات NoneType، وقد تسمي لغات البرمجة الأخرى هذه القيمة بالاسم null أو nil أو undefined. وكما في القيم المنطقية `True` و `False` فيجب أن نكتب `None` بحرف N كبير.

يمكن أن تفيد هذه القيمة-التي-ليس-لها-قيمة حين الحاجة إلى استعمال قيمة لا تمثل شيئًا، فإحدى استخدامات `None` مثلًا هي القيمة المعادة من الدالة `print()` التي تطبع نصًا على الشاشة، لكنها لا تعيد أي قيمة على عكس دوال أخرى مثل `len()` أو `input()`، ولأن من المفترض أن يكون هنالك قيمة ناتجة لجميع استدعاءات الدوال في بايثون فاستدعاء `print()` سيعيد القيمة `None`.

لنجرب السطرين الآتيين في الصدفة التفاعلية لنرى ذلك:

```
>>> spam = print('Hello!')
Hello!
>>> None == spam
True
```

تضيف بايثون العبارة `return None` وراء الكواليس لأي دالة لا يكون لها عبارة `return` محددة، وهذا ما يشبه كيف تحتوي حلقات التكرار `while` و `for` على عبارة `continue` ضمنية في نهايتها. وستعاد القيمة `None` أيضًا.

يُعَد استخدام عبارة `return` دون قيمة، فقد كتبت الكلمة المفتاحية `return` كما هي.

3.5 وسطاء الكلمات المفتاحية والدالة ()print

تعرف أغلبية الوسائط بموضعها حين استدعاء الدالة، فمثلًا الاستدعاء `random.randint(1, 10)` مختلف عن الاستدعاء `random.randint(10, 1)`، فحينما نستدعي الدالة `random.randint(1, 10)` فستعيد لنا رقمًا صحيحًا بين 1 و 10 لأن أول وسيط هو الحد الأدنى من المجال والوسيط الثاني هو الحد الأقصى، بينما يسبب استدعاء `random.randint(10, 1)` خطأً.

لكن بدلًا من تعريف قيم الوسائط عبر موضعها، يمكن أن تعرف وسطاء الكلمات المفتاحية keyword arguments بوضع كلمة مفتاحية قبلها حين استدعاء الدالة، وتستخدم وسطاء الكلمات المفتاحية عادةً للمعاملات الاختيارية optional parameters.

فمثلًا تمتلك الدالة `print()` معاملين اختياريين هما `end` و `sep` لضبط ما الذي سيطبع بعد نهاية طبع الوسائط الممررة إليها وما الذي سيطبع بين تلك الوسائط على التوالي.

إذا شغلنا برنامجًا يحتوي على الشيفرة الآتية:

```
print('Hello')
print('World')
```

فسيكون الناتج كما يلي:

```
Hello
World
```

لاحظ أن السلسلتين النصيتين المطبوعتين مفصولتان بسطر وذلك لأن الدالة `print()` تضيف تلقائيًا محرف السطر الجديد newline character في نهاية السلسلة النصية التي تمرر إليها كوسيط. إلا أننا نستطيع ضبط قيمة الوسيط `end` لتغيير محرف السطر الجديد إلى سلسلة نصية مختلفة، فمثلًا إذا كتبت الشيفرة الآتية:

```
print('Hello', end='')
print('World')
```

فسيكون الناتج:

```
HelloWorld
```

سيطبع الناتج في سطر وحيد لعدم وجود محرف السطر الجديد بعد السلسلة النصية `'Hello'` لأننا مررنا سلسلة نصية فارغة، وهذا ما يفيدك إن أردت تعطيل الإضافة التلقائية للسطر الجديد في نهاية كل استدعاء للدالة `print()`.

إذا مررت عدة سلاسل نصية إلى الدالة `print()` فستفصل الدالة بينها تلقائيًا بفراغ واحد. جرب إدخال السطر الآتي في الصدفة التفاعلية:

```
>>> print('cats', 'dogs', 'mice')
cats dogs mice
```

يمكنك تبديل السلسلة النصية التي تفصل بين الوسائط التي ستطبع باستخدام الوسيط `sep` وتمرير سلسلة نصية مختلفة إليه، جرّب ذلك في الصدفة التفاعلية:

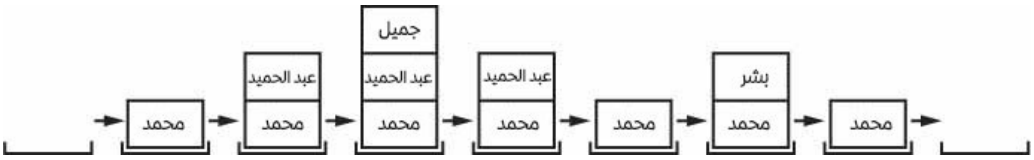
```
>>> print('cats', 'dogs', 'mice', sep=',')
cats,dogs,mice
```

يمكنك إضافة وسائط مفتاحية في الدوال التي تكتبها أيضًا، لكن عليك أن تتعلم أولًا عن القوائم list والقواميس dictionary التي سنشرحها في فصول لاحقة. لكن كل ما عليك معرفته الآن هو أن بعض الدوال لها معاملات اختيارية ويكون لها مفاتيح يمكن الوصول إليها لضبط قيمتها حين استدعاء الدالة.

3.6 مكدس الاستدعاء Call Stack

تخيل أنك تدردش مع أحد أصدقائك، فستتحدث عن صديقك محمد، ثم تتذكر قصة عن زميلك في العمل عبد الحميد، ثم تتذكر شيئًا عن ابن عمك جميل، وحينما تنتهي قصتك عن جميل تعود إلى حديثك عن عبد الحميد، ثم تعود وتتحدث عن محمد، وبعد ذلك تتذكر أخاك بشر، وتقص قصة عن بشر، ثم تعود وتكمل القصة الأصلية لمحمد.

تتبع محادثتك بنيةً شبيهةً بالمكدس stack، كما في الشكل الموالي، التي يكون فيها الموضوع الحالي في أعلى المكدس.



الشكل 20: مكدس القصص في دردشتك

وكما في محادثتك السابقة، عملية استدعاء دالة لا تؤدي إلى نقل التنفيذ إلى بداية الدالة التي جرى استدعاؤها، بل تتذكر بايثون ما هو السطر الذي استدعى تلك الدالة لكي تستطيع توفير القيمة المعادة من تلك الدالة إليه. وإذا استدعت تلك الدالة دوالًا أخرى فستعاد القيم الناتجة عن تلك الدوال إلى أماكن استدعائها الأصلية أولًا.

لنفهم ما يحدث بالتفصيل سنجرب المثال الآتي بعد حفظه في ملف باسم `abcdCallStack.py`:

```
def a():
  print('a() starts')
❶ b()
❷ d()
  print('a() returns')

def b():
  print('b() starts')
❸ c()
  print('b() returns')

def c():
❹ print('c() starts')
  print('c() returns')

def d():
  print('d() starts')
```

```
    print('d() returns')

❺ a()
```

سينتج البرنامج ما يلي حين تشغيله:

```
a() starts
b() starts
c() starts
c() returns
b() returns
d() starts
d() returns
a() returns
```

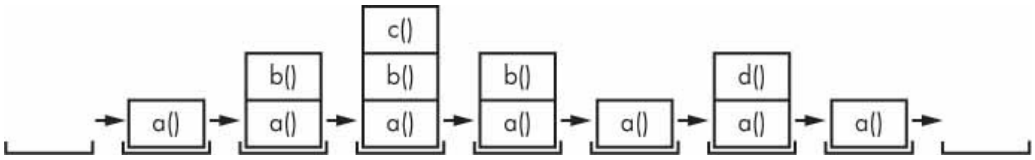
أعرف أن الفقرة الآتية متداخلة، لكن حاول أن تركز معي فيها: حين استدعاء `a()` ❺ فستستدعي `b()` ❶ التي بدورها ستستدعي `c()` ❸. والدالة `c()` لا تستدعي غيرها بل تعرض العبارة `c() starts` ❹ و `c()` `returns` قبل أن يعود التنفيذ إلى السطر الذي استدعاها في `b()` ❸.

بعد أن يعود التنفيذ إلى الشيفرة في `b()` التي استدعت `c()` فستنتهي الدالة `b()` وتطبع `b()` `returns` ثم يعود التنفيذ إلى السطر الذي استدعى `b()` في `a()` ❶.

سيستمر التنفيذ في الدالة `a()` وستستدعى الدالة `d()`، والتي تشبه الدالة `c()` في كونها لا تستدعي دالةً غيرها بل تطبع `d() starts` و `d() returns` قبل أن تعود إلى السطر الذي استدعاها في`a()` ثم سيكمل التنفيذ من هناك، وسيطبع آخر سطر من `a()` العبارة `a() returns` قبل أن ينتهي تنفيذ الدالة `a()` ونصل إلى نهاية البرنامج.

مكدس الاستدعاء call stack هو الآلية التي تستعملها بايثون لتذكر أين سيعود التنفيذ بعد انتهاء تنفيذ استدعاء كل دالة. لا يخزن مكدس الاستدعاء في متغير في برنامجك وإنما تتولى بايثون أمره خلف الكواليس. فحينما يستدعي برنامجك دالةً فستنشِئ بايثون كائن إطار frame object فوق مكدس الاستدعاء، وتخزن كائنات الإطار frame objects رقم السطر الذي استدعى الدالة لكي تعرف بايثون أين يجب أن تعيد القيمة الناتجة من ذاك الاستدعاء. إذا استدعيت دالة أخرى فستضع بايثون كائن إطار آخر في المكدس فوق السابق.

بعد إعادة استدعاء الدالة فستحذف بايثون كائن الإطار من أعلى المكدس وتكمل التنفيذ من السطر المخزن في ذاك الكائن. لاحظ أن كائنات الإطار تضاف وتحذف من أعلى المكدس وليس من أي مكان آخر. يوضح الشكل الموالي حالة مكدس الاستدعاء في البرنامج `abcdCallStack.py` حين استدعاء والعودة من كل دالة:



الشكل 21: حالة مكدس الاستدعاء في البرنامج abcdCallStack.py

كائنات الإطار لمكدس الاستدعاء في كل مرحلة من مراحل تنفيذ البرنامج `abcdCallStack.py`.

يمثل أعلى مكدس الاستدعاء الدالة التي يجري تنفيذها حاليًا، وحينما يكون المكدس فارغًا فسيكون التنفيذ في متن البرنامج وخارج جميع الدوال. لا حاجة إلى تعلم المزيد حول مكدس الاستدعاء لكي تكتب برامجك، لأنه يدخل في تفاصيل تقنية عميقة لا داعي لها حاليًا. من الكافي أن تفهم أن الدوال ستعيد القيم إلى السطر الذي استدعيت فيه؛ لكني آثرت شرح مكدس الاستدعاء هنا لأن فهمه سيسهل استيعاب مفاهيم المجالات العامة والمحلية التي سنشرحها في القسم الآتي.

3.7 المجالات العامة والمحلية

تكون المعاملات parameters والمتغيرات الموجودة داخل إحدى الدوال ضمن مجال محلي local scope. أما المتغيرات التي تسند قيمتها خارج جميع الدوال تكون موجودة في المجال العام global scope.

وبالتالي يسمى المتغير الموجود في مجال محلي بالمتغير المحلي local variable، بينما يسمى المتغير الموجود في المجال العام بالمتغير العام global variable؛ ويجب أن يكون المتغير عامًا أو محليًا، ولا يمكنه أن يكون كلاهما معًا.

يمكنك تخيل المجالات scopes على أنها حاوية للمتغيرات؛ فحينما ينتهي وجود أحد المجالات المحلية فستحذف جميع المتغيرات الموجودة فيه. لاحظ وجود مجال عام وحيد الذي سينُشَأ حينما يبدأ تنفيذ برنامجك وينتهي بإنتهاء تنفيذه. يمكنك التأكد من حذف المتغيرات في المجال العام بعد إنتهاء تنفيذ البرنامج بتجربة الوصول إلى المتغيرات من برنامج آخر.

يُنشَأ المجال المحلي في كل مرة تستدعى فيها إحدى الدوال. فتكون جميع المتغيرات المسندة ضمن الدالة موجودة في المجال المحلي. وحين إعادة قيمة return من الدالة فسينتهي وجود المجال المحلي وستحذف قيمة تلك المتغيرات؛ ولن يتذكر برنامج قيمة المتغيرات المخزنة من الاستدعاء السابق للدالة. تخزن المتغيرات المحلية أيضًا في كائنات الإطار frame objects في مكدس الاستدعاء call stack.

يهمنا معرفة المجال المستخدم لعدة أسباب:

- لا يمكن للشيفرات الموجودة في المجال العام أن تستعمل أي متغيرات محلية
- لكن يمكن للشيفرات في المجال المحلي الوصول إلى المتغيرات العامة

- الشيفرة في المجال المحلي لإحدى الدوال لا تستطيع استخدام أي متغيرات موجودة في المجال المحلي لدالة أخرى
- يمكنك استخدام الاسم نفسه لمتغيرات مختلفة على أن تكون موجودة في مجالات مختلفة. أي يمكن أن يسمى متغير محلي بالاسم `spam` مثلًا ويكون هنالك متغير عام بالاسم `spam` أيضًا.

السبب في امتلاك بايثون لمجالات مختلفة بدل من جعل جميع المتغيرات عامة هو أن بعض المتغيرات تعدل من شيفرة معينة ضمن دالة ما، وتتفاعل هذه الدالة مع بقية البرنامج عبر المعاملات parameters الممررة إليها وعبر القيمة المعادة منها؛ وهذا ما يقلل عدد الأسطر البرمجية التي تتعامل مع متغير ما وتسبب مشكلة برمجية، فإذا كان يحتوي برنامجك على متغيرات عامة فقط وحصل خطأ بسبب ضبط أحد المتغيرات إلى قيمة خطأ فسيصعب كثيرًا تتبع المشكلة في برنامجك، فقد يكون عدد الأسطر البرمجية بالمئات أو الآلاف التي قد تستطيع تعديل قيمة هذا المتغير؛ أما لو كانت العلة البرمجية بسبب قيمة خطأ لمتغير محلي فستعرف تحديدًا ما هي الأسطر البرمجية المسؤولة عن ضبط تلك القيمة وستحل المشكلة بسهولة وسرعة.

لا مشكلة في استخدام المتغيرات العامة في البرامج القصيرة سهولتها، لكنها من غير المستحسن الاعتماد على المتغيرات العامة حينما تكبر برامجك.

3.7.1 لا يمكن استخدام المتغيرات المحلية في المجال العام

أمعن النظر في البرنامج الآتي الذي سيتسبب بخطأ حين محاولة تشغيله:

```
def spam():
❶ eggs = 31337
spam()
print(eggs)
```

إذا جربت هذا البرنامج فسيبدو الناتج كما يلي:

```
Traceback (most recent call last):
 File "C:/test1.py", line 4, in <module>
  print(eggs)
NameError: name 'eggs' is not defined
```

سيحدث الخطأ بسبب وجود المتغير `eggs` داخل المجال المحلي المنشأ من الدالة `spam()` ❶. فبعد انتهاء تنفيذ الدالة `spam` فسيحذف المجال المحلي ولن يبقى هنالك أي متغير باسم `eggs`، وحينما يحاول البرنامج تشغيل السطر `print(eggs)` فستعطيك بايثون خطأً تقول فيه أن المتغير `eggs` غير معرف، وهذا

منطقي إذا فكرت مليًا بالأمر؛ فحينما يكون تنفيذ البرنـامج في المجـال العـام فلا توجـد أي مجـالات محليـة ولن تكون هنالك أي متغيرات محلية، وبالتالي لا يمكننا استخدام سوى المتغيرات العامة في المجال العام.

3.7.2 لا يمكن استخدام المتغيرات المحلية في مجالات محلية أخرى

ينشأ مجال محلي جديد في كل مرة تستدعى فيها إحدى الدوال، بما في ذلك حين استدعاء دالة ضمن دالـة أخرى. انظر إلى المثال الآتي:

```
def spam():
 ❶ eggs = 99
 ❷ olive()
 ❸ print(eggs)

def olive():
  steak = 101
 ❹ eggs = 0

❺ spam()
```

حين بدأ تشغيل البرنامج فستستدعى الدالة `spam()` ❺ وسينشأ مجال محلي، وسيضبط المتغير المحلي `eggs` ❶ إلى 99، ثم ستستدعى الدالة `olive()` ❷، ثم سينشأ مجال محلي جديد؛ فمن الممكن أن تكون عدة مجالات محلية موجودة جنبًـا إلى جنب. وسنضـبط قيمـة المتغـير المحلي `Steak` إلى 101، وسننشِـئ المتغـير المحلي `eggs` -المختلف كليًا عن المتغير الذي يحمل نفس الاسم في المجال المحلي للدالة `spam()`- ونضـبط قيمته إلى 0 ❹. حين إعادة الدالة `olive()` فسيحذف المجال المحلي المنشأ بسبب استدعائها، بما في ذلك المتغير `eggs` الخاص بها. وسيكمل تنفيذ البرنامج في الدالة `spam()` ليطبع لنا قيمة المتغـير `eggs` ❸؛ ولأن المجال المحلي الخاص بالدالة `spam()` ما يزال موجودًا فستكون قيمة `eggs` هي 99 كما ضـبطناها سـابقًا في ذلك المجال.

خلاصـة الكلام السـابق كلـه هـو أن المتغـيرات المحليـة الموجـودة في إحـدى الـدوال مفصـولة تمامًـا عن المتغيرات المحلية في دالة أخرى.

3.7.3 يمكن قراءة المتغيرات العامة من مجال محلي

أمعن النظر في المثال الآتي:

```
def spam():
  print(eggs)
eggs = 42
spam()
print(eggs)
```

حينما حاولنا طباعة المتغير eggs ضمن الدالة spam() بحثت بايثون عن متغير أو معامل باسم eggs في الدالة spam() لكنها لم تجد، فحينها ستعدّه إشارةً إلى المتغير العام eggs، ولهذا سيطبع البرنامج السابق القيمة 42 حين تنفيذه.

3.7.4 المتغيرات المحلية والعامة التي تحمل الاسم نفسه

من المقبول تمامًا من الناحية التقنية في بايثون استخدام نفس الاسم لمتغير في المجال العام وآخر في المجال المحلي؛ لكن لتسهيل مقروئية الشيفرة، حاول تجنب فعل ذلك. لترى ما سيحدث فجرب الشيفرة الآتية:

```
def spam():
❶ eggs = 'spam local'
  print(eggs)    # 'spam local'

def olive():
❷ eggs = 'olive local'
  print(eggs)    # 'olive local'
  spam()
  print(eggs)    # 'olive local'

❸ eggs = 'global'
olive()
print(eggs)        # 'global'
```

سيظهر الناتج الآتي حينما تجرب تشغيل البرنامج السابق:

```
olive local
spam local
olive local
global
```

هنالك ثلاثة متغيرات مختلفة في البرنامج، لكنها كلها مسماة `eggs`، وهي كما يلي:

- ❶ متغير باسم `eggs` موجود في المجال المحلي للدالة `spam()`.
- ❷ متغير باسم `eggs` موجود في المجال المحلي للدالة `olive()`.
- ❸ متغير باسم `eggs` موجود في المجال العام.

ولأن هذه المتغيرات المختلفة لها نفس الاسم فسيكون من العسير تتبع أيها يستعمل الآن؛ لذا تجنب استخدام نفس الاسم لأكثر من متغير.

3.8 العبارة البرمجية global

إذا أردت تعديل قيمة متغير عام ضمن دالة، فعليك استخدام العبارة البرمجية `global`. فإذا كان لديك سطر يشبه `global eggs` في بداية إحدى الدوال فهذا سيخبر بايثون أن «المتغير `eggs` في هذه الدالة يشير إلى متغير عام، ولا حاجة إلى إنشاء متغير محلي بهذا الاسم». فمثلًا جرب الشيفرة الآتية:

```
def spam():
    ❶ global eggs
    ❷ eggs = 'spam'

eggs = 'global'
spam()
print(eggs)
```

استدعاء الدالة `print()` سيؤدي إلى إظهار الناتج الآتي:

```
spam
```

ولأننا صرحنا بأن المتغير `eggs` هو عام global في بداية الدالة `spam()` ❶، فعندما نضبط من `eggs` إلى `'spam'` ❷ فستجرى عملية الإسناد إلى المتغير `eggs` العام، ولن ينشأ أي متغير محلي.

هنالك أربع قواعد لمعرفة إن كان المتغير في المجال المحلي أم العام:

- إذا كان المتغير مستخدمًا في المجال العام، أي خارج جميع الدوال، فسيكون متغيرًا عامًا دومًا.
- إذا كانت هنالك عبارة `global` في إحدى الدوال، فسيكون المتغير عامًا في تلك الدالة.
- خلاف ذلك إذا استخدم المتغير في عبارة الإسناد داخل دالة ما، فسيكون متغيرًا محليًا.
- لكن إن لم يستخدم ذاك المتغير في عبارة إسناد داخل الدالة فسيكون متغيرًا عامًا.

لتأخذ فكرة أفضل عن هذه القواعد فاكتب البرنامج الآتي في محرر الشيفرات واحفظه وجربه:

```
def spam():
 ❶ global eggs
   eggs = 'spam' # هذا المتغير عام

def olive():
 ❷ eggs = 'olive' # هذا المتغير محلي
def Steak():
 ❸ print(eggs) # هذا المتغير عام

eggs = 42 # هذا المتغير عام
spam()
print(eggs)
```

سيكون المتغير `eggs` في الدالة `spam()` عامًا لوجود العبارة `global` في بداية الدالة ❶، وسيكون المتغير `eggs` محليًا في الدالة `olive()` لاستعماله في عبارة إسناد في تلك الدالة ❷، وسيكون `eggs` عامًا في `steak()` ❸ لعدم استخدامه في عبارة إسناد أو العبارة `global`. إذا شغلت البرنامج السابق فستكون النتيجة:

```
spam
```

كقاعدة عامة، سيكون المتغير في دالةٍ ما إما عامًا أو محليًا، ولا يمكن استخدام متغير محلي في دالة باسم `eggs` ثم استخدام متغير عام بنفس الاسم لاحقًا في الدالة ذاتها.

> **ملاحظة**: إذا أردت تعديل قيمة مخزنة في متغير عام ضمن دالة فيجب عليك دومًا استخدام العبارة `global`.

إذا حاولت استخدام متغير محلي ضمن دالة قبل أن تسند قيمةً له كما في البرنامج الآتي، فستظهر لك رسالة خطأ:

```
def spam():
  print(eggs) # خطأ!
❶ eggs = 'spam local'

❷ eggs = 'global'
spam()
```

ستظهر رسالة الخطأ الآتية إذا حاولت تجربة البرنامج السابق:

```
Traceback (most recent call last):
 File "C:/sameNameError.py", line 6, in <module>
  spam()
 File "C:/sameNameError.py", line 2, in spam
  print(eggs) # خطأ!
UnboundLocalError: local variable 'eggs' referenced before assignment
```

يظهر الخطأ لأن بايثون سترى عبارة إسناد للمتغير `eggs` ضمن الدالة `spam()` ❶ وبالتالي ستعد المتغير على أنه محلي، لكننا نحاول طباعة قيمة `eggs` قبل إسناد أي قيمة له، أي أن المتغير المحلي `eggs` غير موجود، فسيظهر الخطأ ولن تستعمل بايثون المتغير العام `eggs` ❷.

3.8.1 تعامل مع الدوال على أنها «صناديق سوداء»

عادةً كل ما تحتاج إليه لاستخدام دالة هو معرفة ما هي المدخلات (أي ما هي المعاملات التي تأخذها) وما هي المخرجات؛ فلا حاجة إلى أن تثقل على نفسك بمعرفة كيف تعمل شيفرة تلك الدالة. فحاول أن تنظر إلى الدوال نظرة شاملة عالية المستوى، وبإمكانك معاملتها على أنها «صندوق أسود».

هذه الفكرة أساسية في البرمجة الحديثة، وسترى عدة وحدات في الفصول اللاحقة من هذا الكتاب فيها دوال مكتوبة من مبرمجين آخرين، وصحيح أنك تستطيع النظر إلى الشيفرة المصدرية لها لكن لا حاجة إلى أن تفهم كيف تعمل لكي تستعملها؛ ولأن من المستحسن كتابة الدوال دون أن تتعامل مع المتغيرات العامة فلا تقلق حول تفاعل الدوال مع المتغيرات الموجودة في المجال العام لبرنامجك.

3.9 التعامل مع الاستثناءات

حينما يحدث خطأ -أو بتعبير أدق «استثناء exception»- في برنامجك فهذا يعني توقف عملية التنفيذ كلها. ولا تريد أن يحدث ذلك عمليًا في البرامج الحقيقية، وإنما تريد أن يحس برنامجك بوجود الأخطاء ويتعامل معها ثم يكمل تنفيذه بسلام. فالبرنامج الآتي يتسبب بخطأ القسمة على صفر. جربه:

```
def spam(divideBy):
  return 42 / divideBy


print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

عرفنا الدالة `spam()` ومررنا إليها وسيطًا بقيم مختلفة وطبعنا قيمة قسمة العدد 42 على القيمة الممررة. هذا هو ناتج تنفيذ الشيفرة السابقة:

```
21.0
3.5
Traceback (most recent call last):
 File "C:/zeroDivide.py", line 6, in <module>
  print(spam(0))
 File "C:/zeroDivide.py", line 2, in spam
  return 42 / divideBy
ZeroDivisionError: division by zero
```

يظهر الاستثناء `ZeroDivisionError` حينما نقسم عددًا على صفر. ويظهر لنا رقم السطر الذي يسبب هذا الاستثناء، وستعرف منه أن العبارة `return` في الدالة `spam()` هي من تسبب الخطأ.

يمكن التعامل مع الاستثناءات باستخدام العبارتين `try` و `except`. إذ نضع الشيفرة التي قد تسبب خطأ أو استثناءً ضمن قسم العبارة `try`، وسينتقل تنفيذ البرنامج إلى بداية القسم الذي يلي العبارة `except` في حال حدوث استثناء.

يمكنك وضع الشيفرة التي قد تسبب بخطأ القسمة على الصفر ضمن كتلة `try` واستخدام كتلة `except` للتعامل مع حدوث الخطأ:

```
def spam(divideBy):
  try:
    return 42 / divideBy
  except ZeroDivisionError:
    print('Error: Invalid argument.')

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

حينما تتسبب الشـيفرة الموجـودة ضـمن `try` باسـتثناء، فسـينتقل تنفيـذ البرنـامج مباشـرةً إلى الشـيفرة الموجودة في `except`، وبعد تنفيذ تلك الشيفرة فسيكمل تنفيذ البرنامج بشكل طبيعي. سـتكون نتيجـة تنفيـذ الشيفرة السابقة هي:

```
21.0
3.5
Error: Invalid argument.
None
42.0
```

لاحظ أن أية أخطاء تحدث أثناء استدعاءات الدوال ضمن كتلة `try` فستعالج أيضًا. جرب البرنامج الآتي التي يستدعي الدالة `spam()` ضمن كتلة `try`:

```
def spam(divideBy):
  return 42 / divideBy
try:
  print(spam(2))
  print(spam(12))
  print(spam(0))
  print(spam(1))
except ZeroDivisionError:
  print('Error: Invalid argument.')
```

سيكون الناتج كما يلي:

```
21.0
3.5
Error: Invalid argument.
```

سبب عدم تنفيذ `print(spam(1))` هو انتقال التنفيذ إلى الشيفرة الموجودة في كتلة `except` مباشـرةً، ولن تعود لإكمال بقية كتلة `try`، بل ستكمل تنفيذ بقية البرنامج كالمعتاد.

3.10 برنامج قصير لرسم زكزاك

لنستعمل المفاهيم البرمجية التي تعلمناها حتى الآن لإنشاء برنـامج حـركي بسـيط. سينشِـئ هـذا البرنـامج شكل زكزاك إلى أن يوقفه المسـتخدم بالضـغط على زر Stop في محـرر Mu أو بالضـغط على `Ctrl+C`. سـيبدو ناتج البرنامج بعد تنفيذه كما يلي:

```
 ********
 ********
 ********
 ********
********
 ********
 ********
 ********
 ********
```

اكتب الشيفرة الآتية واحفظها في ملف باسم `zigzag.py`:

```
import time, sys
indent = 0 # كم فراغًا نضع كمسافة بادئة
indentIncreasing = True # هل ستزيد المسافة البادئة أم لا
try:
  while True: # حلقة تكرار البرنامج الأساسية
    print(' ' * indent, end='')
```

```
    print('********')
    time.sleep(0.1) # توقف لعُشر ثانية


    if indentIncreasing:
      # زيادة المسافة البادئة
      indent = indent + 1
      if indent == 20:
        # تغيير الاتجاه
        indentIncreasing = False
    else:
      # إنقاص عدد الفراغات
      indent = indent - 1
      if indent == 0:
        # تغيير الاتجاه
        indentIncreasing = True
except KeyboardInterrupt:
  sys.exit()
```

لننظر إلى الشيفرة سطرًا بسطر بدءًا من الأعلى.

```
import time, sys
indent = 0 # كم فراغًا نضع كمسافة بادئة

indentIncreasing = True # هل ستزيد المسافة البادئة أم لا
```

في البداية علينا أن نستورد الوحدتين `time` و `sys`، وسيستخدم برنامجنا متغيرين اثنين: المتغـير `indent` الذي يتتبع كم فراغًا يجب أن نضع كمسافة بادئة قبل النجوم الثمانيـة، و `indentIncreasing` الـذي يحتـوي على قيمة منطقية بوليانية لتحدد إذا كانت المسافة البادئة ستزيد أم تنقص.

```
try:
  while True: # حلقة تكرار البرنامج الأساسية
    print(' ' * indent, end='')
    print('********')
    time.sleep(0.1) # توقف لعُشر ثانية
```

ثم وضعنا بقية البرنامج داخل عبارة `try`؛ فحينما يضغط المستخدم على `Ctrl+C` أثناء تشغيل برنامج بايثون فستطلق الاستثناء `KeyboardInterrupt`، وإذا لم تكن هنالك عبارة `try-except` لمعالجة الاستثناء فسينهار البرنامج وتظهر رسالة خطأ قبيحة.

لتفادي ذلك سنعالج الاستثناء `KeyboardInterrupt` بأنفسنا باستدعاء الدالة `sys.exit()` (هذه الشيفرة موجودة بعد نهاية كتلة `try`).

ستعمل حلقة التكرار اللانهائية `while True:` على تكرار التعليمات الموجودة في برنامجنا للأبد، وقد استخدمنا التعبير `' ' * indent` لطباعة العدد الصحيح من المسافات البادئة، لكننا لا نريد أن ننتقل إلى سطر جديد بعد تلك الفراغات فنمرر الوسيط `end=''` إلى الدالة `print()`. الاستدعاء الثاني للدالة `print()` سيطبع لنا 8 نجوم.

لم نشرح الدالة `time.sleep()` بعد، لكن يكفي القول أنها توقف تشغيل البرنامج مؤقتًا لعُشر ثانية 0.1.

```
    if indentIncreasing:
      # زيادة المسافة البادئة
      indent = indent + 1
      if indent == 20:
        # تغيير الاتجاه
        indentIncreasing = False
```

ثم سنعدل مقدار المسافات البادئة للمرة القادمة التي تنفذ فيها حلقة `while`. فإذا كان `indentIncreasing` هو `True` فسنضيف واحد إلى `indent`. لكن حينما تصل المسافة البادئة إلى 20 فنرغب بتقليل المسافة البادئة:

```
    else:
      # إنقاص عدد الفراغات
      indent = indent - 1
      if indent == 0:
        # تغيير الاتجاه
        indentIncreasing = True
```

أما إذا كانت `indentIncreasing` هي `False` فسننقص واحد من المتغير `indent`، وحينما تصل قيمته إلى 0 فسنحتاج إلى زيادة المسافات البادئة مجددًا، وفي كلتا الحالتين سيعود تنفيذ البرنامج إلى بداية حلقة التكرار لطباعة النجوم مجددًا.

```
except KeyboardInterrupt:
  sys.exit()
```

إذا ضغط المستخدم على `Ctrl+C` في أي مرحلة من مراحل تنفيذ حلقة التكرار الموجودة داخل كتلة `try` فسيطلق الاستثناء `KeyboardInterrrupt` ثم يعالج في عبارة `except`، سيستمر تنفيذ البرنامج داخل كتلة `expect` الذي سيشغل الدالة `sys.exit()` لإنهاء البرنامج. وعلى الرغم من أن حلقة التكرار لانهائية، لكننا نوفر طريقة آمنة لإنهاء تشغيل التطبيق من المستخدم.

3.11 أسئلة للتدريب

1. لماذا من المهم استخدم الدوال في برامجك؟
2. متى تنفذ الشيفرة داخل الدالة: عند تعريفها أم عند استدعائها؟
3. ما هي العبارة البرمجية التي تنشِئ دالةً؟
4. ما الفرق بين الدالة واستدعاء الدالة؟
5. كم مجالًا عامًا يكون في برنامج بايثون عادةً؟ وكم مجالًا محليًا؟
6. ماذا يحدث للمتغيرات في المجال المحلي حينما تعيد الدالة قيمةً ما؟
7. ما هي القيمة المعادة؟ وهل يمكن استخدام تعبير برمجي كقيمة معادة؟
8. إذا لم تكن هنالك عبارة `return` في الدالة، ما هي القيمة المعادة من استدعاء تلك الدالة؟
9. كيف يمكنك أن تجعل متغيرًا في دالة يشير إلى متغير في المجال العام؟

10. ما هو نوع البيانات للقيمة `None`؟

11. ماذا تفعل العبارة `import areallyourpetsnamederic`؟

12. إذا كان لديك دالة باسم `olive()` في وحدة باسم `spam`، كيف ستستدعيها بعد استيراد الوحدة `spam`؟

13. كيف تحمي برنامجك من الانهيار حال حدوث استثناء؟

14. ماذا تحتوي العبارة `try` والعبارة `except`؟

3.12 مشاريع للتدريب

لكي تتدرب، اكتب برامج لتنفيذ المهام الآتية.

3.12.1 معضلة كولاتز The Collatz Sequence

اكتب دالةً باسم `collatz()` تقبل معاملًا واحدًا اسمه `number`، إذا كان `number` زوجيًا فستطبع الدالة `number // 2` ثم تعيد تلك القيمة، وإذا كان العدد `number` فرديًا فستطبع وتعيد ناتج `number + 1 * 3`.

ثم اكتب برنامجًا يسمح للمستخدم بإدخال رقم صحيح واستمر باستدعاء الدالة `collatz()` على ذاك الرقم حتى تعيد الدالة القيمة `1`.

(ستعمل هذه الدالة لجميع الأعداد الصحيحة، وستكون النتيجة دومًا `1`! لا يعرف علماء الرياضيات تحديدًا لماذا، لكن برنامجك هو تطبيق عملي على معضلة كولاتز، حتى أن بعضهم يطلق عليها تسمية «أبسط معضلة رياضية مستحيلة».)

تذكر أن تحول القيمة المعادة من `input()` إلى رقم صحيح عبر الدالة `int()`، وإلا فستعدها بايثون على أنها سلسلة نصية.

> **تلميحة**: يكون العدد `number` زوجيًا إذا كان باقي القسمة على 2 هو 0 أي `number % 2 == 0` وفرديًا إذا كان `number % 2 == 1`.

يجب أن يبدو شكل تنفيذ البرنامج كما يلي:

```
Enter number:
3
10
5
16
```

```
8
4
2
1
```

3.12.2 التحقق من المدخلات

استعمل عبارتي `try` و `except` إلى المشروع السابق لتعرف إن كتب المستخدم سلسلةً نصيةً لا تحول إلى رقم.

عادةً ما تطلق الدالة `int()` الاستثناء ValueError إذا لم نمرر إليها سلسلةً نصيةً تحول إلى رقم مثل `int('puppy')`؛ ثم اطبع رسالة في كتلة `except` تخبر المستخدم أن عليه إدخال رقم صحيح.

3.13 الخلاصة

الدوال هي طريقة أساسية لتجزئة شيفراتك إلى مجموعات، ولأن المتغيرات داخل الدوال تكون في مجال محلي خاص بها فلا تؤثر الشيفرات الموجودة في إحدى الدوال على الأخرى، وبالتالي تقل الشيفرات المسؤولة عن تغيير قيمة أحد المتغيرات وبالتالي تسهل عملية تنقيح البرنامج ومعرفة الأخطاء. الدوال هي أداة رائعة لتنظيم شيفراتك، ويمكنك أن تفكر فيها على أنها صناديق سوداء: فهي تقبل المدخلات على شكل معاملات وتخرج النتائج على شكل قيم معادة، والشيفرات داخلها لا تؤثر على بقية الدوال.

تعلمنا استخدام العبارتين `try` و `except` التي تتولى معالجة الاستثناءات، فكان حدوث أي خطأ في البرنامج سيؤدي إلى انهياره كما رأينا في الفصول السابقة، لكن بتعلمنا لمعالجة الاستثناءات أصبح بإمكاننا بناء تطبيقات تتعامل مع الأخطاء الشائعة دون مشاكل.

4. القوائم والصفوف في لغة بايثون

موضوع آخر عليك أن تفهمه جيدًا قبل أن تبدأ جديًا بكتابة البرامج هو أنواع البيانات خصوصًا القوائم Lists والصفوف tuples.

قد تحتوي القوائم والصفوف على قيم متعددة، مما يجعل كتابة البرامج التي تعالج مقدرًا كبيرًا من البيانات أمرًا سهلًا. ولأن القوائم تستطيع احتواء قوائم أخرى فيها فيمكنك استخدامها لترتيب البيانات ترتيبًا هيكليًا.

سنناقش في هذا الفصل أساسيات القوائم، وسنتحدث عن التوابع methods، وهي دوال مرتبطة بنوع بيانات معين تجري عمليات عليه. ثم سنتحدث باختصار عن أنواع البيانات المتسلسلة الأخرى مثل الصفوف tuples والسلاسل النصية strings، وكيف تقارن مع بعضها بعضًا. وسنغطي في الفصل القادم نوعًا جديدًا من البيانات وهو القاموس dictionary.

4.1 نوع البيانات list

القائمة هي قيمة تحتوي على قيم أخرى متعددة داخلها بترتيب متسلسل. والمصطلح «قيمة القائمة list value» يشير إلى القائمة نفسها، والتي هي القيمة التي يمكن أن تخزن في متغير أو تمرر إلى دالة كغيرها من القيم، ولا تشير إلى القيم الموجودة داخل القائمة.

تبدو القائمة بالشكل التالي: `['cat', 'bat', 'rat', 'elephant']` وكما كنّا نكتب السلاسل النصية محاطةً بعلامتَي اقتباس لتحديد متى تبدأ وتنتهي السلسلة النصية، فتبدأ القائمة بقوس مربع وتنتهي بقوس مربع آخر `[]`. وتسمى القيم داخل القائمة بعناصر القائمة items، ويفصل بين عناصر القائمة بفاصلة.

يمكنك إدخال ما يلي في الصدفة التفاعلية للتجربة:

```
>>> [1, 2, 3]
[1, 2, 3]
>>> ['cat', 'bat', 'rat', 'elephant']
['cat', 'bat', 'rat', 'elephant']
>>> ['hello', 3.1415, True, None, 42]
['hello', 3.1415, True, None, 42]
❶ >>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam
['cat', 'bat', 'rat', 'elephant']
```

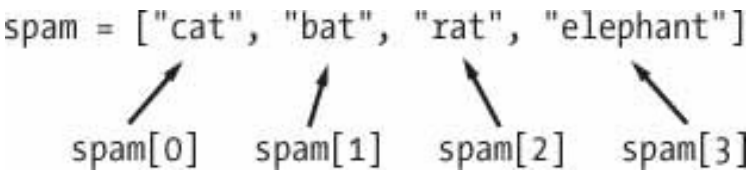
المتغير `spam` ❶ له قيمة واحدة، وهي قيمة القائمة، ولكن القائمة نفسها تحتوي على عناصر أخرى. لاحظ أن القيمة `[]` تعني قائمة فارغة لا تحتوي على قيم مثلها كمثل السلسلة النصية الفارغة `''`.

4.1.1 الوصول إلى عناصر القائمة عبر الفهرس

لنقل أن لديك القائمة `['cat', 'bat', 'rat', 'elephant']` مخزنةً في متغير باسم `spam`، حينها ستكون نتيجة التعبير `spam[0]` هي `'cat'` ونتيجة التعبير `spam[1]` هي `'bat'` وهلم جرًا للبقية.

العدد الصحيح الموجود داخل الأقواس المربعة يسمى فهرسًا index، والقيمة الأولى في القائمة يشار إليها بالفهرس `0`، والقيمة الثانية بالفهرس `1`، والثالثة بالفهرس `2` ...إلخ.

يُظهر التالي قائمةً مسندةً إلى المتغير `spam` مع توضيح فهارس كل قيمة فيها. لاحظ أن فهرس العنصر الأول هو `0` ويكون فهرس آخر عنصر مساويًا لطول القائمة ناقص واحد. أي أن الفهرس `3` في قائمةٍ لها أربع قيم يشير إلى آخر عنصر.



الشكل 22: قائمة مخزنة في متغير مع فهارس كل عنصر فيها

على سبيل المثال، أدخل التعابير البرمجية الآتية في الصدفة التفاعلية وابدأ بضبط قيمة المتغير spam:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0]
'cat'
>>> spam[1]
'bat'
>>> spam[2]
'rat'
>>> spam[3]
'elephant'
>>> ['cat', 'bat', 'rat', 'elephant'][3]
'elephant'
❶ >>> 'Hello, ' + spam[0]
❷ 'Hello, cat'
>>> 'The ' + spam[1] + ' ate the ' + spam[0] + '.'
'The bat ate the cat.'
```

لاحظ أن نتيجة التعبير `'Hello, ' + spam[0]` ❶ هي `'Hello, ' + 'cat'` ذلك لأن نتيجة `spam[0]` هي `'cat'`, وبالنهاية ستكون نتيجة التعبير هي السلسلة النصية `'Hello, cat'` ❷.

ستحصل على رسالة الخطأ IndexError إذا استخدمت فهرسًا يتجاوز عدد عناصر القائمة.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[10000]
Traceback (most recent call last):
 File "<pyshell#9>", line 1, in <module>
  spam[10000]
IndexError: list index out of range
```

يجب أن تكون الفهارس أعدادًا صحيحةً فقط، إذ لا يمكن أن تقبل بالأعداد العشرية؛ فالمثال الآتي يتسبب بخطأ TypeError:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1]
'bat'
>>> spam[1.0]
Traceback (most recent call last):
 File "<pyshell#13>", line 1, in <module>
  spam[1.0]
TypeError: list indices must be integers or slices, not float
>>> spam[int(1.0)]
'bat'
```

يمكن أن تحتوي القائمة على قوائم أخرى عناصر لها، ويمكن الوصول إلى القيم بإدخال أكثر من فهرس:

```
>>> spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]
>>> spam[0]
['cat', 'bat']
>>> spam[0][1]
'bat'
>>> spam[1][4]
50
```

يدل الفهرس الأول على أي عنصر من القائمة الأولى يجب استخدامه، والفهرس الثاني يدل على العنصر الموجود في القائمة الثانية، فمثلًا `spam[0][1]` يطبع `'bat'`، وهي القيمة الثانية من القائمة الموجودة في الفهرس الأول.

وإذا استخدمت فهرسًا واحدًا فسيطبع البرنامج القائمة الموجودة في ذلك الفهرس كاملةً.

4.1.2 الفهارس السالبة

صحيحٌ أن أرقام الفهارس تبدًا من 0، لكنك تستطيع استخدام الأعداد الصحيحة السالبة قيمًا للفهارس. فالقيمة `-1` تشير إلى آخر عنصر في القائمة، والقيمة `-2` تشير إلى العنصر ما قبل الأخير ...إلخ.

جرب ما يلي في الصدفة التفاعلية:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[-1]
'elephant'
>>> spam[-3]
'bat'
>>> 'The ' + spam[-1] + ' is afraid of the ' + spam[-3] + '.'
'The elephant is afraid of the bat.'
```

4.1.3 الحصول على قائمة من قائمة أخرى عبر التقطيع slice

ستحصل على قيمة واحدة حين استخدام الفهارس، لكن التقطيع slice يعيد أكثر من قيمة من تلك القائمة على شكل قائمة جديدة؛ ونكتبه ضمن القوسين المربعين كما في الفهارس لكن سنضع عددين صحيحين يفصل بينهما بنقطتين رأسيتين `:`، لاحظ الاختلاف بينهما:

- `spam[2]` ينتج عنصرًا واحدًا موجودًا في الفهرس المحدد.
- `Spam[1:4]` ينتج قائمة فيها أكثر من عنصر.

يكون أول عدد صحيح حين التقطيع هو مكان بدء القطع، والعدد الثاني هو مكان نهاية القطع، وستصل القائمة المقتطعة إلى فهرس العنصر الثاني دون تضمينه في الناتج، وستكون نتيجة القطع هي قائمة جديدة:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0:4]
['cat', 'bat', 'rat', 'elephant']
>>> spam[1:3]
['bat', 'rat']
>>> spam[0:-1]
['cat', 'bat', 'rat']
```

يمكنك اختصارًا أن تزيل أحد الفهرسين من عملية التقطيع مع الإبقاء على النقطتين الرأسيتين `:`، فإزالة الفهرس الأول تماثل استخدام الفهرس `0` وتشير إلى بداية القائمة، وإزالة الفهرس الثاني تماثل تمرير طول القائمة كاملًا وبالتالي ستنتهي عملية القطع عند نهاية القائمة:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[:2]
['cat', 'bat']
>>> spam[1:]
```

```
['bat', 'rat', 'elephant']
>>> spam[:]
['cat', 'bat', 'rat', 'elephant']
```

4.1.4 الحصول على طول القائمة عبر الدالة ()len

ستعيد الدالة `len()` عدد عناصر قائمة تُمرَّر إليها، وهي تشبه إحصاء عدد المحارف في سلسلة نصية:

```
>>> spam = ['cat', 'dog', 'moose']
>>> len(spam)
3
```

4.1.5 تغيير القيم في قائمة عبر الفهارس

تعودنا أن قيمة المتغير تكون على يسار عامل الإسناد، كما في `spam = 42`، ويمكننا فعل المثل مع القوائم بكتابة فهرس العنصر الذي نريد تغيير قيمته، مثلًا `spam[1] = 'aardvark'` يعني أسند القيمة الموجودة في الفهرس `1` في القائمة `spam` إلى السلسلة النصية `'aardvark'`:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1] = 'aardvark'
>>> spam
['cat', 'aardvark', 'rat', 'elephant']
>>> spam[2] = spam[1]
>>> spam
['cat', 'aardvark', 'aardvark', 'elephant']
>>> spam[-1] = 12345
>>> spam
['cat', 'aardvark', 'aardvark', 12345]
```

4.1.6 جمع القوائم Concatenation وتكرارها Replication

يمكن أن تجمع القوائم وتكرر كما في السلاسل النصية، فالعامل `+` يجمع بين قائمتين لإنشاء قائمة جديدة، والعامل `*` يستخدم لتكرار سلسلة نصية عددًا من المرات:

```
>>> [1, 2, 3] + ['A', 'B', 'C']
[1, 2, 3, 'A', 'B', 'C']
>>> ['X', 'Y', 'Z'] * 3
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
```

```
>>> spam = [1, 2, 3]
>>> spam = spam + ['A', 'B', 'C']
>>> spam
[1, 2, 3, 'A', 'B', 'C']
```

4.1.7 إزالة القيم من القوائم عبر عبارة del

تستخدم العبارة `del` لحذف قيم معينة من قائمة. جميع القيم الموجودة في القائمة بعد العنصر المحذوف سيتغير فهرسها ويصبح أقل بمقدار 1. مثلًا:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat']
```

يمكن أن تستعمل العبارة `del` على المتغيرات العادية أيضًا، وإذا حاولت استخدام أحد المتغيرات بعد حذفه فستحصل على خطأ NameError لأن المتغير لم يعد موجودًا. لكن عمليًا من النادر جدًا أن تحذف قيمة أحد المتغيرات يدويًا وإنما تستعمل استعمالًا رئيسيًا لحذف أحد عناصر القوائم.

4.1.8 التعامل مع القوائم

حينما تبدأ بالبرمجة، قد يغيرك إنشاء متغيرات لكل قيمة من مجموعة قيم تنتمي إلى مجموعة محددة.

مثلًا لو أردت كتابة أسماء قططي فقد أفكر بكتابة شيفرة كالآتية:

```
catName1 = 'Zophie'
catName2 = 'Pooka'
catName3 = 'Simon'
catName4 = 'Lady Macbeth'
catName5 = 'Fat-tail'
catName6 = 'Miss Cleo'
```

لكن هذه شيفرة تعيسة! فماذا يحصل لو كان عدد القطط في برنامج متغيرًا؟ فلن يستطيع برنامجك تخزين أسماء قصص لا تملك متغيرات لها. أفضل إلى ذلك أن هذه الأنواع من البرنامج فيها تكرار كثير للشيفرات نفسها، انظر إلى مقدار التكرار في المثال الآتي الذي أنصحك بكتابته باسم `AllMyCats1.py` وتجربته:

```
print('Enter the name of cat 1:')
catName1 = input()
print('Enter the name of cat 2:')
catName2 = input()
print('Enter the name of cat 3:')
catName3 = input()
print('Enter the name of cat 4:')
catName4 = input()
print('Enter the name of cat 5:')
catName5 = input()
print('Enter the name of cat 6:')
catName6 = input()
print('The cat names are:')
print(catName1 + ' ' + catName2 + ' ' + catName3 + ' ' + catName4 + '
' +
catName5 + ' ' + catName6)
```

بدلًا من استخدام أسماء متغيرات متكررة، يمكنك إنشاء متغير واحد يحتوي على قائمة بكل القسم، فهذه نسخة محسنة من المثال السابق، التي نستخدم فيها قائمةً واحدةً يمكن أن تحتوي أي عدد من أسماء القطط التي يمكن أن يدخلها المستخدم.

جرب المثال `AllMyCats2.py`:

```
catNames = []
while True:
  print('Enter the name of cat ' + str(len(catNames) + 1) +
    ' (Or enter nothing to stop.):')
  name = input()
  if name == '':
    break
  catNames = catNames + [name]  # جمع القوائم
print('The cat names are:')
for name in catNames:
  print('  ' + name)
```

ناتج تجربة المثال السابق:

```
Enter the name of cat 1 (Or enter nothing to stop.):
Zophie
Enter the name of cat 2 (Or enter nothing to stop.):
Pooka
Enter the name of cat 3 (Or enter nothing to stop.):
Simon
Enter the name of cat 4 (Or enter nothing to stop.):
Lady Macbeth
Enter the name of cat 5 (Or enter nothing to stop.):
Fat-tail
Enter the name of cat 6 (Or enter nothing to stop.):
Miss Cleo
Enter the name of cat 7 (Or enter nothing to stop.):

The cat names are:
 Zophie
 Pooka
 Simon
 Lady Macbeth
 Fat-tail
 Miss Cleo
```

الفائدة من استخدام القوائم هي أن بياناتك أصبحت مهيكلة هيكلةً أفضل، وسيكون برنامجك مرنًا في معالجة البيانات والتعامل مع مجموعة مشتركة من المتغيرات.

4.1.9 استخدام حلقات for مع القوائم

تعلمنا في الفصل الثاني من هذا الكتاب عن حلقات التكرار `for` لتنفيذ كتلة من الشيفرات لعدد معين من المرات؛ لكن تقنيًا ما تفعله `for` هو تكرار الشيفرة داخلها مرةً واحدةً لكل عنصر من عناصر القائمة. فالشيفرة:

```
for i in range(4):
  print(i)
```

ستخرج الناتج الآتي:

```
0
1
2
3
```

هذا لأن القيمة المعادة من تنفيذ `range(4)` هي قيمة متسلسلة sequence value التي تعاملها بايثون معاملةً شبيهة بالقائمة `[0, 1, 2, 3]` (سنشرح المتسلسلات Sequences لاحقًا في هذا الفصل).

سيكون ناتج البرنامج السابق مماثلًا تمامًا لما يلي:

```
for i in [0, 1, 2, 3]:
  print(i)
```

حلقة التكرار `for` السابقة تمر على جميع عناصر القائمة `[0, 1, 2, 3]` وتضبط قيمة المتغير `i` إلى قيمة كل عنصر منها.

من الشائع استخدام التعبير `range(len(someList))` مع حلقة التكرار `for` في بايثون للمرور على جميع عناصر قائمة ما:

```
>>> supplies = ['pens', 'staplers', 'flamethrowers', 'binders']
>>> for i in range(len(supplies)):
...     print('Index ' + str(i) + ' in supplies is: ' + supplies[i])

Index 0 in supplies is: pens
Index 1 in supplies is: staplers
Index 2 in supplies is: flamethrowers
Index 3 in supplies is: binders
```

استخدام `range(len(supplies))` في المثال السابق مناسب لأن الشيفرة داخل الحلقة تستطيع الوصول إلى الفهرس عبر المتغير i وإلى القيمة المرتبطة بذاك الفهرس عبر `supplies[i]`، والأفضل من ذلك كله أن `range(len(supplies))` ستؤدي إلى المرور على جميع عناصر القائمة بغض النظر عن عددها.

4.1.10 العاملان in و not in

يمكنك معرفة إن كانت قيمةٌ ما موجودةً -أو غير موجودةٍ- في قائمة ما باستخدام العاملين `in` و `not in`. وكما في بقية العوامل، يستعمل العاملان `in` و `not in` في التعابير وتربطان قيمتين: قيمة نرغب بالبحث عنها في القائمة والقائمة التي نرغب بالبحث فيها؛ وتكون نتيجة هذه التعابير قيمة منطقية بوليانية:

```
>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas']
True
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> 'cat' in spam
False
>>> 'howdy' not in spam
False
>>> 'cat' not in spam
True
```

فمثلًا يسمح البرنامج الآتي للمستخدم بكتابة اسم قطته ليتأكد إن كان اسمها ضمن قائمة من أسماء القطط. احفظه باسم `myPets.py` وجربه:

```
myPets = ['Zophie', 'Pooka', 'Fat-tail']
print('Enter a pet name:')
name = input()
if name not in myPets:
  print('I do not have a pet named ' + name)
else:
  print(name + ' is my pet.')
```

سيشبه الناتج ما يلي:

```
Enter a pet name:
Footfoot
I do not have a pet named Footfoot
```

4.1.11 خدعة للإسناد المتعدد

هنالك اختصار يسمى تقنيًا بنشر الصفوف tuple unpacking يسمح لنا بإسناد عدة قيمة لمتغيرات اعتمادًا على قائمة في سطر واحد. فبدلًا من كتابة:

```
>>> cat = ['fat', 'gray', 'loud']
>>> size = cat[0]
>>> color = cat[1]
>>> disposition = cat[2]
```

نستطيع أن نكتب:

```
>>> cat = ['fat', 'gray', 'loud']
>>> size, color, disposition = cat
```

يجب أن يكون عدد المتغيرات وطول القائمة متساوٍ تمامًا، وإلا فستعطيك بايثون الخطأ ValueError:

```
>>> cat = ['fat', 'gray', 'loud']
>>> size, color, disposition, name = cat
Traceback (most recent call last):
 File "<pyshell#84>", line 1, in <module>
  size, color, disposition, name = cat
ValueError: not enough values to unpack (expected 4, got 3)
```

4.1.12 استخدام الدالة enumerate() مع القوائم

بدلًا من استخدام `range(len(someList))` مع حلقة تكرار `for` للوصول إلى قيمة الفهرس لكل عنصر من عناصر القائمة، فيمكننا استدعاء الدالة `enumerate()` بدلًا منها.

ففي كل دورة لحلقة التكرار ستعيد الدالة `enumerate()` قيمتين: فهرس العنصر الموجود في القائمة والعنصر نفسه على شكل قائمة. فالشيفرة الآتية مماثلة في الوظيفة للمثال المذكور سابقًا في جزئية «استخدام حلقات for مع القوائم»:

```
>>> supplies = ['pens', 'staplers', 'flamethrowers', 'binders']
>>> for index, item in enumerate(supplies):
...     print('Index ' + str(index) + ' in supplies is: ' + item)

Index 0 in supplies is: pens
Index 1 in supplies is: staplers
Index 2 in supplies is: flamethrowers
Index 3 in supplies is: binders
```

الدالة `enumerate()` مفيدة إن كنت تريد الوصول إلى العنصر وفهرسه ضمن حلقة التكرار.

4.1.13 استخدام الدالتين random.choice() و random.shuffle() مع القوائم

الوحدة `random` فيها عدة دوال تقبل القوائم كمعاملات لها. الدالة `random.choice()` تعيد عنصرًا مختارًا عشوائيًا من القائمة:

```
>>> import random
>>> pets = ['Dog', 'Cat', 'Moose']
>>> random.choice(pets)
'Dog'
>>> random.choice(pets)
'Cat'
>>> random.choice(pets)
'Cat'
```

يمكن القول أن `random.choice(someList)` نسخة من:

```
someList[random.randint(0, len(someList) - 1]
```

لكنها مختصرة. الدالة `random.shuffle()` تعيد ترتيب العناصر ضمن القائمة عشوائيًا، وتعـدل القائمـة مباشرة دون إعادة قائمة جديدة:

```
>>> import random
>>> people = ['Alice', 'Bob', 'Carol', 'David']
>>> random.shuffle(people)
>>> people
['Carol', 'David', 'Alice', 'Bob']
>>> random.shuffle(people)
>>> people
['Alice', 'David', 'Bob', 'Carol']
```

4.1.14 عوامل الإسناد المحسنة

من الشائع حين إسناد قيمة ما إلى متغير أن تستعمل قيمة المتغير الابتدائية أساسًا للقيمة الجديدة.

فمثلًا بعد إسنادك القيمة `42` للمتغير `spam` وأردت زيادة قيمة المتغير بمقدار واحد:

```
>>> spam = 42
>>> spam = spam + 1
>>> spam
43
```

يمكنك بدلًا من ذلك استخدام عامل الإسناد المحسن `+=` لنفس النتيجة:

```
>>> spam = 42
>>> spam += 1
>>> spam
43
```

هنالك عوامل إسناد محسنة للعوامل `+` و `-` و `*` و `/` و `%` موضحة في الجدول الآتي:

عامل الإسناد المحسن	التعبير البرمجي المكافئ
spam += 1	spam = spam + 1
spam -= 1	spam = spam - 1
spam *= 1	spam = spam * 1
spam /= 1	spam = spam / 1
spam %= 1	spam = spam % 1

الجدول 8: عوامل الأسناد المحسنة

يمكن استخدام عامل الإسناد المحسن `+=` لدمج قائمتين، والمعامل `*=` لتكرار قائمة:

```
>>> spam = 'Hello,'
>>> spam += ' world!'
>>> spam
'Hello world!'
>>> olive = ['Zophie']
>>> olive *= 3
>>> olive
['Zophie', 'Zophie', 'Zophie']
```

4.2 التوابع Methods

يمكننا القول مجازًا أن التابع `method` يكافئ الدوال لكنها «تستدعى على» قيمة ما. فمثلًا إذا استدعيت دالة القوائم `index()` -التي سنشرحها بعد قليل- على قائمة فستكتب: `list.index('hello')`؛ أي أن التابع يأتي بعد القيمة ويفصل عنها بنقطة.

لكل نوع من أنواع البيانات مجموعة توابع خاصة به، ولنوع بيانات القوائم عدد من التوابع المفيدة للبحث والإضافة والحذف ومختلف عمليات التعديل الأخرى.

4.2.1 العثور على قيمة في قائمة عبر التابع index()

تمتلك القوائم التابع `index()` الذي تقبل معاملًا وهو القيمة الـتي سـيجري البحث عنهـا في القائمـة، وإذا كان العنصر موجودًا فسيعاد فهرس ذاك العنصر، وإذا لم يكن موجودًا فستطلق بايثون الخطأ ValueError:

```
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> spam.index('hello')
0
>>> spam.index('heyas')
3
>>> spam.index('howdy howdy howdy')
Traceback (most recent call last):
 File "<pyshell#31>", line 1, in <module>
  spam.index('howdy howdy howdy')
ValueError: 'howdy howdy howdy' is not in list
```

وعند وجود قيم مكررة في القائمة فسيعاد الفهرس لأول قيمة يُعثَـر عليهـا، لاحـظ أن التـابع `index()` قـد أعاد 1 وليس 3 في هذا المثال:

```
>>> spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']
>>> spam.index('Pooka')
1
```

4.2.2 إضافة قيم إلى القوائم عبر append() و insert()

استخدم التابعين `append()` و `insert()` لإضافة قيم جديدة إلى قائمة:

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.append('moose')
>>> spam
['cat', 'dog', 'bat', 'moose']
```

أدى استدعاء التابع `append()` إلى إضافة قيمة المعامل الممرر إليه إلى نهاية القائمة.

التابع `insert()` يضيف قيمةً جديدةً إلى أي فهرس في القائمة، ويكون الوسـيط الأول الممـرر إلى التـابع `insert()` هو فهرس القيمة الجديدة، والوسيط الثاني هو القيمة التي نريد إضافتها:

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.insert(1, 'chicken')
>>> spam
['cat', 'chicken', 'dog', 'bat']
```

لاحظ أن الشيفرة التي كتبناهـا هي `spam.append('moose')` و `spam.insert(1, 'chicken')` وليسـت `spam = spam.append('moose')` أو `spam = spam.insert(1, 'chicken')`، إذ لا يعيد التابع `append()` أو `insert()` القيمة الجديدة للقائمة `spam` . وفي الواقع تكـون نتيجـة اسـتدعائها هي `None`، فلا حاجة إلى تخزين قيمة استدعاء تلك التوابع في متغير، بل تعدل تلك التوابع القائمةَ مباشرةً.

سنتحدث بالتفصيل عن القوائم القابلة للتغيير وغير القابلة للتغيير لاحقًا في هذا الفصل.

التوابع التي ترتبط بنوع بيانات محدد -مثـل `append()` و `insert()`- هي خاصـة بـذاك النـوع، فلا يمكن استخدامها على قيم أخرى مثل السلاسل النصية أو الأعداد الصحيحة.

لاحظ ظهور رسالة الخطأ AttributeError في المثال الآتي:

```
>>> eggs = 'hello'
>>> eggs.append('world')
Traceback (most recent call last):
 File "<pyshell#19>", line 1, in <module>
  eggs.append('world')
AttributeError: 'str' object has no attribute 'append'

>>> olive = 42
>>> olive.insert(1, 'world')
Traceback (most recent call last):
 File "<pyshell#22>", line 1, in <module>
  olive.insert(1, 'world')
AttributeError: 'int' object has no attribute 'insert'
```

4.2.3 إزالة القيم من القوائم عبر التابع remove()

نمرر إلى التابع `remove()` القيمة التي نريد حذفها من القائمة التي يستدعى التابع عليها:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('bat')
```

```
>>> spam
['cat', 'rat', 'elephant']
```

إذا حاولنا حذف قيمة غير موجودة في القائمة فسيظهر الخطأ ValueError:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('chicken')
Traceback (most recent call last):
 File "<pyshell#11>", line 1, in <module>
  spam.remove('chicken')
ValueError: list.remove(x): x not in list
```

إذا تكررت القيمة التي نريد حذفها أكثر من مرة في القائمة فستزال أول نسخة من تلك القيمة:

```
>>> spam = ['cat', 'bat', 'rat', 'cat', 'hat', 'cat']
>>> spam.remove('cat')
>>> spam
['bat', 'rat', 'cat', 'hat', 'cat']
```

لاحظ أن العبارة `del` مفيدة حينما تعرف فهرس العنصر الـذي تريـد حذفـه من القائمـة، بينمـا يفيـد التـابع `remove()` إذا كنت تعرف قيمة العنصر الذي تريد حذفه.

4.2.4 ترتيب عناصر قائمة عبر التابع sort()

يمكن ترتيب القوائم التي تحتوي على أعداد أو على سلاسل نصية باستخدام التابع `sort()`:

```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
[-7, 1, 2, 3.14, 5]
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

يمكنك تمرير القيمة `True` قيمةً للوسيط المسمى `reverse` من أجل جعل التـابع`sort()` يـرتب النتـائج ترتيبًا عكسيًا:

```
>>> spam.sort(reverse=True)
>>> spam
['elephants', 'dogs', 'cats', 'badgers', 'ants']
```

هنالك ثلاثة أمور أساسية يجب عليك معرفتها حـول التـابع `sort()`: بدايـةً يـرتب التـابع `sort()` عناصـر القائمة مباشرةً دون إعادة قائمة جديدة، أي ليس هنالك فائدة من كتابة شيء يشبه `spam = spam.sort()`.

الأمر الثاني هو أنك لا تستطيع ترتيب القوائم التي تحتوي على قيم عددية ونصية في آن واحد، لأن بايثون لا تعرف كيف تقارن هذه القيم مع بعضها بعضًا. لاحظ الخطأ TypeError في المثال الآتي:

```
>>> spam = [1, 3, 2, 4, 'Alice', 'Bob']
>>> spam.sort()
Traceback (most recent call last):
 File "<pyshell#70>", line 1, in <module>
  spam.sort()
TypeError: '<' not supported between instances of 'str' and 'int'
```

وأخيرًا، يستعمل التابع `sort()` ترتيب ASCII للسلاسل النصية بدلًا من الـترتيب الهجـائي، هـذا يعـني أن الأحرف الكبيرة في الإنكليزية تأتي قبل الأحرف الصغيرة أي أن a سيكون بعد Z مباشرةً:

```
>>> spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
>>> spam.sort()
>>> spam
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
```

إذا أردت ترتيب القيم ترتيبًا هجائيًا، فمرر القيمة `str.lower` للوسيط المسمى `key` في التابع `sort()`:

```
>>> spam = ['a', 'z', 'A', 'Z']
>>> spam.sort(key=str.lower)
>>> spam
['a', 'A', 'z', 'Z']
```

هذا سيجعل التابع `sort()` يتعامل مع جميع عناصر القائمة كمـا لـو أنهـا في حالـة الأحـرف الصـغيرة دون تعديل القيم نفسها.

4.2.5 قلب ترتيب عناصر قائمة عبر التابع ()reverse

إذا احتجت إلى قلب ترتيب عناصر إحدى القوائم سريعًا فاستعمل التابع `reverse()`:

```
>>> spam = ['cat', 'dog', 'moose']
>>> spam.reverse()
>>> spam
['moose', 'dog', 'cat']
```

وكما في التابع `sort()`، لا يعيد التابع `reverse()` قائمةً بل يعدلها مباشرةً، ولهذا نكتب `spam.reverse()` وليس `spam = spam.reverse()`.

ا. استثناءات من قواعد المسافات البادئة في بايثون

في أغلبية الحالات، يكون عدد المسافات البادئة قبل كل سطر برمجي دليلًا يقول لمفسر لغة بايثون في أي كتلة ينتمي ذاك السطر. لكن هنالك بعض الاستثناءات لهذه القائمة، فمثلًا يمكن أن تمتد كتابة القائمة على أكثر من سطر في الشيفرة المصدرية، ولا تهم المسافة البادئة هنا، لأن مفسر بايثون يفهم أن تعريف القائمة لا ينتهي إلا بوجود قوس الإغلاق `]`. فيمكن أن يكون لدينا شيفرة كما يلي:

```
spam = ['apples',
  'oranges',
        'bananas',
'cats']
print(spam)
```

لكن عمليًا يستعمل أغلبية المبرمجين المسافات البادئة استعمالًا صحيحًا لتسهل عملية قراءة شيفرتهم.

يمكنك أن تقسم تعليمة برمجية واحدة على أكثر من سطر باستخدام محرف إكمال السطر `\` في نهاية السطر، يمكنك أن تقول أن محرف إكمال السطر `\` يقول «سنكمل هذه التعليمة في السطر القادم»، ولن تكون المسافة البادئة في السطر الذي يلي محرف إكمال السطر `\` مهمةً، فالشيفرة الآتية صحيحة:

```
print('Four score and seven ' + \
  'years ago...')
```

ستستفيد من هذه الأمور حينما تحتاج إلى تقسيم الأسطر الطويلة إلى أسطر أقصر لتسهيل قابلية قراءتها.

4.3 مثال عملي: إعادة كتابة برنامج الكرة السحرية باستخدام القوائم

يمكننا كتابة نسخة أفضل من مثال الكرة السحرية الذي كتبناه عبر عبارات `elif` سابقًا، إذ يمكننا إنشاء قائمة واحدة وسنجعل البرنامج يتعامل معها مباشرةً. احفظ ما يلي في ملف باسم `magic8ball2.py`:

```
import random

messages = ['It is certain',
  'It is decidedly so',
  'Yes definitely',
  'Reply hazy try again',
  'Ask again later',
  'Concentrate and ask again',
  'My reply is no',
  'Outlook not so good',
  'Very doubtful']

print(messages[random.randint(0, len(messages) - 1)])
```

حينما تجرب هذا البرنامج فسيبدو ناتجه كما في المثال `magic8ball.py` الأصلي.

لاحظ التعبير الذي استخدمناه كفهرس للقائمة:

```
messages: random.randint (0, len(messages) - 1)
```

وهو يولد عدد عشوائيًا نستعمله كفهرس بغض النظر عن عدد العناصر الموجودة في القائمة `messages`.

ذاك التعبير يولد قيمةً عشوائيًا بين `0` وقيمة `len(messages) - 1`، والقائدة من هذه الطريقة أننا نستطيع إضافة وإزالة العناصر من القائمة `messages` دون الحاجة إلى تغيير أي شيفرات أخرى، فعندما تحدث شيفرة البرنامج مستقبلًا لإضافة عناصر جديدة إلى القائمة، فلا تضطر إلى تعديلات إضافية، وكلما قللت من الأمور التي تغيرها قلَّت احتمالية استحداث علل برمجية جديدة.

4.4 أنواع البيانات المتسلسلة Sequence Data Types

القوائم هي إحدى أنواع البيانات التي تمثل قائمةً من القيم، لكنها ليست الوحيدة. فمثلًا السلاسل النصية strings والقوائم lists متشابهة جدًا إذا تخيلنا أن السلسلة النصية هي «قائمة» من المحارف.

أنواع البيانات المتسلسلة في بايثون تتضمن القوائم lists، والسلاسل النصية strings، وكائنات المجالات range objects المعادة من الدالة `range()`، والصفوف tuples.

أغلبية الأمور التي تستطيع فعلها مع القوائم يمكنك فعلها مع بقية أنواع البيانات المتسلسلة، بما في ذلك: الفهرسة، والتقطيع، واستخدامها مع حلقات `for`، واستخدام `len()`، واستخدام العوامل `in` و `not in`.

جرب المثال الآتي لترى ذلك عمليًا:

```
>>> name = 'Zophie'
>>> name[0]
'Z'
>>> name[-2]
'i'
>>> name[0:4]
'Zoph'
>>> 'Zo' in name
True
>>> 'z' in name
False
>>> 'p' not in name
False
>>> for i in name:
...     print('* * * ' + i + ' * * *')

* * * Z * * *
* * * o * * *
* * * p * * *
* * * h * * *
* * * i * * *
* * * e * * *
```

4.5 أنواع البيانات القابلة وغير القابلة للتعديل

تختلف القوائم والسلاسل النصية عن بعضها اختلافًا جوهريًا، فالقوائم هي من أنواع البيانات القابلة للتعديل mutable data type، فيمكن أن تضاف أو تحذف أو تعدل عناصرها؛ بينما السلاسل النصية غير قابلة للتعديل immutable، فإذا حاولت ضبط قيمة أحد محارف السلسلة النصية يدويًا فسيحدث الخطأ TypeError كما في المثال الآتي:

```
>>> name = 'Zophie a cat'
>>> name[7] = 'the'
Traceback (most recent call last):
 File "<pyshell#50>", line 1, in <module>
  name[7] = 'the'
TypeError: 'str' object does not support item assignment
```

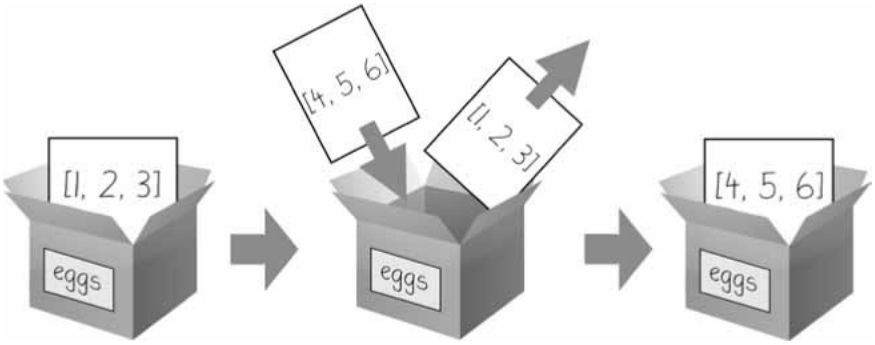
الطريقة المعتمدة «لتعديل» سلسلة نصية هي استخدام التقطيع والجمع لبناء سلسلة نصية جديدة اعتمادًا على أجزاء من السلسلة النصية القديمة:

```
>>> name = 'Zophie a cat'
>>> newName = name[0:7] + 'the' + name[8:12]
>>> name
'Zophie a cat'
>>> newName
'Zophie the cat'
```

استعملنا `[0:7]` و `[8:12]` للإشارة إلى المحارف التي لا نريد تعديلها، لاحظ أن السلسلة النصية الأصلية `'Zophie a cat'` لم تعدل، بل أنشأنا سلسلة نصية جديدة.

وصحيحٌ أن القوائم قابلة للتعديل، لكن السطر الثاني في المثال الآتي لن يعدل القائمة `eggs`:

```
>>> eggs = [1, 2, 3]
>>> eggs = [4, 5, 6]
>>> eggs
[4, 5, 6]
```



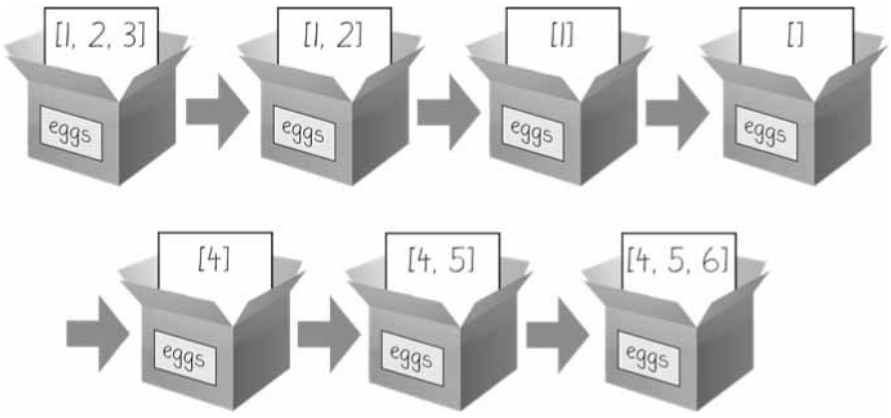
الشكل 23: ما يحدث عند إسناد قائمة جديدة إلى متغير

لم تبدل القيم في القائمة `eggs` هنا؛ بل أُنشِئت قائمة جديدة `[4, 5, 6]` وكتبت فوق القائمة القديمة `[1, 2, 3]` كما في الشكل التالي.

إذا أردت فعليًا تعديل القائمة الأصلية المخزنة في `eggs` لتحتوي على `[4, 5, 6]`، فعليك فعل شيء يشبه ما يلي:

```
>>> eggs = [1, 2, 3]
>>> del eggs[2]
>>> del eggs[1]
>>> del eggs[0]
>>> eggs.append(4)
>>> eggs.append(5)
>>> eggs.append(6)
>>> eggs
[4, 5, 6]
```

يوضح الشكل التالي التعديلات السبع التي جرت على القائمة `eggs` لتصل إلى النتيجة النهائية.



الشكل 24: تُغيِّر العبارة del والتابع ()append قيمة القائمة مباشرة

تغيير قيمة نوع بيانات قابل للتعديل (مثل استخدام العبارة `del` والتابع `append()` كما في المثال السابق) سيؤدي إلى تغيير القيمة في مكانها، وذلك لأن قيمة المتغير لا تبدل إلى قائمة جديدة.

قد يبدو لك الآن أن الفرق بين أنواع البيانات القابلة وغير القابلة للتعديل تافه أو بسيط ولا يهم، لكن قسم «تمرير المرجعيات» سيشرح لك الفرق في السلوك حين استدعاء الدوال مع وسائط قابلة للتعديل ووسائط غير قابلة للتعديل. لكن قبل ذلك دعنا نتعلم عن نوع بيانات جديد وهو الصفوف tuples، وهو يشبه القوائم لكنه غير قابل للتعديل.

4.6 الصفوف Tuples

يكاد يماثل نوع البيانات tuple القوائم تمامًا، مع استثناء أمرين اثنين: الأول أننا نعرف الصفوف عبر قوسين هلاليين `()` بدلًا من القوسين المربعين `[]`:

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[0]
'hello'
>>> eggs[1:3]
(42, 0.5)
>>> len(eggs)
3
```

والثاني -وهو المهم- أن الصفوف هي نوع بيانات غير قابل للتعديل كما في السلاسل النصية، أي لا يمكننا تعديل قيم عناصر الصف أو إضافتها أو حذفها. لاحظ رسالة الخطأ TypeError حين تنفيذ المثال الآتي:

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[1] = 99
Traceback (most recent call last):
 File "<pyshell#5>", line 1, in <module>
  eggs[1] = 99
TypeError: 'tuple' object does not support item assignment
```

إذا كانت لديك قيمة واحدة في الصف، فيمكنك أن تطلب من بايثون تعريف ذاك الصف بوضع فاصلة بعد تلك القيمة، وإلا فستظن بايثون أنك كتبت قيمةً عاديةً ضمن قوسين، فالفاصلة هنا هي ما سيخبر بايثون أن ما تريده هو نوع البيانات tuple. لاحظ أن من الطبيعي في بايثون وجود فاصلة بعد آخر عنصر في صف أو قائمة على عكس بعض لغات البرمجة الأخرى.

جرب الدالة `type()` في المثال الآتي لترى الفرق الذي يحدثه استخدام الفاصلة بعد القيمة عمليًا:

```
>>> type(('hello',))
<class 'tuple'>
>>> type(('hello'))
<class 'str'>
```

يمكنك استخدام الصفوف في برنامجك لتقول لمن يقرأه من المبرمجين أنك لا تنوي لهذه السلسلة من القيم أن تتغير؛ أي لو أردت قيمًا متسلسلة مرتبة لا تتغير فاستخدم الصفوف.

ميزة أخرى من مزايا الصفوف بدلًا من القوائم هي أن بايثون تستطيع تطبيق بعض التحسينات الداخلية لمعالجة الصفوف معالجةً أسرع من القوائم، وذلك لعلمها أنها قيم متسلسلة غير قابلة للتعديل.

4.6.1 تبديل أنواع التسلسلات باستخدام الدوال list() و tuple()

كما تعيد الدالة `str(42)` القيمة `'42'` وهو التمثيل النصي للرقم 42؛ تعيد الدالتان `list()` و `tuple()` نسخة القائمة والصف من القيم الممررة إليها. جرب المثال الآتي ولاحظ كيف أن نوع القيمة المعادة مختلف عن نوع القيمة الممررة:

```
>>> tuple(['cat', 'dog', 5])
('cat', 'dog', 5)
>>> list(('cat', 'dog', 5))
['cat', 'dog', 5]
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
```

قد يفيدك تحويل صف إلى قائمة إن احتجت إلى نسخة قابلة للتعديل من قيمة ذاك الصف.

4.7 المرجعيات References

كما رأيت سابقًا، «تُخزِّن» المتغيرات قيم السلاسل النصية والأعداد، لكن هذا تبسيط لما تقوم به بايثون فعلًا. فتقنيًا تخزن المتغيرات مرجعيةً أو إشارةً إلى مكان تخزين قيمة المتغير في ذاكرة الحاسوب:

```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
>>> cheese
42
```

فعندما تسند القيمة 42 إلى المتغير spam فأنت تنشِئ القيمة 42 في ذاكـرة الحاسـوب ثم تخـزِّن مرجعيـةً reference إليها في المتغير spam، وعندما تنسخ القيمة في spam وتسندها إلى المتغير cheese فأنت فعليًا تنسخ المرجعية إلى القيمة 42 في ذاكرة الحاسوب وليس القيمة نفسها، أي أن كلا المتغـيرين spam و cheese يشيران إلى القيمة 42 نفسها في ذاكرة الحاسوب.

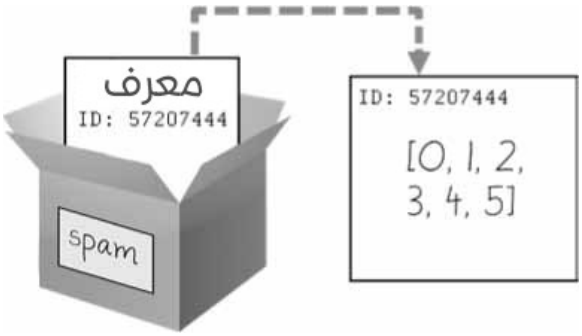
ثم حينما تغيّر قيمة المتغير spam إلى 100 فأنت تنشِئ قيمةً جديدةً وهي 100 ثم تخزن مرجعيـةً إليهـا في المتغير spam، وهذا لا يؤثر على القيمة الموجودة في المتغير cheese.

تذكر أن القيم العددية من أنواع البيانات غير القابلة للتعـديل، أي أن تغيـير قيمـة spam سـيؤدي إلى تغيـير المرجعية التي تشير إليها في الذاكرة. لكن لا تعمل القوائم بهذه الطريقـة، وذلـك لأن القـوائم من أنـواع البيانـات القابلة للتعديل. هذا المثال يسهِّل فهم الآلية السابقة والفروق بين القوائم وغيرها من أنواع البيانات:

```
❶ >>> spam = [0, 1, 2, 3, 4, 5]
❷ >>> cheese = spam # ستنسخ المرجعية وليست القائمة
❸ >>> cheese[1] = 'Hello!' # وهذا ما يغير قيمة عنصر القائمة
 >>> spam
 [0, 'Hello!', 2, 3, 4, 5]
 >>> cheese # يشير المتغير إلى القائمة نفسها
 [0, 'Hello!', 2, 3, 4, 5]
```

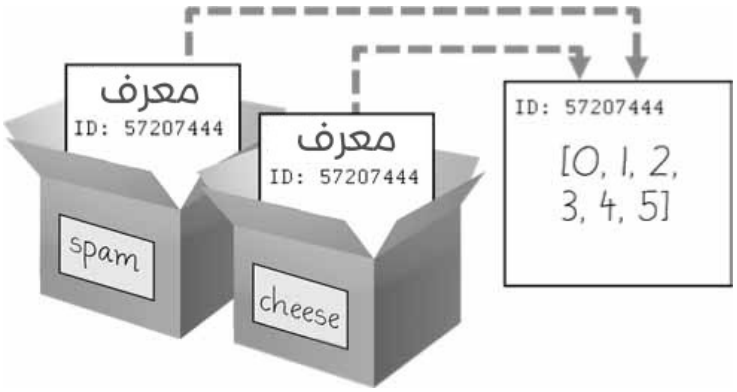
قد يبدو الناتج السابق غريبًا بالنسبة إليك، فأنت تعـدل فيـه على القائمـة cheese لكن التغيـيرات حـدثت على المتغير cheese و spam معًا! عند إنشائك للقائمة ❶ فأنت تخـزن مرجعيـةً إليهـا في المتغـير spam، وفي السطر التالي ❷ نسخت المرجعية الموجودة في spam إلى cheese وليس القائمة نفسها. وهذا يعـني أن القيم المخزنة في المتغيرين spam و cheese تشير إلى القائمة نفسها. لاحظ أن هنالك قائمـة واحـدة لأن القائمـة لم تنسَخ بحد ذاتها بل نُسِخَت المرجعية إليها؛ لذا حينما تعدل أحد عناصر القائمة cheese ❸ فأنت تعدل نفس القائمة التي يُشار إليها عبر المتغير spam.

تذكر أن المتغيرات تشبه الصـناديق الـتي تحتـوي على قيم، والرسـومات التوضـيحية الـتي رأيتهـا في هـذا الفصل لحد الآن ليست دقيقة تمامًا لأن من غير الممكن احتواء قائمة داخل صندوق، بل تحتوي الصـناديق على إشارات لتلك القوائم، وتلك الإشارات تملك معرفات ID تستعملها بايثون داخليًا، ولتصـحيح تخيلنـا للمتغـيرات والقوائم فانظر إلى الشكل التالي:



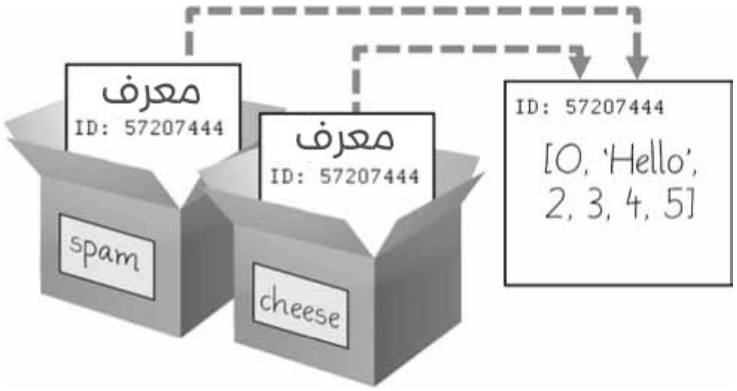
الشكل 25: تخزن مرجعية إلى قائمة في المتغيرات،
وليست القائمة نفسها

في الشكل القادم سننسخ المرجعية الموجـودة في `spam` إلى `cheese`، لاحـظ تخـزين قيمـة المرجعيـة في `cheese` وليس القائمة. لاحظ كيف يشير كلا المتغيرين إلى القائمة نفسها:



الشكل 26: إسناد قيمة متغير إلى آخر ينسخ المرجعية وليس القائمة

وحينما تعدل القائمة التي يشير إليها المتغير `cheese` فأنت تعدل القائمـة الـتي يشـير إليهـا `spam` أيضًـا، لأنهما يشيران إلى القائمة نفسها، يمكنك ملاحظة ذلك في الشكل التالي:



الشكل 27: تغيير عنصر في قائمة يشار إليها من متغيرين مختلفين

صحيحٌ أن بـايثون تخـزن مرجعيـات في المتغـيرات، لكن من الشـائع أن يقـول المطـورون أن «المتغـيرات تحتوي على قيم» وليس «المتغيرات تحتوي على مرجعيات تشير إلى قيم».

4.7.1 المعرفات والدالة id()

قد تتساءل لماذا يطبـق السـلوك السـابق الغـريب الـذي ناقشـناه في القسـم السـابق على القـوائم القابلـة للتعديل ولا يحدث على القيم غير القابلة للتعديل كالأعداد أو السلاسل النصية.

يمكننا استخدام الدالة `id()` لفهم ذلك، فكل القيم في بايثون لها معـرف خـاص بهـا يمكن الحصـول عليـه باستخدام الدالة `id()`:

```
>>> id('Howdy') # ستختلف القيمة المعادة في حاسوبك
44491136
```

عندما تشغل بايثون العبارة البرمجيـة `id('Howdy')` فهي تنشِـئ السلسـلة النصـية `'Howdy'` في ذاكـرة حاسوبك، ويعاد عنوان الذاكرة الرقمي الذي خُزِّنَت السلسلة النصية فيه عبر الدالة `id()`، وتختار بـايثون العنـوان اعتمادًا على أي بايتات تتوافر في ذاكرة حاسوبك في وقت التنفيذ،و لهذا ستختلف القيمـة في كـل مـرة تشـغل فيها الشيفرة.

وككل السلاسل النصية، السلسـلة `'Howdy'` غـير قابلـة للتعـديل، وإذا حـاولت «تعـديل» قيمـة السلسـلة النصية الموجودة في متغير، فستُنشَأ سلسلة نصـية جديـدة في مكـان آخـر في الـذاكرة ثم سيشـير المتغـير إلى السلسلة النصية الجديدة. جرب المثال الآتي ولاحظ تغيير المعرف الذي يشير إليه المتغير `olive`:

```
>>> olive = 'Hello'
>>> id(olive)
44491136
>>> olive += ' world!' # سلسلة نصية جديدة
>>> id(olive) # يشير المتغير إلى سلسلة نصية مختلفة
44609712
```

لكن يمكن تعديل القوائم لأنها من أنواع البيانات القابلة للتعديل. فالتابع `append()` لا ينشِئ قائمة جديـدة حين تنفيذه، بل يعدل القائمة الموجودة، ونسمي هذا السلوك «بالتعديل في المكان in-place»:

```
>>> eggs = ['cat', 'dog'] # إنشاء قائمة جديدة
>>> id(eggs)
35152584
```

```
>>> eggs.append('moose') # يضيف التابع القيم مباشرة
>>> id(eggs) # يشير المتغير إلى نفس القائمة السابقة
35152584
>>> eggs = ['bat', 'rat', 'cow'] # إنشاء قائمة جديدة لها معرف مختلف
>>> id(eggs) # يشير المتغير إلى قائمة مختلفة كليًا
44409800
```

إذا أشار متغيران أو أكثر إلى القائمة نفسها (كما في المثال في القسم السابق) ثم تغيرت قيمة القائمة، فستحدث التغييرات على كلا المتغيرات لأنهما يشيران إلى القائمة نفسها.

التوابع `append()` و `extend()` و `remove()` و `sort()` و `reverse()` وغيرها من توابع القوائم ستعمل القوائم في مكانها. جامع القمامة التلقائي في Python (أي Garbage Collector) يحذف أي قيم لا يشار إليها من المتغيرات لكي يُفرِّغ الذاكرة، وهذا رائع لأن الإدارة اليدوية للذاكرة في لغات البرمجة الأخرى هي سبب رئيسي للعلل البرمجية.

4.7.2 تمرير المرجعيات

من المهم فهم المرجعيات لاستيعاب كيف تمرر الوسائط إلى الدوال.

حين استدعاء دالة ما، فإن القيم الممررة كوسائط arguments تنسخ إلى المعاملات parameters، وبالنسبة إلى القوائم (والقواميس التي سنتعرف عليها في الفصل القادم) هذا يعني أن المرجعية التي تشير إلى القائمة ستنسخ من الوسيط إلى المعامل، ولكي تعي آثار ذلك جرب المثال الآتي باسم `passingReference.py`:

```
def eggs(someParameter):
  someParameter.append('Hello')


spam = [1, 2, 3]
eggs(spam)
print(spam)
```

لاحظ أنه حين استدعاء `eggs()` فلن تستعمل القيمة المعادة من الدالة لإسناد قيمة جديدة إلى المتغير `spam`، بل ستعدل المتغير `spam` في مكانه مباشرةً؛ وسيخرج الناتج الآتي:

```
[1, 2, 3, 'Hello']
```

وصحيحٌ أن قيمة spam نسخت إلى someParameter لكن ما نسخ فعليًا هو المرجعية إلى نفس القائمة، ولهذا سيؤدي استدعاء التابع append('Hello') إلى تعديل القائمة خارج الدالة.

أبقِ هذا السلوك في ذهنك أثناء كتابة الشيفرات، فلو نسيت كيف تتعامل بايثون مع القوائم والقواميس فستقع في أخطاء وعلل كان من السهل تفاديها.

4.7.3 الدالة copy() و deepcopy() في الوحدة copy

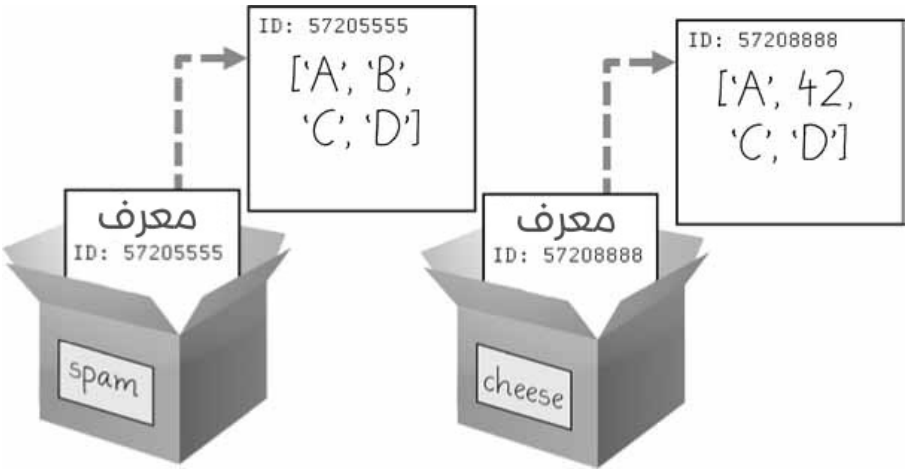
صحيحٌ أن تمرير المرجعيات للإشارة إلى القوائم والقواميس يسهل التعامل معها، لكن إن كانت لدينا دالة تغير القائمة أو القاموس الممرر إليها وكنّا لا نريد إجراء تلك التعديلات على القائمة أو القاموس الأصليين، فحينها يمكننا الاستفادة من الوحدة التي توفرها بايثون باسم copy التي توفر الدالتين copy() و deepcopy().

أول دالة منهما copy.copy() تنشِئ نسخةً طبق الأصل من قيمة قابلة للتعديل كالقوائم أو القواميس:

```
>>> import copy
>>> spam = ['A', 'B', 'C', 'D']
>>> id(spam)
44684232
>>> cheese = copy.copy(spam)
>>> id(cheese) # قائمة مختلفة بمعرف مختلف
44685832
>>> cheese[1] = 42
>>> spam
['A', 'B', 'C', 'D']
>>> cheese
['A', 42, 'C', 'D']
```

يشير المتغيران spam و cheese إلى قوائم مختلفة، ولهذا السبب سنجد أن القائمة المشار إليها عبر المتغير cheese هي من تغيرت حينما ضبطنا العنصر ذا الفهرس 1 إلى القيمة 42.

لاحظ في الآتي أن أرقام المعرفات ID مختلفة لكلا المتغيرين، لأن كل واحد منهما يشير لقائمة مختلفة.



الشكل 28: نسخ القائمة عبر ()copy ينشِئ قائمة قابلة للتعديل المستقل عن القائمة الأصلية

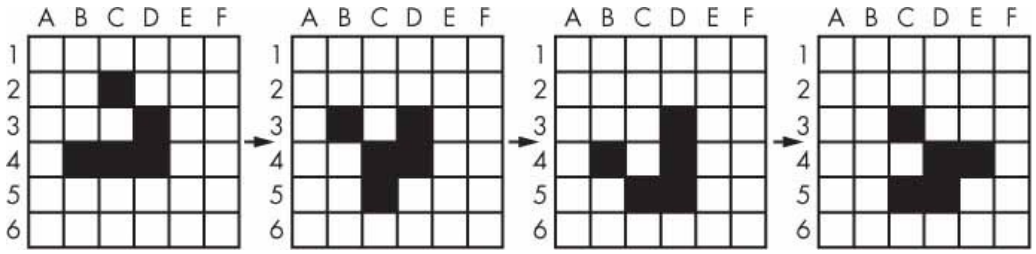
إذا كـانت لـديك قائمـة تـرغب بنسـخ محتوياتهـا أيضًـا فاسـتعمال الدالـة `copy.deepcopy()` بـدلًا من `copy.copy()`. ستنسخ الدالة `deepcopy()` القوائم الداخلية أيضًا.

4.8 برنامج قصير: لعبة الحياة

لعبة الحياة لكونواي Conway's Game of Life هي مثال عن خلايا ذاتية السلوك cellular automata: مجموعة من القوائم التي تحكم سلوك حقل مؤلف من خلايا منفصلة. عمليًا هذه طريقة لإنشـاء أشـكال متحركـة جميلة، يمكنك أن ترسل كل خطوة على ورقة رسم بياني، وتمثل المربعات في ورقة الرسم الخلايا.

- المربع الممتلئ هو خلية «حية»، بينما المربع الفارغ هو خلية «ميتة».
- تموت أي خلية حية لها أقل من اثنتين من الجيران الأحياء.
- أي خلية حية لها اثنتين أو ثلاثة جيران من الخلايا الحية تعيش إلى الجيل القادم.
- تموت أي خلية حية لها أكثر من ثلاثة جيران من الخلايا الحية.
- أي خلية ميتة تصبح حية عندما يصبح حولها بالضبط ثلاثة من الخلايا الأحياء.
- تموت أي خلية أخرى أو تبقى ميتة في الجيل القادم.

يمكنك النظر إلى تمثيل لتقدم أجيل لعبة الحياة في الشكل التالي:



الشكل 29: أربع خطوات أو أجيال في لعبة الحياة

صحيح أن القواعد بسيطة نسبيًا، لكن قد تحدث بعض السلوكيات المثيرة، فيمكن أن تتحرك الأنماط في لعبة الحياة أو أن تتكاثر، أو حتى تحاكي عمل المعالجات المركزية CPUs. لكن في أساس كل هذه الأنماط المعقدة برنامج بسيط.

يمكننا استخدام قائمة تحتوي على قوائم داخلها لتمثيل الحقل ثنائي الأبعاد، وتمثل القوائم الداخلية عمودًا من المربعات، وتخزن القيمة `'#'` للخلايا الحية، والقيمة `' '` للخلايا الميتة.

اكتب المثال الآتي في ملف باسم `conway.py`، ولا مشكلة إن لم تفهم كل ما هو مذكور فيه، كل ما عليك هو إدخاله ومحاولة فهم التعليقات والشروحات:

```
# لعبة الحياة

import random, time, copy
WIDTH = 60
HEIGHT = 20


# إنشاء قوائم الخلايا

nextCells = []
for x in range(WIDTH):
  column = [] # إنشاء عمود جديد

  for y in range(HEIGHT):

    if random.randint(0, 1) == 0:

      column.append('#') # إضافة خلية حية

    else:

      column.append(' ') # إضافة خلية ميتة

  nextCells.append(column) # nextCells هي قائمة تتألف من قوائم للأعمدة
```

```
while True: # حلقة البرنامج الرئيسية
  print('\n\n\n\n\n') # فصل الخطوة أو الجيل القادم بأسطر فارغة
  currentCells = copy.deepcopy(nextCells)

  # طباعة الخلايا الحالية على الشاشة
  for y in range(HEIGHT):
    for x in range(WIDTH):
      print(currentCells[x][y], end='') # طباعة # أو فراغ
    print() # طباعة سطر جديد في نهاية السطر

  # حساب الخطوة أو الجيل القادم اعتمادًا على القيم الحالية للخلايا
  for x in range(WIDTH):
    for y in range(HEIGHT):
      # الوصول إلى إحداثيات الخلايا المجاورة
      # `% WIDTH` يضمن أن leftCoord سيكون بين 0 و WIDTH - 1
      leftCoord  = (x - 1) % WIDTH
      rightCoord = (x + 1) % WIDTH
      aboveCoord = (y - 1) % HEIGHT
      belowCoord = (y + 1) % HEIGHT

      # إحصاء عدد الخلايا المجاورة
      numNeighbors = 0
      if currentCells[leftCoord][aboveCoord] == '#':
        numNeighbors += 1 # الخلية في الركن العلوي الأيسر حية
      if currentCells[x][aboveCoord] == '#':
        numNeighbors += 1 # الخلية في الأعلى حية
```

```
      if currentCells[rightCoord][aboveCoord] == '#':
        numNeighbors += 1 # الخلية في الركن العلوي الأيمن حية
      if currentCells[leftCoord][y] == '#':
        numNeighbors += 1 # الخلية على اليسار حية
      if currentCells[rightCoord][y] == '#':
        numNeighbors += 1 #الخلية على اليمين حية
      if currentCells[leftCoord][belowCoord] == '#':
        numNeighbors += 1 # الخلية في الركن السفلي الأيسر حية
      if currentCells[x][belowCoord] == '#':
        numNeighbors += 1 # الخلية في الأسفل حية
      if currentCells[rightCoord][belowCoord] == '#':
        numNeighbors += 1 #  الخلية في الركن السفلي الأيمن حية

      # ضبط قيمة القيمة اعتمادًا على قواعد لعبة الحياة
      if currentCells[x][y] == '#' and (numNeighbors == 2 or
numNeighbors == 3):
        # الخلايا التي لها خلايا جارة حية عددها 2 أو 3
        nextCells[x][y] = '#'
      elif currentCells[x][y] == ' ' and numNeighbors == 3:
        # الخلايا الميتة التي لها 3 خلايا جارة حية
        nextCells[x][y] = '#'
      else:
        # كل ما بقي يكون ميتًا أو سيبقى ميتًا
        nextCells[x][y] = ' '
    time.sleep(1) # التوقف لمدة ثانية لتجنب تأثير الوموض المزعج
```

لنلقي نظرةً على الشيفرة سطرًا بسطر بدءًا من الأعلى:

```
# لعبة الحياة

import random, time, copy
WIDTH = 60
HEIGHT = 20
```

استوردنا بدايةً الوحدات التي تحتوي على الدوال التي سنحتاج إليها، تحديدًا الدوال `random.randint()` و`time.sleep()` و`copy.deepcopy()`.

```
# إنشاء قوائم الخلايا

nextCells = []
for x in range(WIDTH):
  column = [] # إنشاء عمود جديد

  for y in range(HEIGHT):

    if random.randint(0, 1) == 0:

      column.append('#') # إضافة خلية حية

    else:

      column.append(' ') # إضافة خلية ميتة

  nextCells.append(column) # nextCells هي قائمة تتألف من قوائم للأعمدة
```

أول خطوة من إنشاء خلايا ذاتية السلوك هي خطوة (أو جيل) عشوائية تمامًا. سنحتاج إلى إنشاء قائمة تضم قوائم لتخزين السلاسل النصية `'#'` و `' '` التي تمثل الخلايا الحية والميتة، وسيدل مكانها في قائمة القوائم على مكانها في الشاشة، فكل قائمة داخلية تمثل عمودًا من الخلايا، واستدعاؤنا للدالة `random.randint(0, 1)` سيعطي الخلية احتمال 50% أن تكون حية و 50% أن تكون ميتة.

سنضع قائمة القوائم في متغير اسمه `nextCells`، لأن أول خطوة ستجريها في حلقة البرنامج الرئيسية هي نسخ `nextCells` إلى `currentCells`.

ستبدأ إحداثيات محور السينات `X` من 0 في الأعلى وستزداد نحو اليمين، بينما ستبدأ إحداثيات محور العينات `Y` من 0 أيضًا في الأعلى وستزداد نحو الأسفل، أي أن `nextCells[0][0]` ستمثل الخلية في الركن العلوي الأيسر من الشاشة، بينما `nextCells[1][0]` ستمثل الخلية التي على يمينها، و `nextCells[0][1]` ستمثل الخلية التي تدنوها.

```
while True: # حلقة البرنامج الرئيسية
  print('\n\n\n\n\n') # فصل الخطوة أو الجيل القادم بأسطر فارغة
  currentCells = copy.deepcopy(nextCells)
```

كل دورة من حلقة تكرار البرنامج الرئيسية ستمثل خطوة أو جيلًا من الخلايا ذاتية السلوك التي لدينا. وفي كل دورة سننسخ قيمة `nextCells` إلى `currentCells` ثم نطبع `currentCells` على الشاشة، ثم سنستخدم الخلايا الموجودة في `currentCells` لحساب الخلايا التي ستكون في `nextCells`.

```
  # طباعة الخلايا الحالية على الشاشة
  for y in range(HEIGHT):
    for x in range(WIDTH):
      print(currentCells[x][y], end='') # طباعة # أو فراغ
    print() # طباعة سطر جديد في نهاية السطر
```

حلقات for المتشعبة تعني أننا سنطبع سطرًا كاملًا من الخلايا على الشاشة، ثم يكون متبوعًا بسطر فارغ، ثم نكرر العملية لكل سطر في `nextCells`.

```
  # حساب الخطوة أو الجيل القادم اعتمادًا على القيم الحالية للخلايا
  for x in range(WIDTH):
    for y in range(HEIGHT):
      # الوصول إلى إحداثيات الخلايا المجاورة
      # `% WIDTH` يضمن أن leftCoord سيكون بين 0 و WIDTH - 1
      leftCoord  = (x - 1) % WIDTH
      rightCoord = (x + 1) % WIDTH
      aboveCoord = (y - 1) % HEIGHT
      belowCoord = (y + 1) % HEIGHT
```

ثم سنعمل حلقتي `for` متشعبتين لحساب كل خلية للخطوة أو الجيل القادم، ولمّا كانت حالة الخلية إن كانت ستحيى أم ستموت في الجيل القادم معتمدةً على جاراتها من الخلايا، فعلينا أولًا حساب فهرس الخلايا التي على يسارها ويمينها وأعلاها وأدناها.

عامل باقي القسمة % يجري عمليـة «التفـاف للسـطر»، فالجـار الأيسـر لخليـة موجـودة في العمـود الأيسـر سيكون 1 - 0 أو 1-، والالتفاف العمود إلى فهرس العمود الأيمن 59 فسنحسب (0 - 1) % WIDTH، ولأن قيمة WIDTH هي 60 فستكون نتيجة التعبير هي 59. يمكن فعـل المثـل بالنسـبة إلى الخلايـا الجـارة الـتي تعلـو وتدنو وعلى يمين الخلية الحالية.

```
        # إحصاء عدد الخلايا المجاورة
        numNeighbors = 0
        if currentCells[leftCoord][aboveCoord] == '#':
          numNeighbors += 1 # الخلية في الركن العلوي الأيسر حية
        if currentCells[x][aboveCoord] == '#':
          numNeighbors += 1 # الخلية في الأعلى حية
        if currentCells[rightCoord][aboveCoord] == '#':
          numNeighbors += 1 # الخلية في الركن العلوي الأيمن حية
        if currentCells[leftCoord][y] == '#':
          numNeighbors += 1 # الخلية على اليسار حية
        if currentCells[rightCoord][y] == '#':
          numNeighbors += 1 #الخلية على اليمين حية
        if currentCells[leftCoord][belowCoord] == '#':
          numNeighbors += 1 # الخلية في الركن السفلي الأيسر حية
        if currentCells[x][belowCoord] == '#':
          numNeighbors += 1 # الخلية في الأسفل حية
        if currentCells[rightCoord][belowCoord] == '#':
          numNeighbors += 1 #  الخلية في الركن السفلي الأيمن حية
```

لتقرير إن كانت الخلية الموجودة في nextCells[x][y] ستكون حيةً أو ميتـةً فنحتـاج إلى إحصـاء عـدد الخلايا الحية المجاورة للخلية currentCells[x][y]، والسلسلة السابقة من تعابير if الشرطية تتحقـق من الجارات الثمانية المجاورة للخلية، وتضيف القيمة 1 للمتغير numNeighbors لكل خلية حية.

```
        # ضبط قيمة القيمة اعتمادًا على قواعد لعبة الحياة
        if currentCells[x][y] == '#' and (numNeighbors == 2 or
    numNeighbors == 3):
            # الخلايا التي لها خلايا جارة حية عددها 2 أو 3
            nextCells[x][y] = '#'
        elif currentCells[x][y] == ' ' and numNeighbors == 3:
            # الخلايا الميتة التي لها 3 خلايا جارة حية
            nextCells[x][y] = '#'
        else:
            # كل ما بقي يكون ميتًا أو سيمين
            nextCells[x][y] = ' '
    time.sleep(1) # التوقف لمدة ثانية لتجنب تأثير الوموض المزعج
```

ثم بمعرفتنا لعدد الخلايا الجارة الحية للخلية `currentCells[x][y]`، فيمكننا ضبط قيمة `nextCells[x][y]` إلى `'#'` أو `' '`. وبعد المرور على جميع إحداثيات `x` و `y` فسيتوقف التنفيذ لبرهة باستدعاء `time.sleep(1)` ثم سيكمل تنفيذ البرنامج في بداية حلقة التكرار مجددًا.

هنالك عدد من الأنماط للخلايا لها أسماء مثل «الطائرة الشراعية» أو «الطائرة ذات المروحة» أو «سفينة الفضاء الثقيلة». نمط «الطائرة الشراعية» هو النمط الذي رأيته في الشكل 28 وهو «يتحرك» قطريًا كل أربع خطوات. يمكنك إنشاء «طائرة شراعية» بتبديل السطر الآتي في برنامج `conway.py`:

```
    if random.randint(0, 1) == 0:
```

إلى هذا السطر:

```
    if (x, y) in ((1, 0), (2, 1), (0, 2), (1, 2), (2, 2)):
```

4.9 أسئلة للتدريب

1. ما هي الأقواس المربعة `[]`؟
2. كيف ستسند القيمة `'hello'` كثالث قيمة في قائمة مخزنة في متغير باسم `spam` (على فرض أن قيمة `spam` هي `[2, 4, 6, 8, 10]`).
3. لنفترض أن قيمة `spam` هي `['a', 'b', 'c', 'd']` للأسئلة الثلاث القادمة:
   - ما هي نتيجة `spam[int(int('3' * 2) // 11)]`؟
   - ما هي نتيجة `spam[-1]`؟
   - ما هي نتيجة `spam[:2]`؟
4. لنفترض أن قيمة `olive` هي `[3.14, 'cat', 11, 'cat', True]` للأسئلة الثلاث القادمة:
   - ما هي نتيجة `olive.index('cat')`؟
   - كيف ستكون القائمة في `olive` بعد تنفيذ `olive.append(99)`؟
   - كيف ستكون القائمة في `olive` بعد تنفيذ `olive.remove('cat')`؟
5. ما هي العوامل المستخدمة لجمع القوائم وتكرارها؟
6. ما الفرق بين التابعين `append()` و `insert()`؟
7. ما هي الطريقتان التي يمكن بهما حذف عناصر قائمة؟
8. اذكر عددًا من أوجه الشبه بين القوائم والسلاسل النصية.
9. ما الفرق بين القوائم والصفوف؟
10. كيف تعرف صفًا فيه القيمة العددية `42` فقط؟
11. كيف يمكنك تحويل قائمة إلى صف؟ وكيف تحول صفًا إلى قائمة؟
12. المتغيرات التي «تحتوي» على قوائم لا تخزنها داخلها مباشرةً، فماذا تخزن؟
13. ما الفرق بين `copy.copy()` و `copy.deepcopy()`؟

4.10 مشاريع للتدريب

لكي تتدرب، اكتب برامج لتنفيذ المهام الآتية.

4.10.1 افصل بفاصلة

لنقل لدينا القائمة الآتية:

```
spam = ['apples', 'bananas', 'tofu', 'cats']
```

اكتب دالة تأخذ قائمةً كمعامل وتعيد سلسلةً نصيةً فيها كل عناصر تلك القائمة مفصولٌ بينها بفاصلة ثم فراغ، وأضف الكلمة and قبل آخر عنصر. فلو كانت لدينا القائمة السابقة فستعيد الدالة القيمة 'apples, bananas, tofu, and cats'؛ لكن يجب أن تعمل دالتك مع جميع القيم. تذكر أن تجرب الدالة على قائمة فارغة [].

4.10.2 سلسلة رمي القطع النقدية

سنجري تجربة بسيطة. إذا رمينا قطعة نقدية مئة مرة، وكتبنا الحرف H لكل رأس والحرف T لكل نقش، فسيكون لدينا قائمة تشبه T T T T H H H H T T.

إذا طلبنا من كائن بشري (أهلًا بك أخي البشري :-)) أن يكتب عشوائيًا نتائج مئة رمية لقطعة نقد، فستحصل على شيء يشبه H T H T H H T H T T والذي يبدو عشوائيًا إذا نظر كائن بشري إليه، لكنه لا يمثل سلسلة عشوائية رياضيًا. فلن يكتب الكائن البشري سلسلة من ستة رؤوس أو ستة نقوش متتالية، مع أن ذلك ممكن رياضيًا لو كانت عملية رمي القطع النقدية عشوائيًا فعليًا. فمن المتوقع أن تكون التوقعات العشوائية للكائنات البشرية تعيسةً (آسف أخي البشري، لكنها الحقيقة :-( ).

اكتب برنامجًا يعرف كم مرة ظهرت سلسلة من ستة رؤوس أو ستة نقوش في قائمة مولدة عشوائيًا.

سيقسم برنامجك هذه التجربة إلى قسمين: القسم الأول سيولد قائمة عشوائية من الرؤوس والنقوش، والقسم الثاني سيتحقق من سلسلةً متتاليةً من الروؤس أو النقوش موجودة في تلك القائمة.

أجرِ هذه التجربة 10,000 مرة حتى تعرف النسبة التي تحتوي فيها قائمة الرؤوس والنقوش على ستة رؤوس متتالية أو ستة نقوش متتالية. تذكر أن الدالة random.randint(0, 1) ستعيد القيمة 0 بنسبة 50 % والقيمة 1 بنسبة 50 %. يمكنك أن تستفيد من القالب الآتي:

```
import random
numberOfStreaks = 0
for experimentNumber in range(10000):
  # الشيفرة التي ستولد قائمة من 100 قيمة عشوائية لعملية رمي القطعة النقدية

  # الشيفرة التي ستتحقق من ظهور 6 رؤوس أو 6 نقوش متتالية
print('Chance of streak: %s%%' % (numberOfStreaks / 100))
```

تـذكر أن الـرقم النـاتج تقريـبي عملي، لكن حجم العينـة (عشـرة آلاف) مناسـب؛ لكن إذا كنت تعـرف بعض المبادئ الأساسية في الاحتمالات والإحصاء الرياضي فستعرف الإجابة الدقيقـة دون الحاجـة إلى كتابـة البرنـامج السابق، لكن من الشائع أن تكون معرفة المبرمجين بالرياضيات تعيسة (على عكس المتوقع).

4.10.3 صورة حرفية

لنقل أن لدينا قائمة تضم قوائم أخرى، وكل قيمة في القوائم الداخلية هي حرف واحد كما يلي:

```
grid = [['.', '.', '.', '.', '.', '.'],
   ['.', 'O', 'O', '.', '.', '.'],
   ['O', 'O', 'O', 'O', '.', '.'],
   ['O', 'O', 'O', 'O', 'O', '.'],
   ['.', 'O', 'O', 'O', 'O', 'O'],
   ['O', 'O', 'O', 'O', 'O', '.'],
   ['O', 'O', 'O', 'O', '.', '.'],
   ['.', 'O', 'O', '.', '.', '.'],
   ['.', '.', '.', '.', '.', '.']]
```

تخيل أن العنصر `grid[x][y]` هو المحرف الموجود في الإحداثيات x و y «للصورة» الـتي سنرسـمها عـبر الأحـرف. مبـدأ الإحـداثيات `(0, 0)` هـو الـركن العلـوي الأيسـر، وسـتزداد إحـداثيات `x` بالـذهاب نحـو اليمين، وإحداثيات `y` نحو الأسفل.

انسخ الشبكة السابقة واكتب شيفرة لطباعة الشكل التالي منها:

```
..OO.OO..
.OOOOOOO.
.OOOOOOO.
..OOOOO..
...OOO...
....O....
```

تلميحة: ستحتاج إلى حلقة تكرار داخل حلقة تكـرار، لكي تطبـع `grid[0][0]` ثم grid1 ثم `grid[2][0]` وهلم جرًا إلى أن تصل إلى `grid[8][0]`؛ ثم ستنتهي من أول صف وتطبع سـطرًا جديـدًا، ثم تطبـع `grid0` ثم `grid[1][1]` ثم `grid2` ...إلخ. وآخر عنصر سيطبعه برنامجك هو `grid[8][5]`.

تـذكر أن تمـرر الوسـيط المسـمى `end` إلى الدالـة `print()` إذا لم تكن تريـد طباعـة سـطر جديـد بعـد كـل استدعاء للدالة `print()`.

4.11 الخلاصة

القوائم هي نوع بيانات مفيد يسمح لك بكتابة شيفرات تتعامـل مـع قيم متعـددة مخزنـة في متغـير واحـد. سنرى برامج في فصول هذا الكتاب كان من المستحيل برمجتها دون الاعتماد على القوائم.

القوائم هي نوع من أنواع البيانات المتسلسلة القابلة للتعـديل. أي أن محتوياتهـا قـد تتعـدل برمجيًـا. بينمـا تكون الصفوف tuple والسلاسل النصية من أنواع البيانات المتسلسلة لكنها غير قابلة للتعديل.

يمكن إعادة كتابة قيمة متغير يحتوي على سلسلة نصية أو صـف بقيمـة أخـرى، لكن هـذا لا يكـافئ تعـديل قيمة السلسلة النصية أو الصف في مكانها، مثلما تفعل التوابع `append()` أو `remove()` على القوائم.

لا تخزن المتغيرات قيم القوائم مباشـرةً فيهـا، بـل هي تشـير إلى تلـك القـوائم، ومن المهم اسـتيعاب هـذه النقطة حين نسخ المتغيرات أو تمرير القوائم كوسـائط إلى الـدوال. ولأن القيمـة الـتي ستنسـخ هي مرجعيـة إلى القائمة وليست القائمة نفسها، فأي تعديل يجرى على القائمة سيؤثر عليهـا في كامـل البرنـامج؛ لكننـا نسـتطيع استخدام الدالة `copy()` أو `deepcopy()` لنسخ القوائم ثم إجراء تعديلات عليها لا تؤثر على القائمة الأصلية.

دورة تطوير تطبيقات الويب
باستخدام لغة Ruby

مميزات الدورة

- بناء معرض أعمال قوي بمشاريع حقيقية
- وصول مدى الحياة لمحتويات الدورة
- تحديثات مستمرة على الدورة مجانًا
- شهادة معتمدة من أكاديمية حسوب
- إرشادات من المدربين على مدار الساعة
- من الصفر دون الحاجة لخبرة مسبقة

اشترك الآن

5. القواميس وهيكلة البيانات

سنشرح في هذا الفصل نوع البيانات المسمى بالقاموس dictionary، الذي يوفر طريقة مرنة للوصـول إلى البيانات وتنظيمها، ثم سنتعلم كيفيـة كتابـة لعبـة إكس-أو عـبر دمج مـا تعلمنـاه في الفصـول السـابقة من هـذا الكتاب مع القواميس.

5.1 نوع البيانات dictionary

القواميس تشبه القوائم في أنها مجموعة قابلة للتعديل mutable من القيم، لكن بدلًا من وجـود الفهـارس الرقمية للعناصر كما في القوائم، يمكن استخدام مختلف أنواع البيانات في القواميس.

نسمي المفاتيح في القواميس بالمفاتيح keys، وكل مفتاح مرتبط بقيمة، ونسمي ذلك زوجًـا من المفـاتيح والقيم key-value pair. نكتب القواميس في بايثون بإحاطها بقوسين مجعدين أو معقوصين {}:

```
>>> myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
```

السـطر السـابق يسـند قاموسًـا إلى المتغـير myCat، ومفـاتيح هـذا القـاموس هي 'size' و 'color' و'disposition'، والقيم المرتبطـــة بتلـــك المفـــاتيح هي 'fat' و 'gray' و 'loud' على التـــوالي وبالترتيب. يمكنك الوصول إلى هذه القيم عبر مفاتيحها:

```
>>> myCat['size']
'fat'
>>> 'My cat has ' + myCat['color'] + ' fur.'
'My cat has gray fur.'
```

ما يزال بالإمكان استخدام الأعداد الصحيحة مفاتيحًا لقيم القواميس، لكن ليس من الضروري أن تبدأ من الصفر 0 ويمكنك استخدام أي رقم:

```
>>> spam = {12345: 'Luggage Combination', 42: 'The Answer'}
```

5.1.1 مقارنة القواميس والقوائم

على خلاف القوائم، لا تكون القواميس مرتبةً، فأول عنصر في قائمة اسمها `spam` سيكون `spam[0]`، لكن لا يوجد «أول» عنصر في القاموس. سيكون ترتيب العناصر مهمًا في حال أردنا تحديد إن كانت قائمتان متساويتين، بينما لا يفرق ترتيب كتابة أزواج المفاتيح-القيم في القواميس:

```
>>> spam = ['cats', 'dogs', 'moose']
>>> olive = ['dogs', 'moose', 'cats']
>>> spam == olive
False
>>> eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}
>>> steak = {'species': 'cat', 'age': '8', 'name': 'Zophie'}
>>> eggs == steak
True
```

ولأن القواميس غير مرتبة، فلا يمكن تقسيمها كما في القوائم.

محاولة الوصول إلى مفتاح غير موجود في قاموس ما سيؤدي إلى حدوث الخطأ KeyError، وهي تشبه رسالة الخطأ IndexError في القوائم حين محاولة الوصول إلى قيمة خارج مجال فهارس القائمة. لاحظ رسالة الخطأ الآتية التي تظهر لعدم وجود المفتاح `'color'`:

```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> spam['color']
Traceback (most recent call last):
 File "<pyshell#1>", line 1, in <module>
  spam['color']
KeyError: 'color'
```

وصحيحٌ أن القواميس غير مرتبة، لكن إمكانية استخدام أي قيمة تريدها للمفاتيح يسمح لنا بترتيب البيانات بطرائق رائعة!

لنقل أننا نريد كتابة برنامج يخزن معلومات حول أعياد ميلاد أصدقائك، يمكنك أن تكتب الشيفرة الآتية التي تستخدم القواميس وتجعل أسماء أصدقائك مفاتيحًا للقيم: `birthdays.py`

```
❶ birthdays = {'Alice': 'Apr 1', 'Bob': 'Dec 12', 'Carol': 'Mar 4'}

 while True:
   print('Enter a name: (blank to quit)')
   name = input()
   if name == '':
     break

 ❷ if name in birthdays:
   ❸ print(birthdays[name] + ' is the birthday of ' + name)
   else:
     print('I do not have birthday information for ' + name)
     print('What is their birthday?')
     bday = input()
   ❹ birthdays[name] = bday
     print('Birthday database updated.')
```

أنشأنا قاموسًا وخزناه في المتغير `birthdays` ❶، ثم تحققنا إن كان الاسم المدخل موجودًا في القاموس عبر الكلمة المحجوزة `in` ❷ كما كنا نفعل مع القوائم.

إذا كان الاسم موجـودًا في القـاموس فيمكنـك الوصـول إلى القيمـة المرتبطـة بـه عـبر اسـتخدام الأقـواس المربعة❸، وإلّا فيمكنك إضافتها باستخدام صيغة الأقواس المربعة مع عامل الإسناد ❹:

```
Enter a name: (blank to quit)
Alice
Apr 1 is the birthday of Alice
Enter a name: (blank to quit)
Eve
I do not have birthday information for Eve
What is their birthday?
```

```
Dec 5
Birthday database updated.
Enter a name: (blank to quit)
Eve
Dec 5 is the birthday of Eve
Enter a name: (blank to quit)
```

للأسف ستحذف كل المعلومات التي أدخلتها في هـذا البرنـامج حين انتهـاء تنفيـذه. سـتتعلم كيفيـة حفـظ الملفات على القرص في الفصل التاسع من هذا الكتاب.

5.1.2 القواميس المرتبة في بايثون 3.7

صحيح أن القواميس غير مرتبة ولا يوجد فيها عنصر «أول»، لكن القواميس في الإصدار 3.7 من بايثون وما يليه ستتذكر ترتيب أزواج القيم الموجودة فيها حين إنشاء متسلسـل `sequence` منهـا. فمثلًا لاحـظ أن تـرتيب العناصر في القائمتين المنشأتين من القاموسين `eggs` و `steak` يطابق ترتيب إدخالها:

```
>>> eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}
>>> list(eggs)
['name', 'species', 'age']
>>> steak = {'species': 'cat', 'age': '8', 'name': 'Zophie'}
>>> list(steak)
['species', 'age', 'name']
```

ستبقى القواميس غير مرتبة، ولا يمكنك أن تصل إلى العناصـر فيهـا عـبر فهـرس رقمي مثـل `eggs[0]` أو `steak[2]`، لا يجدر بك الاعتماد على هذا السلوك لأن بايثون لا تتـذكر تـرتيب إدخـال العناصـر في الإصـدارات القيمة منها، فلاحظ المثال الآتي الذي لا يطابق ترتيب عناصر القاموس الناتج النهائي الترتيبَ الذي أدخلتها بـه، ناتج التنفيذ الآتي على إصدار بايثون 3.5:

```
>>> spam = {}
>>> spam['first key'] = 'value'
>>> spam['second key'] = 'value'
>>> spam['third key'] = 'value'
>>> list(spam)
['first key', 'third key', 'second key']
```

5.1.3 التوابع keys() و values() و items()

هنالك ثلاثة توابع خاصة بـالقواميس الـتي تعيـد قيمًـا شـبيهة بـالقوائم list-like من مفـاتيح القـواميس أو قيمها أو كلًا من المفاتيح والقيم معًا وهي التوابع `keys()` و `values()` و `items()` بالترتيب.

القيم المعادة من هذه التوابع ليست قوائم حقيقيةً، فلا يمكننا تعـديلها وليس لهـا التـابع `append()`، لكن أنواع البيانات المعادة (وهي dict_keys و dict_values و dict_items بالترتيب) يمكن أن تستخدم في حلقات التكرار `for`:

```
>>> spam = {'color': 'red', 'age': 42}
>>> for v in spam.values():
...     print(v)

red
42
```

ستمر حلقة `for` هنا على كل قيمة في القاموس `spam`، يمكن لحلقة `for` المرور على المفاتيح فقط، وعلى المفاتيح والقيم معًا:

```
>>> for k in spam.keys():
...     print(k)

color
age
>>> for i in spam.items():
...     print(i)

('color', 'red')
('age', 42)
```

حين استخدامنا للتوابع `keys()` و `values()` و `items()` فيمكن للحلقة `for` المـرور على قيم المفـاتيح أو قيم العناصر أو قيم أزواج المفتـاح-القيمـة على التـوالي. لاحـظ أن القيم المعـادة من `items()` هي صـفوف tuples تحتوي على المفتاح ثم قيمته. إذا أردتَ قائمةً حقيقية من ناتج أحد تلك التوابع، فيمكننـا تمريـر القيمـة الشبيهة بالقوائم إلى الدالة `list()` كما يلي:

```
>>> spam = {'color': 'red', 'age': 42}
>>> spam.keys()
dict_keys(['color', 'age'])
```

```
>>> list(spam.keys())
['color', 'age']
```

يأخذ السطر `list(spam.keys())` القيمة ذات النوع dict_keys المعـادة من التـابع `keys()` ويمررهـا إلى الدالة `list()`، والتي تعيد بدورها قائمةً فيها `['color', 'age']`.

يمكنك استخدام الإسناد المتعدد مع حلقة `for` لإسناد المفتاح والقيمة إلى متغيرات منفصلة:

```
>>> spam = {'color': 'red', 'age': 42}
>>> for k, v in spam.items():
...     print('Key: ' + k + ' Value: ' + str(v))

Key: age Value: 42
Key: color Value: red
```

5.1.4 التحقق من وجود مفتاح أو قيمة في قاموس

نتذكر من الفصل السابق أن العاملين `in` و `not in` يمكن أن يستخدما للتحقق من وجود قيمة في قائمة. ويمكننا استخدام نفس العاملين للتحقق من وجود قيمة ما في مفتاح أو قيمة في قاموس:

```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> 'name' in spam.keys()
True
>>> 'Zophie' in spam.values()
True
>>> 'color' in spam.keys()
False
>>> 'color' not in spam.keys()
True
>>> 'color' in spam
False
```

لاحظ أن كتابة `'color' in spam` هي مطابقـة لكتابـة `'color' in spam.keys()`. فـإذا أردت التحقق من وجـود (أو عـدم وجـود) مفتـاح مـا في قـاموس فاسـتعمل الكلمـة المحجـوزة `in` (أو `not in`) مـع القاموس نفسه.

5.1.5 التابع get()

من الرتيب أن نتحقق من وجود مفتاح ما في القاموس قبل الوصول إلى القيمة المرتبطة به، لكن لحسن الحظ هنالك تابع باسم `get()` يأخذ وسيطين: الأول هو المفتاح الذي نريد الوصول إلى قيمته، والثاني هو قيمة افتراضية ستعاد إن لم يكن المفتاح موجودًا:

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems.get('cups', 0)) + ' cups.'
'I am bringing 2 cups.'
>>> 'I am bringing ' + str(picnicItems.get('eggs', 0)) + ' eggs.'
'I am bringing 0 eggs.'
```

ولعدم وجود المفتاح `'eggs'` في القاموس picnicItems فستعاد القيمة `0` من التابع `get()`، وإن لم نستعمل التابع `get()` في المثال السابق فستظهر رسالة خطأ كما يلي:

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
Traceback (most recent call last):
 File "<pyshell#34>", line 1, in <module>
  'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
KeyError: 'eggs'
```

5.1.6 التابع setdefault()

الحاجة إلى ضبط قيمة في القاموس مرتبطة بمفتاح معين إن لم يكن ذاك المفتاح موجودًا مسبقًا هو أمرٌ شائع، وتكون الشيفرة بالشكل التالي:

```
spam = {'name': 'Pooka', 'age': 5}
if 'color' not in spam:
  spam['color'] = 'black'
```

يوفر التابع `setdefault()` طريقة أسهل لفعل ذلك بسطر برمجي وحيد، فأول وسيط يمرر إلى التابع هو المفتاح الذي سنتحقق من وجوده، والوسيط الثاني هو القيمة التي ستُضبَط إن لم يكن المفتاح موجودًا.

إذا كان المفتاح موجودًا فسيعيد التابع `setdefault()` قيمة ذاك المفتاح:

```
>>> spam = {'name': 'Pooka', 'age': 5}
>>> spam.setdefault('color', 'black')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
>>> spam.setdefault('color', 'white')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
```

حينما استدعينا التابع `setdefault()` أول مرة، فتغيرت قيمة القاموس `spam` إلى `{'color': 'black', 'age': 5, 'name': 'Pooka'}`، وسيعيد التابع `setdefault()` القيمة `'black'` لأنها القيمة المضبوطة للمفتاح `'color'` حاليًا.

لكن حين استدعاء `spam.setdefault('color', 'white')` فإن القيمة لن تتغير إلى `'white'` لأن القاموس `spam` يحتوي على مفتاح باسم `'color'`.

التابع `setdefault()` هو اختصار جميل للتأكد من وجود مفتاح معين وضبط قيمته. هذا مثال عن برنامج يعدّ عدد مرات وجود كل حرف في سلسلة نصية. احفظ المثال الآتي باسم `characterCount.py`:

```
message = 'It was a bright cold day in April, and the clocks were
striking
thirteen.'
count = {}

for character in message:
❶ count.setdefault(character, 0)

❷ count[character] = count[character] + 1


print(count)
```

سيمر البرنامج على كل محرف في السلسلة النصية المخزنة في المتغير `message`، ويعد كم مرة يظهر فيها كل محرف.

استدعاء الدالة `setdefault()` ❶ سيضمن وجود المفتاح في القاموس `count` ويضبط قيمته الافتراضية إلى الصفر 0، وبالتالي لا يرمي البرنامج الخطأ KeyError حين تنفيذ التعبير البرمجي `count[character] = count[character] + 1` ❷. سيبدو الناتج كما يلي:

```
{' ': 13, ',': 1, '.': 1, 'A': 1, 'I': 1, 'a': 4, 'c': 3, 'b': 1, 'e':
5, 'd': 3, 'g': 2,
'i': 6, 'h': 3, 'k': 2, 'l': 3, 'o': 2, 'n': 4, 'p': 1, 's': 3, 'r':
5, 't': 6, 'w': 2, 'y': 1}
```

سترى من الناتج السابق أن الحرف `c` الصغير مكرر 3 مرات، بينما الفراغ مكرر 13 مرة، والحرف `A` الكبير يظهر مرة واحدة. سيعمل البرنامج السابق على جميع السلاسل النصية بغض النظر عن محتويات المتغير `message` حتى لو كان يحتوي على مليون حرف!

5.2 تجميل الطباعة

إذا استوردت الوحدة `pprint` في برامجك، فيمكنك الوصول إلى الدالتين `pprint()` و `pformat()` التي «تجمل طباعة pretty print» قيم القواميس، ستستفيد من هذه الدوال إن أردت عرض قيم القواميس عرضًا أجمل من طريقة عرض الدالة `print()`، لنعدل المثال السابق:

```
import pprint
message = 'It was a bright cold day in April, and the clocks were
striking
thirteen.'
count = {}

for character in message:
  count.setdefault(character, 0)

  count[character] = count[character] + 1


pprint.pprint(count)
```

سيظهر لنا الناتج الجميل الآتي:

```
{' ': 13,
 ',': 1,
 '.': 1,
 'A': 1,
 'I': 1,
 --snip--
```

```
 't': 6,
 'w': 2,
 'y': 1}
```

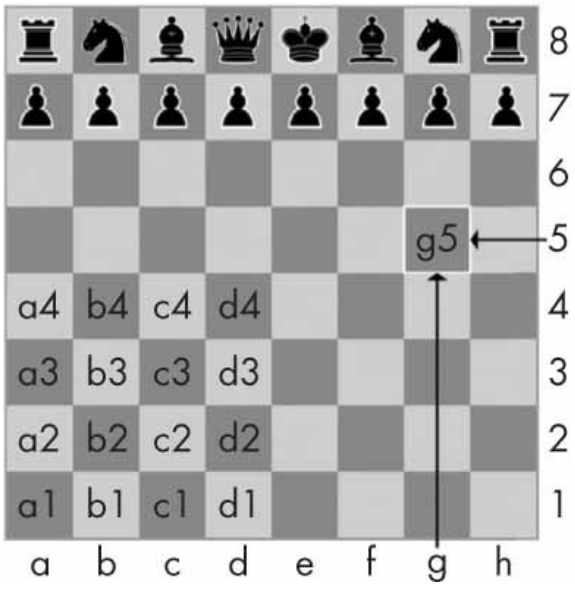
سنستفيد فعليًا من الدالة `pprint.pprint()` عندما يحتوي القاموس على قـوائم أو قـواميس متشـعبة داخله nested.

إذا أردت الحصـول على قيمـة النص المجمّـل بـدلًا من طباعتـه على الشاشـة مباشـرةً، فاسـتدعِ الدالـة `pprint.pformat()`. السطران الآتيان متكافئان تمامًا:

```
pprint.pprint(someDictionaryValue)
print(pprint.pformat(someDictionaryValue))
```

استخدام بنى المعطيات لنمذجة عناصر حقيقية كان بإمكاننا لعب الشطرنج مع شخص آخـر عن بعـد قليـل ظهور الإنترنت، فكان يعد كل لاعب رقعة الشطرنج في منزلـه، ثم يبـادلان الرسـائل البريديـة يصـف كـل منهمـا خطوته، ولكي يستطيعوا فعل ذلك كان لاعبو الشطرنج بحاجة إلى وصف حالـة رقعـة الشـطرنج والحركـات الـتي يجريها وصفًا لا لبس فيه.

تمثل الفراغات في رقعة الشطرنج في التأشير الجبري Algebraic chess notation بإحداثيات تتألف من حرف ورقم كما في الشكل التالي.



الشكل 30: إحداثيات رقعة الشطرنج في التأشير الجبري

تُعرّف قطع الشطرنج بالأحرف: `K` للملك king، و `Q` للملكة queen (يسميها البعض بالوزير)، و `R` للقلعة rook، و `B` للفيل bishop، و `N` للحصان knight. أما الجنود فلا رمز لهم.

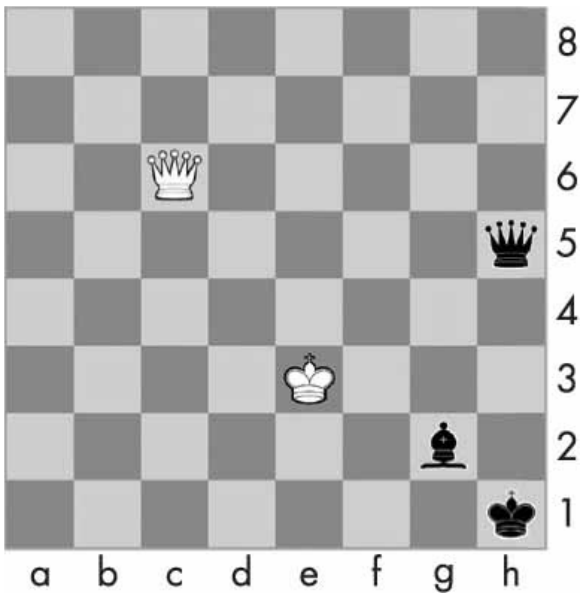
يكون وصف الحركات متألفًا من الحرف الذي يمثل القطعة، وإحداثيات الوجهة. وزوج من تلك الحركات يصف ما يحدث في دورٍ واحد (بفرض أن صاحب اللون الأبيض يبدأ أولًا)؛ فمثلًا التأشير `2. Nf3 Nc6` يعني أن الأبيض حرك الحصان إلى `f3` والأسود حرك الحصان إلى `c6` في الدور الثاني من اللعبة.

هنالك المزيد من القواعد للتأشير الجبري للشطرنج، لكن الفكرة التي أحاول إيصالها هي أننا نستطيع وصف لعبة الشطرنج دون الحاجة إلى أن نكون أمام رقعة، ويمكن أن يكون خصمك في الطرف الثاني من الكوكب، وإذا كانت لديك ذاكرة ومخيلة جيدة فلا تحتاج إلى رقعة شطرنج حقيقية من الأساس: فيمكنك أن تقرأ الحركات من الرسائل البريدية وتحدِّث الرقعة الموجودة في مخيلتك!

تمتلك الحواسيب ذواكر رائعة، فيمكن أن يخزن الحاسوب مليارات السلاسل النصية من الشكل `'2. Nf3 Nc6'`، وبهذا يمكن أن تلعب الحواسيب الشطرنج دون لوحة حقيقية؛ فما تفعله الحواسيب هو نمذجة البيانات لتمثيل رقعة الشطرنج، ويمكنك كتابة شيفرة تفعل ذلك بنفسك.

هنا تلعب القوائم والقواميس دورها، فمثلًا القاموس التالي يمثل الرقعة في الشكل الآتي.

```
{'1h': 'bking', '6c': 'wqueen', '2g': 'bbishop', '5h': 'bqueen', '3e':
'wking'}
```

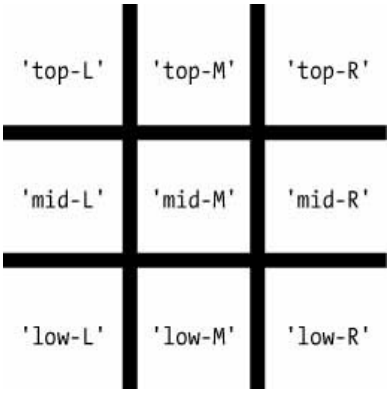


الشكل 31: رقعة شطرنج منمذجة وفق قيمة قاموس

لكن لمثالنا القادمة سنستعمل لعبة أسهل وأبسط من الشطرنج وهي لعبة إكس-أو.

5.2.1 لعبة إكس-أو X-0

لعبة إكس-أو (تسمى بالإنكليزية tic-tac-toe) تشبه رمز # كبير فيه 9 خانـات يمكن أن تكـون قيمهـا X أو O أو أن تكون فارغة. لتمثيل هذه الرقعة بقاموس، فيجب أن نسند لكل خانة زوجًا من المفتاح.



الشكل 32: خانات رقعة إكس-أو مع
المفاتيح الموافقة لها

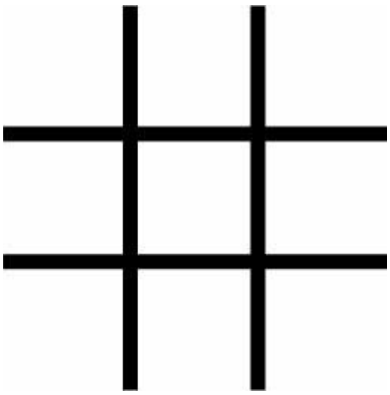
يمكنك استخدام القيم النصية لتمثيل ما هو موجود في كـل خانـة في الرقعـة: `'x'` أو `'o'` أو `' '` (فـراغ)، وبالتالي نحتاج إلى تسع سلاسل نصية.

يمكنك استخدام قاموس من القيم لهذا الأمر، فالقيمة النصية المرتبطة مع المفتاح `'top-R'` تمثل القيمة في الركن العلـوي الأيمن، والسلسـل النصـية المرتبطـة مـع المفتـاح `'low-L'` تمثـل الـركن السـفلي الأيسـر، والسلسلة النصية المرتبطة مع المفتاح `'mid-m'` تمثل المنتصف، وهلم جرًا للبقية.

هـذا القـاموس هـو بنيـة معطيـات تمثـل رقعـة لعبـة إكس-أو، ولنخـزن هـذا القـاموس في متغـير باسـم `theBoard`، ولنحفظ الشيفرة الآتية في ملف باسم `ticTacToe.py`:

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
     'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
     'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

بنية المعطيات المخزنة في المتغير `theBoard` تمثل رقعة إكس-أو الموضحة في الشكل التالي.

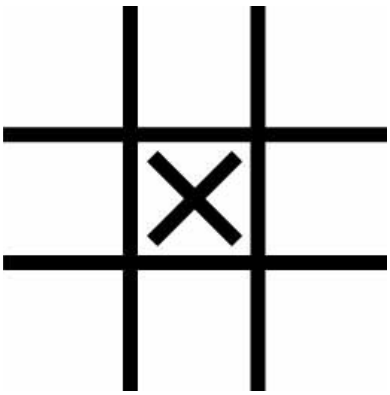


الشكل 33: لوحة إكس-أو فارغة

ولما كانت قيمة كل مفتاح في القاموس `theBoard` هي فراغ واحد فيمثل ذاك القاموس رقعةً فارغةً تمامًا. وإذا بدأت اللاعب X واختار الخانة في المنتصف تمامًا فسيصبح القاموس الذي يمثل الرقعة على الشكل:

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
     'mid-L': ' ', 'mid-M': 'X', 'mid-R': ' ',
     'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

أصبحت بنية المعطيات `theBoard` تمثل الرقعة الموضحة في الشكل التالي.

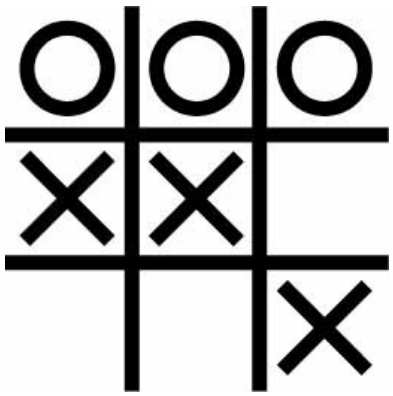


الشكل 34: الحركة الأولى

وتكون رقعة ربح فيها اللاعب O بوضع الشكل O في الصف العلوي كما يلي:

```
theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O',
     'mid-L': 'X', 'mid-M': 'X', 'mid-R': ' ',
     'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}
```

وهي ممثلة في الشكل التالي.



الشكل 35: ربح اللاعب O

وبالتأكيد لا يستطيع أن يرى اللاعب إلا ما يطبع على الشاشة أمامه، ولا يعرف محتويات المتغيرات، فلننشئ دالةً تطبع القاموس الذي يحتوي على الرقعة على الشاشة. أضف ما يلي إلى ملف `ticTacToe.py`:

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
     'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
     'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
def printBoard(board):
  print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
  print('-+-+-')
  print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
  print('-+-+-')
  print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])
printBoard(theBoard)
```

حينما تشغل هذا البرنامج فستطبع رقعة إكس-أو فارغة:

```
  |  |
-+-+-
  |  |
-+-+-
  |  |
```

يمكن أن تتوالى الدالة `printBoard()` أي بنية معطيات تمثل رقعة إكس-أو تمررها إليها، جرب تغيير الشيفرة إلى ما يلي:

```
theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O', 'mid-L': 'X',
'mid-M':
'X', 'mid-R': ' ', 'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}


def printBoard(board):
  print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
  print('-+-+-')
  print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
  print('-+-+-')
  print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])
printBoard(theBoard)
```

ستعرض الرقعة الآتية على الشاشة:

```
O|O|O
-+-+-
X|X|
-+-+-
 | |X
```

ولأنك أنشأت بنية معطيات تمثل لوحة إكس-أو وكتبت الشيفرة في `printBoard()` التي تفسر بنية المعطيات وتظهر الرقعة، فأنت كتبت برنامجًا «ينمذج models» رقعة إكس-أو. كان بإمكانك تنظيم البيانات في بنية المعطيات بطريقة مختلفة، فمثلًا يمكنك استخدام المفتاح `'TOP-LEFT'` بدلًا من `'top-L'`، لكن طالما كانت شيفرتك تعمل مع بنية المعطيات التي لديك، فأنت كتبت عملية النمذجة بشكل صحيح.

فمثلًا تتوقع الدالة `printBoard()` أن بنية المعطيات التي تمثل الرقعة هي قاموس فيه مفاتيح لجميع الخانات التسع، لكن إن كان في قاموسك مفتاحٌ ناقص وليكن `'mid-L'` فلن يعمل برنامجك:

```
O|O|O
-+-+-
Traceback (most recent call last):
 File "ticTacToe.py", line 10, in <module>
```

```
  printBoard(theBoard)

 File "ticTacToe.py", line 6, in printBoard

  print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])

KeyError: 'mid-L'
```

لنضف الآن الشيفرة التي تسمح للاعبين بإدخال حركاتهم. لنعدل برنامجنا ليبدو كما يلي:

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ', 'mid-L': ' ',
'mid-M':
' ', 'mid-R': ' ', 'low-L': ' ', 'low-M': ' ', 'low-R': ' '}


def printBoard(board):
  print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])

  print('-+-+-')

  print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])

  print('-+-+-')

  print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])

turn = 'X'
for i in range(9):
 ❶ printBoard(theBoard)

  print('Turn for ' + turn + '. Move on which space?')

 ❷ move = input()

 ❸ theBoard[move] = turn

 ❹ if turn == 'X':

    turn = 'O'

  else:

    turn = 'X'

 printBoard(theBoard)
```

الشيفرة الجديدة تطبـع اللوحـة في بدايـة كـل دور ❶ ثم تطلب المـدخلات من اللاعب الحـالي ❷ ثم تحـدث الرقعة وفقًا لذلك ❸ ثم تبدل اللاعب الحالي ❹ قبل الانتقال إلى الدور القادم.

```
 | |
-+-+-
 | |
-+-+-
 | |
Turn for X. Move on which space?
mid-M
 | |
-+-+-
 |X|
-+-+-
 | |

--snip--

O|O|X
-+-+-
X|X|O
-+-+-
O| |X
Turn for X. Move on which space?
low-M
O|O|X
-+-+-
X|X|O
-+-+-
O|X|X
```

صحيحٌ أن البرنامج ليس لعبة إكس-أو كاملة، فلن يتحقق إن ربح أحد اللاعبين مثلًا؛ لكنه كافٍ لمعرفة كيفيـة استخدام بنى المعطيات في برامج حقيقية.

5.3 القواميس والقوائم المتشعبة

تُعَد نمذجة لوحة إكس-أو X-O أمرًا سهلًا للغاية: تحتاج اللوحة إلى قاموس فيه 9 مفاتيح تمثل خاناتها. لكن إن أردت نموذج أمور أكثر تعقيدًا فستجد أنك تحتاج إلى القواميس والقوائم التي تحتوي على قواميس وقوائم أخرى داخلها.

تناسب القوائم تخزين سلسلة مرتبة من القيم، بينما تفيد القواميس بتخزين قواميس التي ترتبط فيها المفاتيح مع القيم.

هذا مثال يستعمل قاموسًا يحوي قواميس داخله فيها الأغراض التي أتى بها الضيوف إلى الرحلة. يمكن أن تقرأ الدالة `totalBrought()` بنية المعطيات وتحسب العدد الكلي للعناصر المجلوبة من كل الضيوف:

```
allGuests = {'Alice': {'apples': 5, 'pretzels': 12},
             'Bob': {'steak sandwiches': 3, 'apples': 2},
             'Carol': {'cups': 3, 'apple pies': 1}}

def totalBrought(guests, item):
  numBrought = 0
  ❶ for k, v in guests.items():
    ❷ numBrought = numBrought + v.get(item, 0)
  return numBrought

print('Number of things being brought:')
print(' - Apples         ' + str(totalBrought(allGuests, 'apples')))
print(' - Cups           ' + str(totalBrought(allGuests, 'cups')))
print(' - Cakes          ' + str(totalBrought(allGuests, 'cakes')))
print(' - Steak Sandwiches ' + str(totalBrought(allGuests, 'steak
sandwiches')))
print(' - Apple Pies     ' + str(totalBrought(allGuests, 'apple
pies')))
```

داخل الدالة `totalBrought()` هنالك حلقة `for` تدور على أزواج مفتاح-قيمة في المتغير `guests` ❶، وسنسند داخل الحلقة اسم كل ضيف إلى المتغير `k`، وسنسند القاموس الذي يحتوي على قائمة الأغراض التي سيجلبها معه إلى الرحلة إلى المتغير `v`. إذا كان أحد المعامل `item` موجودة في القاموس، فستضاف قيمته

(كمية الأغراض المجلوبة) إلى المتغير `numBrought` ❷، لكن إذا لم يكن المفتاح موجودًا فيسعيد التابع `get()` القيمة `0` لإضافتها إلى `numBrought`.

سيكون ناتج تنفيذ البرنامج كما يلي:

```
Number of things being brought:
- Apples 7
- Cups 3
- Cakes 0
- Steak Sandwiches 3
- Apple Pies 1
```

قد يبدو لك أن حالة الاستخدام السابقة بسيطة ولا حاجة إلى نمذجتها، لكن فكر أن الدالة `totalBrought()` يمكن أن تتعامل بسهولة مع قاموس يحتوي على آلاف الضيوف، وكل ضيف يجلب آلاف العناصر، وتنظيم هذه المعلومات في بنية معطيات واضحة ووجود الدالة `totalBrought()` سيوفر عليك وقتًا كثيرًا.

يمكنك أن تنمذج العناصر في بنى المعطيات بالطريقة التي تراها مناسبة، لطالما كانت بقية الشيفرة في برنامج قادرةً على التعامل مع بنية المعطيات المنشأة.

حينما تبدأ البرمجة فلا تقلق أن تنمذج البيانات بالطريقة «الصحيحة»، فكلما ازدادت خبرتك أصبحت تخطر ببالك طرائق أكثر فاعلية وكفاءة للنمذجة، لكن أهم ما في الأمر أن تعمل بنية المعطيات مع احتياجات برنامجك.

5.4 أسئلة للتدريب

1. كيف تبدو الشيفرة التي تمثل قاموسًا فارغًا؟
2. كيف نكتب قاموسًا فيه عنصر مفتاحه `'foo'` وقيمته `42`؟
3. ما الفرق الرئيسي بين القواميس والقوائم؟
4. ماذا يحدث حين محاولة الوصول إلى `spam['foo']` إذا كانت قيمة `spam` هي `{'bar': 100}`؟
5. إذا كان هنالك قاموسٌ مخزنٌ في المتغير `spam`، فما الفرق بين استخدام التعبيرين `'cat' in spam` و`'cat' in spam.keys()`؟

6. إذا كان هنالك قاموسٌ مخزنٌ في المتغير `spam`، فما الفرق بين استخدام التعبيرين `'cat' in spam` و`'cat' in spam.values()`؟

7. ما اختصار الشيفرة الآتية:

```
if 'color' not in spam:
spam['color'] = 'black'
```

8. ما الوحدة والدالة التي تستخدم لتجميل طابعة القواميس؟

5.5 مشاريع للتدريب

لكي تتدرب، اكتب برامج لتنفيذ المهام الآتية.

5.5.1 مدقق لقواميس الشطرنج

استعملنا في هذا الفصل قيمةً مثل:

```
{'1h': 'bking', '6c': 'wqueen', '2g': 'bbishop', '5h': 'bqueen', '3e':
'wking'}
```

لتمثيل رقعة الشطرنج. اكتب دالةً باسم `isValidChessBoard()` التي تقبل وسيطًا هو قاموس وتعيد القيمة `True` أو `False` اعتمادًا إن كان القاموس صالحًا لتمثيل رقعة الشطرنج.

تحتوي الرقعة السليمة على ملك أسود واحد وملك أبيض واحد. ويمكن لأيٍ من اللاعبين امتلاك 16 قطعة كحد أقصى، و 8 جنود كحد أقصى، ويجب أن تكون جميع القطع في المجال بين 1a و 8h، أي لا يمكن أن تكون القطعة في المكان 9z. يجب أن تبدأ أسماء القطع بحرف w أو b لتمثيل اللونين الأبيض أو الأسود، متبوعًا بإحدى الكلمات pawn أو knight أو bishop أو rook أو queen أو king.

5.5.2 قائمة الأدوات في لعبة

نحن نعمل على لعبة فيها قائمة أدوات يمكن أن يمتلكها اللاعب، والتي سننمذجها باستخدام بنية معطيات تتألف من قاموس تكون فيه مفاتيحه هي سلاسل نصية تصف القيمة الموجودة في قائمة الأدوات، وقيمتها هي عدد نصي يمثل عدد الأدوات التي يمتلكها اللاعب. مثلًا القاموس الآتي:

```
{'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}
```

يعني أن اللاعب يملك حبلًا واحدًا، و 6 شعلات، و 42 قطعة ذهبية ...إلخ.

اكتب دالةً باسم `displayInventory()` التي تأخذ أي قاموس يمثل قائمة أدوات ويعرضه بالشكل:

```
Inventory:
12 arrow
42 gold coin
1 rope
6 torch
1 dagger
Total number of items: 62
```

تلميح: يمكنك استخدام حلقة `for` للمرور على جميع المفاتيح في القاموس:

```
# inventory.py
stuff = {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow':
12}


def displayInventory(inventory):
  print("Inventory:")

  item_total = 0

  for k, v in inventory.items():

    # أكمل الشيفرة هنا

  print("Total number of items: " + str(item_total))


displayInventory(stuff)
```

5.5.3 دالة تحويل قائمة إلى قاموس في لعبة

لنفترض أن لاعبنا في اللعبة السابقة قد حصل على غنيمة ممثلة في القائمة الآتية:

```
playerLoot = ['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby']
```

اكتب دالةً باسم `addToInventory(inventory, addedItems)` حيث أن المعامل `inventory` هو قاموس يمثل قائمة أدوات اللاعب (كما في المثال السابق) والمعامل `addedItems` يشبه المتغير `playerLoot`. يجب أن تعيد الدالة `addToInventory()` قاموسًا يمثل قائمة الأدوات المحدثة.

لاحظ أن القائمة الموجودة في `addedItems` قد تحتوي على عدة نسخ من نفس الأداة.

يفترض أن تكون شيفرتك شبيهة بما يلي:

```
def addToInventory(inventory, addedItems):
  # اكتب الدالة هنا


inv = {'gold coin': 42, 'rope': 1}
dragonLoot = ['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby']
inv = addToInventory(inv, dragonLoot)
displayInventory(inv)
```

يجب أن يظهر البرنامج السابق (مع الدالة `displayInventory()` من المثال السابق) الناتج الآتي:

```
Inventory:
45 gold coin
1 rope
1 ruby
1 dagger

Total number of items: 48
```

5.6 الخلاصة

يمكن أن تحتوي القوائم والقواميس على عدّة قيم، بما فيها قوائم وقواميس أخرى. القواميس مفيدة لأنك تستطيع ربط مفتاح معين مع قيمة، على عكس القوائم التي هي سلسلة من القيم المرتبة.

يمكن الوصول إلى القيم داخل القاموس عبر استخدام الأقواس المربعة كما في القوائم، لكن بدلًا من استخدام فهرس رقمي فيمكن أن تكون المفاتيح في القواميس من مختلف القيم سواءً كانت أعدادًا صحيحةً أو أعدادًا عشريةً أو سلاسل نصية أو صفوف tuples.

يمكنك تمثيل الكائنات الحقيقية بتنظيم قيم البرنامج في بنى المعطيات، ورأينا مثال ذلك عمليًا على لعبة إكس-أو X-O.

خمسات

لبيع وشراء الخدمات المصغرة

أكبر سوق عربي لبيع وشراء الخدمات المصغرة
اعرض خدماتك أو احصل على ما تريد بأسعار تبدأ من 5$ فقط

تصفح الخدمات

6. معالجة النصوص باستخدام لغة بايثون

النصوص هي أكثر أنواع البيانات التي ستتعامل معها في برنامجك. لقد تعلمت كيفية جمع سلسلتين نصيتين مع بعضها عبر العامل +، لكنك تستطيع أكثر من ذلك بكثير؛ إذ تستطيع استخراج سلاسل نصية فرعية، وإضافة وحذف الفراغات، وتحويل الأحرف إلى أحرف كبيرة أو صغيرة، والتحقق من تنسيق السلاسل تنسيقًا صحيحًا؛ وتستطيع أيضًا كتابة برنامج بايثون للوصول إلى الحافظة في حاسوبك لنسخ ولصق النصوص.

كل ما سبق ستتعلمه في هذا الفصل وسنزيد على ذلك؛ وسنرى مشروعين عمليين: حافظة بسيطة التي تخزن عدة سلاسل نصية من النصوص، وبرنامج لأتمتة العملية المملة لتنسيق النصوص يدويًا.

6.1 التعامل مع السلاسل النصية

لنلقِ الآن نظرةً على الطرائق التي تسمح لنا فيها بايثون بالكتابة والطباعة والوصول إلى السلاسل النصية في برامجنا.

6.1.1 السلاسل النصية المجردة

كتابة القيم النصية في شيفرة بايثون هو أمر بالغ السهولة: نبدأ وننتهي بعلامة اقتباس مفردة.

لكن ما الذي قد يحدث في حال أردنا استخدام علامة اقتباس داخل السلسلة النصية؟

إذا كتبنا `'That is Alice's cat.'` فسيحدث خطأ لأن بايثون ستظن أن السلسلة النصية قد انتهت بعد `Alice` وبقية السطر `s cat.'` هي شيفرة غير صالحة في بايثون.

لحسن الحظ هنالك عدة طرائق لكتابة السلاسل النصية.

ا. علامات الاقتباس المزدوجة

قد تبدأ السلاسل النصية وتنتهي بكتابة علامتي اقتباس مزدوجتين، وهي تعمل كما في علامات الاقتباس المفردة. إحدى فوائد استخدام علامات الاقتباس المزدوجة هي إمكانية كتابة علامة الاقتباس الفردية داخلها:

```
>>> spam = "That is Alice's cat."
```

ولأن السلسلة النصية تبدأ بعلامة اقتباس مزدوجة، فستعلم بايثون أن علامة الاقتباس المفردة هي جزء من السلسلة النصية ولا تمثل نهايتها. لكن ماذا لو احتجنا إلى استخدام علامات الاقتباس المفردة والمزدوجة معًا في السلسلة النصية؟ سنحتاج هنا إلى محارف التهريب.

ب. محارف التهريب

تسمح لنا محارف التهريب escape characters باستخدام محارف ليس من الممكن إدراجها مباشرةً في السلسلة النصية.

يتألف محرف التهريب من خط مائل خلفي backslash `\` متبوعًا بأحد المحارف التي نريد إضافتها إلى السلسلة النصية؛ وصحيحٌ أن اسمه هو «محرف» التهريب، لكن يتألف من محرفين ويشار إليه عادةً بصيغة المفرد escape character.

فمثلًا محرف التهريب لعلامة الاقتباس المفردة هو `'` ويمكنك استخدامه داخل السلاسل النصية التي تبدأ وتنتهي بعلامة اقتباس مفردة، لنجرب الشيفرة الآتية:

```
>>> spam = 'Say hi to Bob\'s brother.'
```

ولأننا وضعنا خط مائل خلفي قبل علامة الاقتباس المفردة في `Bob\'s` فستعلم بايثون أننا لا نقصد أن ننهي السلسلة النصية. تسمح لنا محارف التهريب `\'` و `\"` بوضع علامات الاقتباس داخل السلاسل النصية المحاطة بعلامات اقتباس مفردة وأخرى مزدوجة على الترتيب. الجدول الآتي يوضح بعض محارف التهريب التي تستطيع استخدامها.

محارف التهريب	كيف يطبع
علامة اقتباس مفردة	`\'`
علامة اقتباس مزدوجة	`"`
علامة الجدولة tab	`\t`
سطر جديد newline	`\n`
خط مائل خلفي	`\\`

الجدول 9: محارف التهريب

جرب السلسلة النصية الآتية:

```
>>> print("Hello there!\nHow are you?\nI\'m doing fine.")
Hello there!
How are you?
I'm doing fine.
```

ج. السلاسل النصية الخام

يمكنك وضع الحرف `r` قبل علامة الاقتباس في بداية السلسلة النصية لجعلها سلسلة نصية خام. السلسلة النصية الخام raw string تتجاهل جميع محارف التهريب وتظهر جميع الخطوط المائلة الخلفية كما هي:

```
>>> print(r'That is Carol\'s cat.')
That is Carol\'s cat.
```

ولأنها سلسلة نصية خام فستعدّ بايثون الخط الخلفي المائل على أنه جزء من السلسلة النصية وليس جزءًا من محرف التهريب. يمكن أن تكون السلاسل النصية الخام مفيدةً إن كانت تكتب سلاسل نصية فيها عدد من الخطوط المائلة الخلفية، مثل مسارات الملفات في نظام ويندوز `r'C:\Users\Al\Desktop'` أو التعابير النمطية regular expressions المشروحة في الفصل القادم.

د. السلاسل النصية متعددة الأسطر بعلامات اقتباس ثلاثية

صحيحٌ أن بإمكاننا استخدام محرف التهريب `\n` لطابعة سطر جديد داخل سلسلة نصية، لكن من الأسهل استخدام السلاسل النصية متعددة الأسطر. تبدأ السلسلة النصية متعددة الأسطر في بايثون بثلاث علامات اقتباس مفردة أو مزدوجة، وستعد جميع علامات الاقتباس ومسافات الجدولة tabs والأسطر الجديدة على أنها جزءٌ من السلسلة النصية المحاطة بعلامات اقتباس ثلاثية.

لاحظ أن قواعد تنسيق بايثون الخاصة بالمسافات البادئة في الكتل البرمجية لا تنطبق على السلاسل النصية متعددة الأسطر.

```
print('''Dear Alice,


Eve's cat has been arrested for catnapping, cat burglary, and
extortion.


Sincerely,
Bob''')
```

احفظ ما سبق في ملف باسم `catnapping.py` وشغله:

```
Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and
extortion.

Sincerely,
Bob
```

لاحظ أن علامة الاقتباس المفردة في `Eve's` لا تحتاج إلى تهريب، فتهريب علامات الاقتباس المفردة والمزدوجة هو أمرٌ اختياري حين استخدام السلاسل النصية متعددة الأسطر.

السلسلة النصية الموجودة في دالة `print()` الآتية تكافئ المثال السابق دون استخدام السلاسل النصية متعددة الأسطر:

```
print('Dear Alice,\n\nEve\'s cat has been arrested for catnapping, cat
burglary, and extortion.\n\nSincerely,\nBob')
```

ه. التعليقات متعددة الأسطر

صحيحٌ أن رمز المربع `#` يرمز إلى بداية تعليق حتى نهاية السطر، لكن من الشائع استخدام سلسلة نصية متعددة الأسطر للتعليقات التي تمتد لأكثر من سطر. الشيفرات الآتية صالحة تمامًا في بايثون:

```
"""This is a test Python program.
Written by Al Sweigart al@inventwithpython.com

This program was designed for Python 3, not Python 2.
"""

def spam():
  """This is a multiline comment to help

  explain what the spam() function does."""

  print('Hello!')
```

6.1.2 فهرسة وتقسيم السلاسل النصية

تستعمل السلاسل النصية الفهارس ويمكن تقسيمها كما في القوائم lists. يمكنك أن تعدّ السلسلة النصية `'Hello, world!'` على أنها قائمة وكل حرف فيها يمثل عنصرًا في تلك القائمة مع فهرس مرتبط به.

```
'  H  e  l  l  o  ,     w  o  r  l  d  !  '
0  1  2  3  4  5  6  7  8  9  10 11 12
```

لاحظ تضمين الفراغ وإشارة التعجب، وسيكون عدد الأحرف هو 13، بـدءًا من الحـرف H في الفهـرس 0 إلى الحرف ! في الفهرس 12.

```
>>> spam = 'Hello, world!'
>>> spam[0]
'H'
>>> spam[4]
'o'
>>> spam[-1]
'!'
>>> spam[0:5]
'Hello'
>>> spam[:5]
'Hello'
>>> spam[7:]
'world!'
```

إذا حددت فهرسًا فستحصل على المحـرف الموجـود في ذاك الموضـع في السلسـلة النصـية، وإذا حـددت مجالًا من فهرسٍ ما إلى آخر فسيُضمَّن المحرف الموجود في فهرس البدايـة ولن يضـمن المحـرف الموجـود في فهـرس النهايـة، ولـذا إذا كـان لـدينا المتغـير spam الـذي فيـه 'Hello, world!' فـإن spam[0:5] هـو 'Hello'؛ فالسلسلة النصية الجزئية التي ستحصل عليها من spam[0:5] ستتضمن كل محـرف موجـود في المجال spam[0] حتى spam[4] ولن تحتوي الفاصلة الموجودة في الفهرس 5 ولا الفـراغ في المحـرف 6.هـذا السلوك يشبه سلوك الدالة range(5) التي ستؤدي -حين اسـتعمالها مـع for- إلى المـرور على الأرقـام حـتى الرقم 5 دون تضمينه.

لاحظ أن تقسيم السلاسل النصية لا يغير السلسلة النصية الأصـلية، وأنـه يمكننـا حفـظ القيمـة الناتجـة في متغير منفصل:

```
>>> spam = 'Hello, world!'
>>> fizz = spam[0:5]
>>> fizz
'Hello'
```

بتقسيم السلسلة النصية ثم تخزينها في متغير آخر فسنتمكن من الوصول إلى السلسلة النصية الأصلية والسلسلة النصية المقتطعة بسهولة.

ا. العوامل in و not in مع السلاسل النصية

يمكن استخدام العاملين `in` و `not in` مع السلاسل النصية كما في القوائم lists. التعبير البرمجي الذي فيه سلسلتين نصيتين مجموعٌ بينها بالعامل `in` أو `not in` سينتج القيمة المنطقية `True` أو `False`:

```
>>> 'Hello' in 'Hello, World'
True
>>> 'Hello' in 'Hello'
True
>>> 'HELLO' in 'Hello, World'
False
>>> '' in 'spam'
True
>>> 'cats' not in 'cats and dogs'
False
```

تختبر التعابير البرمجية السابقة إن كانت السلسلة النصية الأولى (كما هي بحذافيرها، مع حالة الأحرف فيها) موجودةً في السلسلة النصية الثانية.

6.1.3 وضع السلاسل النصية داخل سلاسل نصية أخرى

من الشائع في البرمجة وضع سلسلة نصية داخل سلسلة نصية أخرى، واستعملنا حتى الآن العامل `+` لجمع السلاسل النصية كما يلي:

```
>>> name = 'Al'
>>> age = 4000
>>> 'Hello, my name is ' + name + '. I am ' + str(age) + ' years old.'
'Hello, my name is Al. I am 4000 years old.'
```

لكن ذلك يتطلب كتابةً كثيرة، ومن الأسهل أن ندس السلاسل النصية داخل بعضها string interpolation، وبهذه الطريقة نستعمل العامل `%s` داخل السلاسل النصية كمؤشر لكي يستبدل مسبقًا إلى القيم التي تلي السلسلة النصية.

إحدى ميزات استخدام هذه الطريقة هي الحاجة إلى استدعاء الدالة `str()` لتحويل القيم إلى سلاسل نصية:

```
>>> name = 'Al'
>>> age = 4000
>>> 'My name is %s. I am %s years old.' % (name, age)
'My name is Al. I am 4000 years old.'
```

أضافت بايثون 3.6 ما يسمى بالسلاسل النصية المنسقة f-strings، وهي تشبه دس السلاسل النصـية لكنها تتيح إضافة التعابير البرمجية بين قوسين مجعدين مباشرةً؛ تذكر أن تضيف الحرف f قبل علامة الاقتبـاس الابتدائية في السلسلة النصية المنسقة:

```
>>> name = 'Al'
>>> age = 4000
>>> f'My name is {name}. Next year I will be {age + 1}.'
'My name is Al. Next year I will be 4001.'
```

إذا لم تتذكر تضمين الحرف f قبل علامة الاقتباس فستعامل الأقواس على أنها جزء من السلسلة النصية:

```
>>> 'My name is {name}. Next year I will be {age + 1}.'
'My name is {name}. Next year I will be {age + 1}.'
```

6.1.4 توابع مفيدة للتعامل مع السلاسل النصية

يشرح هذا القسم أكثر التوابع شيوعًا التي تحلل السلاسل النصية وتعالجها وتنتج سلاسل نصية معدلة.

I. التوابع upper() و lower() و isupper() و islower()

يعيد التابعان upper() و lower() سلسلةً نصيةً جديدةً تحوَّل فيها أحرف السلسلة النصـية الأصـلية إلى حالة الأحرف الكبيرة أو الصغيرة على التوالي. أما المحارف غير النصية أو غير اللاتينية فتبقى كما هي:

```
>>> spam = 'Hello, world!'
>>> spam = spam.upper()
>>> spam
'HELLO, WORLD!'
>>> spam = spam.lower()
>>> spam
'hello, world!'
```

لاحظ أن هذان التابعان لا يغيران السلسلة النصية الأصـلية وإنمـا يعيـدان سلسـلةً نصـيةً جديـدةً. إذا أردت تعديل السلسلة النصية الأصلية فعليك استدعاء التابع upper() أو lower() على السلسلة النصـية ثم إسـناد الناتج مباشرةً إلى المتغير الذي يحتوي على السلسلة النصـية الأصـلية؛ أي أننـا سـنكتب شـيئًا يشـبه spam =

`spam.upper()` لتعديل السلسلة النصية المخزنة في المتغير `spam` بـدلًا من كتابـة `spam.upper()` فقـط، وهذا يشبه حالة وجود متغير اسمه `eggs` يحتوي القيمة `10`، فكتابة التعبير الـبرمجي `eggs + 3` لا يـؤدي إلى تغيير القيمة المخزنة في المتغير `eggs` مباشرة، وإنما علينا كتابة `eggs = eggs + 3`.

التابعان `upper()` و `lower()` مفيدان إن أردنـا مقارنـة سلسـلتين نصـيتين مقارنـةً غـير حساسـةٍ لحالـة الأحـرف. فمثلًا السلسـلتان النصـيتان `'great'` و `'GREat'` غـير متسـاويتين؛ لكن قـد لا يهمنـا في بعض الحالات التي نطلب فيها مدخلات المستخدم إن كان قد كتب `Great` أم `GREAT` أم `gREAt`. انظـر المثـال الآتي الذي نحول فيه السلسلة النصية إلى حالة الأحرف الصغيرة:

```
print('How are you?')
feeling = input()
if feeling.lower() == 'great':
  print('I feel great too.')
else:
  print('I hope the rest of your day is good.')
```

حينما تشغل البرنـامج السـابق فسـيظهر لـك سـؤال، وإذا أدخلت الكلمـة `great` بـأي شـكل من الأشـكال فستظهر لك العبارة `'I feel great too.'`. من المفيد جدًا إضافة آلية للتعامل مع اختلاف حالات الأحرف في مدخلات المستخدم في برامجك، إذ سيسهل ذلك استخدامها كثيرًا.

```
How are you?
GREat
I feel great too.
```

يعيد التابعان `isupper()` و `islower()` قيمةً منطقيـة `True` إن كـانت السلسـلة النصـية تحتـوي على حرف واحد على الأقل وكان ذاك الحرف كبيرًا أو صغيرًا على التوالي وبالترتيب؛ وإلا فستعيد القيمة `False`:

```
>>> spam = 'Hello, world!'
>>> spam.islower()
False
>>> spam.isupper()
False
>>> 'HELLO'.isupper()
True
>>> 'abc12345'.islower()
True
```

```
>>> '12345'.islower()
False
>>> '12345'.isupper()
False
```

ولأن القيمة المعادة من التابعين `upper()` و `lower()` هي سلاسل نصية، فيمكننا استدعاء توابع التعامل مع السلاسل النصية على القيم المعادة من تلك التوابع مباشرةً، وتبدو هذه التعابير البرمجية على أنها سلسلة من التوابع وراء بعضها كما في المثال الآتي:

```
>>> 'Hello'.upper()
'HELLO'
>>> 'Hello'.upper().lower()
'hello'
>>> 'Hello'.upper().lower().upper()
'HELLO'
>>> 'HELLO'.lower()
'hello'
>>> 'HELLO'.lower().islower()
True
```

ب. مجموعة توابع isX()

بالإضافة إلى التابعين `islower()` و `isupper()` هنالك عدد من التوابع التي يبدأ اسمها بالكلمة `is`، وتعيد تلك التوابع قيمةً منطقية التي تشرح طبيعة السلسلة النصية.

هذه بعض تلك التوابع:

- `isalpha()` يعيد `True` إذا احتوت السلسلة النصية على أحرف عادية فقط ولم تكن فارغة.
- `isalnum()` يعيد `True` إذا احتوت السلسلة النصية على أحرف عادية وأرقام فقط ولم تكن فارغة.
- `isdecimal()` يعيد `True` إذا احتوت السلسلة النصية على أرقام فقط ولم تكن فارغة.
- `isspace()` يعيد `True` إذا احتوت السلسلة النصية على فراغات عادية ومسافات جدولة tabs وأسطر جديدة newlines فقط ولم تكن فارغة.
- `istitle()` يعيد `True` إذا بدأت السلسلة النصية بحرف كبير متبوعٌ بمجموعة من الأحرف الصغيرة.

لنجرب تلك التوابع عمليًا:

```
>>> 'hello'.isalpha()
True
>>> 'hello123'.isalpha()
False
>>> 'hello123'.isalnum()
True
>>> 'hello'.isalnum()
True
>>> '123'.isdecimal()
True
>>> '    '.isspace()
True
>>> 'This Is Title Case'.istitle()
True
>>> 'This Is Title Case 123'.istitle()
True
>>> 'This Is not Title Case'.istitle()
False
>>> 'This Is NOT Title Case Either'.istitle()
False
```

تفيد توابع السلاسل النصية `isX()` حينما نريد التحقق من مدخلات المستخدم، فالبرنامج الآتي يطلب من المستخدم إدخال عمره وكلمة مرور صالحة:

```
while True:
  print('Enter your age:')
  age = input()
  if age.isdecimal():
    break
  print('Please enter a number for your age.')

while True:
  print('Select a new password (letters and numbers only):')
  password = input()
```

```
    if password.isalnum():
      break
    print('Passwords can only have letters and numbers.')
```

طلبنا من المستخدم في أول حلقة `while` أن يدخل عمره، ونخزنه في المتغير `age`. إذا كانت قيمة `age` هي قيمة عددية صحيحة، فسنخرج من أول حلقة `while` وندخل في الثانية التي تسأله عن كلمة المرور. وإلا فسنسأل المستخدم عن عمره مجددًا.

في حلقة `while` الثانية طلبنا كلمة المرور وخزناها في المتغير `password`، وسنخرج من الحلقة إن كانت كلمة المرور تتألف من أرقام أو أحرف، وإلا فسنطلب من المستخدم إدخال كلمة مرور صالحة مجددًا.

```
Enter your age:
forty two
Please enter a number for your age.
Enter your age:
42
Select a new password (letters and numbers only):
secr3t!
Passwords can only have letters and numbers.
Select a new password (letters and numbers only):
secr3t
```

تمكّنا من التحقق من صلاحية مدخلات المستخدم في المثال السابق باستخدام التابعين `isdecimal()` و`isalnum()`، ولم نقبل كتابة `forty two` بل قبلنا `42`، ولم نقبل `secr3t!` بل قبلنا `secr3t`.

6.1.5 التابعان startswith() و endswith()

يعيد التابعان `startswith()` و `endswith()` القيمة `True` إن بدأت أو انتهت السلسلة النصية التي استدعت عليها (على التوالي) بالسلسلة النصية الممررة إلى التابع؛ وإلا فستعيد `False`:

```
>>> 'Hello, world!'.startswith('Hello')
True
>>> 'Hello, world!'.endswith('world!')
True
>>> 'abc123'.startswith('abcdef')
False
>>> 'abc123'.endswith('12')
```

```
False
>>> 'Hello, world!'.startswith('Hello, world!')
True
>>> 'Hello, world!'.endswith('Hello, world!')
True
```

هذه التوابع هي بديل مفيد لعامل المساواة == إذا أردنا التحقق من الجزء الأول أو الأخير من السلسلة النصية فقط، بدلًا من مقارنتها كلها.

6.1.6 التابعان join() و split()

يفيد التابع `join()` حينما يكون لدينا قائمة فيها سلاسل نصية ونريد أن نجمعها كلها مع بعضها بعضًا في سلسلة نصية واحدة؛ ويستدعى التابع `join()` على سلسلة نصية، ويقبل معاملًا هو قائمة list فيها سلاسل نصية، ويعيد سلسلةً نصيةً تساوي دمج السلاسل النصية كلها:

```
>>> ', '.join(['cats', 'rats', 'bats'])
'cats, rats, bats'
>>> ' '.join(['My', 'name', 'is', 'Simon'])
'My name is Simon'
>>> 'ABC'.join(['My', 'name', 'is', 'Simon'])
'MyABCnameABCisABCSimon'
```

لاحظ أن السلسلة النصية التي استدعينا عليها التابع `join()` أصبحت موجودة بين كل سلسلتين نصيتين في القائمة الممررة كوسيط. فمثلًا حين استدعاء `join(['cats', 'rats', 'bats'])` على السلسلة النصية `', '` فيكون الناتج هو `'cats, rats, bats'`.

تذكر أن التابع `join()` يستدعى على سلسلة نصية ونمرر إليه قائمة، وليس العكس.

يفعل التابع `split()` عكس فعل التابع `join()` تمامًا: يستدعى على سلسلة نصية وتعيد قائمةً من السلاسل النصية:

```
>>> 'My name is Simon'.split()
['My', 'name', 'is', 'Simon']
```

ستُقسَم السلسلة النصية `'My name is Simon'` افتراضيًا عند كل محرف يمثل فراغًا (سواءً كان فراغًا عاديًا `' '` أو محرف جدولة tab أو سطرًا جديدًا newline)، ولن تضمن الفراغات في السلاسل النصية الموجودة في القائمة المعادة من استدعاء هذا التابع.

يمكننا تمرير محرف الفصل إلى التابع `split()` لتحديد محرف آخر غير محارف الفراغات:

```
>>> 'MyABCnameABCisABCSimon'.split('ABC')
['My', 'name', 'is', 'Simon']
>>> 'My name is Simon'.split('m')
['My na', 'e is Si', 'on']
```

من الشائع استدعاء التابع `split()` من أجل تقسيم سلسلة نصية متعددة الأسطر في مكان وقوع محـرف السطر الجديد:

```
>>> spam = '''Dear Alice,
How have you been? I am fine.
There is a container in the fridge
that is labeled "Milk Experiment."

Please do not drink it.
Sincerely,
Bob'''
>>> spam.split('\n')
['Dear Alice,', 'How have you been? I am fine.', 'There is a container
in the
fridge', 'that is labeled "Milk Experiment."', '', 'Please do not
drink it.',
'Sincerely,', 'Bob']
```

يسمح لنا استدعاء التابع `split()` مع تمرير محرف السطر الجديد `'\n'` إلى تقسيم سلسلة نصية متعددة الأسطر إلى قائمة يمثل فيها كل عنصر سطرًا من الأسطر.

6.1.7 تقسيم السلاسل النصية باستخدام التابع partition()

يمكن أن يقسم التابع `partition()` سلسلةً نصية إلى أقسام، ويعمل بتمرير سلسلة نصية إليه كفاصـل، التي سيبحث عنها في السلسلة النصية التي استدعي عليهـا، ويعيـد صـفًا tuple فيـه السلسـلة النصـية الـتي تسبق الفاصل، والفاصل، والسلسلة النصية التي تلي الفاصل:

```
>>> 'Hello, world!'.partition('w')
('Hello, ', 'w', 'orld!')
>>> 'Hello, world!'.partition('world')
('Hello, ', 'world', '!')
```

إذا احتـوت السلسـلة النصـية الـتي تسـتدعي التـابع `partition()` عليهـا على أكـثر من تكـرار للفاصـل، فستقسم السلسلة النصية عند أول وقوع له فقط:

```
>>> 'Hello, world!'.partition('o')
('Hell', 'o', ', world!')
```

وإن لم يعثر على الفاصل في السلسلة النصية، فستعاد السلسلة النصية التي استدعي عليها التابع كأول عنصر في الصف، وستكون السلسلتان النصيتان الباقيتان فارغتين:

```
>>> 'Hello, world!'.partition('XYZ')
('Hello, world!', '', '')
```

يمكننا استخدام نشر المتغيرات بالإسناد الجماعي لإسناد السلاسل النصية المعادة إلى ثلاثة متغيرات:

```
>>> before, sep, after = 'Hello, world!'.partition(' ')
>>> before
'Hello,'
>>> after
'world!'
```

يفيد التابع `partition()` إن كنت تحتاج إلى الحصول على السلسلة النصية التي تسبق فاصلًا محددًا، والفاصل نفسه، والسلسلة النصية التي تلي ذاك الفاصل.

6.1.8 محاذاة النصوص عبر rjust() و ljust() و center()

يعيد التابعان `rjust()` و `ljust()` سلسلةً نصية محاطة بفراغات افتراضيًا. يمثل أول وسيط يمرر إلى تلك التوابع قيمةً لعدد الفراغات المحيطة بالسلسلة النصية:

```
>>> 'Hello'.rjust(10)
'     Hello'
>>> 'Hello'.rjust(20)
'               Hello'
>>> 'Hello, World'.rjust(20)
'         Hello, World'
>>> 'Hello'.ljust(10)
'Hello     '
```

التابع `'Hello'.rjust(10)` يعني أننا نريد محاذاة السلسلة النصية `'Hello'` إلى اليمين ويكون طول السلسلة النصية الكاملة هو 10. ولما كانت `'Hello'` هي 5 محارف، فستضاف 5 فراغات على يسارها، مما يؤدي إلى إعادة سلسلة نصية طولها 10 محارف وتكون فيها محاذاة `'Hello'` على اليمين.

وسيطٌ اختياري للتابعين `rjust()` و `ljust()` يحدد محرفًا للملء بخلاف الفراغ:

```
>>> 'Hello'.rjust(20, '*')
'***************Hello'
>>> 'Hello'.ljust(20, '-')
'Hello---------------'
```

التابع `center()` يعمل مثـل التـابعين `ljust()` و `rjust()` لكنـه يوسِّـط النص بـدلًا من محاذاتـه إلى اليسار أو اليمين:

```
>>> 'Hello'.center(20)
'       Hello        '
>>> 'Hello'.center(20, '=')
'=======Hello========'
```

تفيد هذه التوابع حينمـا نريـد طباعـة جـداول من البيانـات ويكـون لهـا تباعـد صـحيح. اكتب البرنـامج الآتي واحفظه في الملف `picnicTable.py`:

```
def printPicnic(itemsDict, leftWidth, rightWidth):
  print('PICNIC ITEMS'.center(leftWidth + rightWidth, '-'))
  for k, v in itemsDict.items():
    print(k.ljust(leftWidth, '.') + str(v).rjust(rightWidth))


picnicItems = {'sandwiches': 4, 'apples': 12, 'cups': 4, 'cookies':
8000}
printPicnic(picnicItems, 12, 5)
printPicnic(picnicItems, 20, 6)
```

عرفنا الدالة `printPicnic()` التي تأخذ قاموسًا كمعامل لها، واستخدمنا التوابع `center()` و`ljust()` و`rjust()` لطباعة المعلومات بصيغة جميلة تشبه الجدول.

القـاموس الـذي سـنمرره إلى الدالـة `printPicnic()` هـو `picnicItems`، ولـدينا في القـاموس `picnicItems` 4 صندويشات و 12 تفاحة و 4 كاسات و 8,000 كعكة (نعم ثمانية آلاف!). نرغب بتنظيم هـذه المعلومات في عمودين، ونضع اسم العنصر على اليسار والكمية على اليمين.

لفعل ذلك نحتـاج إلى أن نقـرر كم سـيكون عـرض العمـودين الأيسـر والأيمن؛ لـذا نحتـاج إلى تمريـر هـاتين القيمتين مع القاموس إلى الدالة `printPicnic()` عـبر المعـاملين `leftWidth` لعـرض العمـود الأيسـر من الجدول و `rightWidth` لعرض العمود الأيمن من الجدول.

ستطبع الدالة `printPicnic()` عنوانًا متوسطًا فوق العنصر PICNIC ITEMS، ثم ستمر على عناصر القاموس وتطبع كل زوج من القيم في سطر وتحاذي المفتاح على اليسار وله حاشية متألفة من فواصل، والقيمة على اليمين ولها حاشية متألف من فراغات.

بعد تعريف الدالة `printPicnic()` سنعرف القاموس `picnicItems` ونستدعي الدالة `printPicnic()` مرتين، مع تمرير عرض مختلف للعمودين الأيسر والأيمن؛ يكون فيها عرض العمود الأيسر 12 والأيمن 5 في المرة الأولى، ثم 20 و 6 في المرة الثانية.

```
---PICNIC ITEMS--
sandwiches..    4
apples......   12
cups........    4
cookies..... 8000
-------PICNIC ITEMS-------
sandwiches.....     4
apples.............    12
cups...............     4
cookies.............  8000
```

رأينا كيف استفدنا من `rjust()` و `ljust()` و `center()` لطباعة سلاسل نصية منسقة تنسيقًا جميلًا، حتى لو لم نكن نعرف كم سيكون العدد الدقيق لمحارف السلسلة النصية التي سنطبعها.

6.1.9 حذف الفراغات عبر strip() و rstrip() و lstrip()

نحتاج أحيانًا إلى حذف الفراغات البيضاء (التي هي محارف الفراغ العادي ومسافة الجدولة tab والسطر الجديد newline) من الطرف الأيسر أو الأيمن أو من كلا طرفي السلسلة النصية.

سيعيد التابع `strip()` سلسلةً نصيةً جديدةً لا تحتوي على أي فراغات في بدايتها أو نهايتها، بينما يحذف التابعان `lstrip()` و `rstrip()` الفراغات البيضان من الطرف الأيسر أو الأيمن على التوالي:

```
>>> spam = '    Hello, World    '
>>> spam.strip()
'Hello, World'
>>> spam.lstrip()
'Hello, World    '
>>> spam.rstrip()
'    Hello, World'
```

يمكننا تمرير وسيط اختياري يحتوي على المحارف التي نريد حذفها بدلًا من الفراغات:

```
>>> spam = 'SpamSpamOliveSpamEggsSpamSpam'
>>> spam.strip('ampS')
'OliveSpamEggs'
```

تمرير الوسيط `'ampS'` إلى التابع `strip()` سيؤدي إلى حذف جميع تكرارات المحـارف `a` و `m` و `p` و `S` من بداية ونهاية السلسلة النصية `spam`. لاحظ أن ترتيب الأحرف في السلسلة النصية الممررة إلى التـابع `strip()` لا يهم، فكتابة `strip('ampS')` تكافئ كتابة `strip('mapS')` أو `strip('Spam')`.

6.1.10 القيم العددية للأحرف مع الدالتين ord() و chr()

تخزن الحواسيب البيانات على هيئة بايتات، وهي سلاسل من البتات بنظام العد الثنـائي، وهـذا يعـني أننـا نحتاج إلى تحويل النصوص إلى أرقام لكي نستطيع تخزينها؛ ولهذا يكـون لكـل محـرف character قيمـة رقميـة مرتبطة به تسمى Unicode code point. فمثلًا القيمة الرقمية `65` تمثل `'A'`، والقيمة `52` تمثل `'4'`، والقيمة `33` تمثل `'!'`.

يمكننا استخدام الدالة `ord()` للحصول على القيمة الرقمية لسلسـلة نصـية تحتـوي محرفًـا واحـدًا، والدالـة `chr()` للحصول على المحرف الذي وفرنا قيمته الرقمية:

```
>>> ord('A')
65
>>> ord('4')
52
>>> ord('!')
33
>>> chr(65)
'A'
```

تفيد هذه الدوال حينما نحتاج إلى ترتيب المحارف أو إجراء عمليات رياضية عليها:

```
>>> ord('B')
66
>>> ord('A') < ord('B')
True
>>> chr(ord('A'))
'A'
>>> chr(ord('A') + 1)
'B'
```

هنالك تفاصيل كثيرة عن يونيكود وأرقام المحارف، لكنها خارجة عن سياق هذا الكتاب.

6.2 نسخ ولصق السلاسل النصية باستخدام الوحدة pyperclip

تمتلك الوحدة `pyperclip` الدالتين `copy()` و `paste()` اللتي يمكنها إرسال واستقبال النص من حافظة نظام التشغيل الذي تستعمله. إذ يسهل عليك إرسال ناتج برنامجك إلى الحافظة أن تلصقه في رسالةٍ بريدية أو في محرر النصوص أو غيره من البرمجيات.

6.2.1 تشغيل سكربتات بايثون خارج محرر Mu

شغلنا كل سكربتات بايثون التي كتبناها حتى الآن باستخدام الصدفة التفاعلية ومحرر الشيفرات Mu؛ لكننا لسنا بحاجة إلى فتح محرر Mu في كل مرة نريد فيها تنفيذ سكربتات بايثون التي كتبناها. لحسن الحظ هنالك طرائق تسهل علينا تشغيل سكربتات بايثون، لكن هذه الطرائق تختلف من نظام ويندوز إلى MacOS ولينكس، وهي مشروحة في الفصل الأول من هذا الكتاب، لذا أنصحك بالانتقال إلى الفصل الأول لتعرف كيف تشغل سكربتات بايثون بسهولة على نظامك، وكيف تمرر خيارات سطر الأوامر command line arguments إليها، لاحظ أنك لا تستطيع تمرير خيارات سطر الأوامر باستخدام محرر Mu.

الوحدة `pyperclip` غير مضمنة في بايثون، لذا عليك تثبيتها باتباع التعليمات المذكورة في الفصل الأول من الكتاب أيضًا. يمكنك أن تدخل ما يلي في الصدفية التفاعلية بعد تثبيت الوحدة `pyperclip`:

```
>>> import pyperclip
>>> pyperclip.copy('Hello, world!')
>>> pyperclip.paste()
'Hello, world!'
```

إذا غيّر أحد البرامج محتويات الحافظة فستعيدها الدالة `paste()`، فانسخ مثلًا عبارةً من المتصفح أو من محرر الشيفرات ثم استدعِ الدالة `paste()`:

```
>>> pyperclip.paste()
'For example, if I copied this sentence to the clipboard and then
called
paste(), it would look like this:'
```

6.3 مشروع: حافظة فيها رسائل تلقائية

إذا سبق وأن رددت على عدد كبير من رسائل البريد الإلكتروني بعبارات متشابهة فمن المرجح أنك قد قضيت وقتًا طويلًا في الكتابة على الحاسوب، ومن المرجح أن لديك ملف نصي فيه تلك العبارات ليسهل عليك

نسخها ولصقها من الحافظة؛ لكن يمكن لحافظة نظام تشغيلك الاحتفاظ برسالة واحدة فقط في آن واحد وهذا ليس مريحًا في العمل. لنحاول سويةً تسهيل هذه المهمة بكتابة برنامج يخزن عددًا من العبارات.

6.3.1 الخطوة 1: تصميم البرنامج وبنى المعطيات

نرغب أن نشغل هذا البرنامج من سطر الأوامر مع تمرير وسيط إليه الذي يحتوي على كلمة مفتاحية واحدة مثل «موافق agree» أو «مشغول busy». وستُنسَخ الرسالة المرتبطة بتلك الكلمة المفتاحية إلى الحافظة لكي يستطيع المستخدم لصقها مباشرةً في الردود، وبهذا يمكن أن نستعمل عبارات طويلة دون الحاجة إلى إعادة كتابتها كل مرة.

ا. مشاريع الفصول

هذا هو أول «مشروع» في هذا الكتاب ومن الآن فصاعدًا ستجد في كل فصل عددًا من المشاريع التي تستعمل المفاهيم المشروحة في ذاك الفصل وتبدأ هذه المشاريع من الصفر حتى الحصول على برنامج يعمل تمامًا، وأنصحك وبشدة أن تطبق أولًا بأول ولا تكتفي بالقراءة فقط.

افتح نافذة محرر جديدة واحفظ البرنامج الآتي باسم `mclip.py`، ستحتاج إلى بدء البرنامج مع الرمز `#!` (الذي يسمى shebang، راجع الفصل الأول من هذا الكتاب)، ثم كتابة تعليق يشرح باختصار ما يفعله البرنامج.

لمّا كنّا نريد أن نربط كل جملة نصية بمفتاح، فمن المنطقي أن نخزنها في قاموس، والذي سيكون هو بنية المعطيات الأساسية في برنامجنا:

```
#! python3
# mclip.py - برنامج متعدد الحافظة

TEXT = {'agree': """Yes, I agree. That sounds fine to me.""",
   'busy': """Sorry, can we do this later this week or next week?""",
   'upsell': """Would you consider making this a monthly donation?"""}
```

6.3.2 الخطوة 2: التعامل مع وسائط سطر الأوامر

تخزن الوسائط الممررة من سطر الأوامر command line arguments في المتغير `sys.argv` (راجع الفصل الأول من هذا الكتاب لمزيدٍ من المعلومات حول استخدام وسائط سطر الأوامر في نظامك).

يجب أن يكون أول عنصر في القائمة `sys.argv` هو سلسلة نصية تمثل اسم الملف `'mclip.py'` أما العنصر الثاني في القائمة فهو قيمة الوسيط الأول الممرر عبر سطر الأوامر.

سيمثل الوسيط الأول قيمة مفتاح العبارة التي نريد نسخها، ولأننا نريد إجبار المستخدم على تمرير وسيط في سطر الأوامر، فستعرض رسالةً إلى المستخدم إن نسي توفيره للبرنامج (إذا كانت القائمة `sys.argv` تحتوي على أقل من قيمتين):

```
#! python3
# mclip.py - برنامج متعدد الحافظة.

TEXT = {'agree': """Yes, I agree. That sounds fine to me.""",
   'busy': """Sorry, can we do this later this week or next week?""",
   'upsell': """Would you consider making this a monthly donation?"""}

import sys
if len(sys.argv) < 2:
  print('Usage: python mclip.py [keyphrase] - copy phrase text')
  sys.exit()

keyphrase = sys.argv[1]     # أول وسيط في سطر الأوامر هو مفتاح العبارة التي نريد نسخها
```

6.3.3 الخطوة 3: نسخ العبارة الصحيحة

أصبح لدينا مفتاح العبارة مخزنًا في المتغير `keyphrase`.

لذا ستحتاج إلى التحقق إن كان هذا المفتاح موجودًا في القاموس `TEXT`، فإن كان موجودًا فسننسخ العبارة المرتبطة بهذا المفتاح إلى الحافظة باستخدام الدالة `pyperclip.copy()`، ولمّا كنّا نستعمل الوحدة `pyperclip` فسنحتاج إلى استيرادها أيضًا.

لاحظ أننا لسنا بحاجة إلى إنشاء المتغير `keyphrase`، إذ نستطيع استخدام `sys.argv[1]` في أي مكان استعملنا فيه keyphrase لكن وجود متغير باسم `keyphrase` سيسهل قراءة الشيفرة كثيرًا:

```
#! python3
# mclip.py - برنامج متعدد الحافظة.

TEXT = {'agree': """Yes, I agree. That sounds fine to me.""",
   'busy': """Sorry, can we do this later this week or next week?""",
```

```
    'upsell': """Would you consider making this a monthly donation?"""}

import sys, pyperclip
if len(sys.argv) < 2:
  print('Usage: py mclip.py [keyphrase] - copy phrase text')
  sys.exit()

keyphrase = sys.argv[1]     # keyphrase الوسيط الأول لسطر الأوامر هو

if keyphrase in TEXT:
  pyperclip.copy(TEXT[keyphrase])
  print('Text for ' + keyphrase + ' copied to clipboard.')
else:
  print('There is no text for ' + keyphrase)
```

ستبحث الشيفرة في القاموس `TEXT` عن المفتاح، وإن وجدت فسنحصل على العبارة المرتبطة بذاك المفتاح، ثم ننسخها إلى الحافظة، ونطبع رسالة فيها العبارة المنسوخة، وإن لم نعثر على المفتاح فستظهر رسالة للمستخدم تخبره بعدم وجود هكذا مفتاح.

السكربت السابق كامل، ويمكننا اتباع التعليمات الموجودة في الفصل الأول لتشغيل البرامج السطرية بسهولة، وأصبح لدينا الآن طريقة سريعة لنسخ الرسائل الطويلة إلى الحافظة بسهولة. لا تنسَ أن تعدل قيمة القاموس `TEXT` في كل مرة تحتاج فيها إلى إضافة عبارة جديدة.

إذا كنت على نظام ويندوز فيمكنك إنشاء ملف دفعي batch file لتشغيل البرنامج باستخدام نافذة تشغيل البرامج Run (بالضغط على `Win+R`). أدخل ما يلي في محرر النصوص واحفظه باسم `mclip.bat` في مجلد `C:\Windows`:

```
@py.exe C:\path_to_file\mclip.py %*
@pause
```

يمكننا بعد إنشاء الملف الدفعي أن نشغل برنامجنا بالضغط على `Win+R` ثم كتابة mclip key_phrase.

6.4 مشروع: إضافة قائمة منقطة إلى ويكيبيديا

حينما تعدل فصلًا في ويكيبيديا، يمكنك إنشاء قائمة منقطة بوضع كل عنصر من عناصر القائمة في سطر خاص به، ووضع نجمة قبله؛ لكن لنقل أن لدينا قائمةً طويلةً جدًا من العناصر التي تريد تحويلها إلى قائمة منقطة. هنا يمكنك أن تقضي بعض الوقت بإضافة رمز النجمة في بداية كل سطر يدويًا أو تكتب سكربتًا يؤتمت هذه العملية.

يأخذ السكربت `bulletPointAdder.py` النص الموجود في الحافظة ويضيف نجمة وفراغًا إلى بداية كل سطر فيه، ثم يحفظ النص الناتج إلى الحافظة. فمثلًا إذا نسخت النص الآتي من صفحة ويكيبيديا المسماة «قائمة القوائم List of Lists of Lists» إلى الحافظة:

```
Lists of animals
Lists of aquarium life
Lists of biologists by author abbreviation
Lists of cultivars
```

ثم شغلت البرنامج `bulletPointAdder.py` فستصبح محتويات الحافظة كما يلي:

```
* Lists of animals
* Lists of aquarium life
* Lists of biologists by author abbreviation
* Lists of cultivars
```

ونستطيع الآن لصق النص الناتج إلى ويكيبيديا كقائمة منقطة.

6.4.1 الخطوة 1: النسخ واللصق من وإلى الحافظة

نريد من برنامج `bulletPointAdder.py` أن يفعل ما يلي:

1. يأخذ النص الموجود في الحافظة.
2. يجري عليه عمليات.
3. يحفظ الناتج في الحافظة.

الخطوة الثانية صعبة بعض الشيء، لكن الخطوات 1 و 3 سهلة جدًا: كل ما علينا فعله هو استدعاء الدالتين `pyperclip.copy()` و `pyperclip.paste()`.

لننشِئ برنامجًا ينفذ الخطوات 1 و 3:

```
#! python3
# bulletPointAdder.py - إضافة قائمة منقطة لويكيبيديا في البداية

# لكل سطر من النص في الحافظة
import pyperclip
text = pyperclip.paste()

# TODO: فصل الأسطر وإضافة رمز النجمة
pyperclip.copy(text)
```

التعليق الذي يبدأ بالكلمة TODO هو تذكير لنا أن علينا إكمال هذا الجزء من البرنـامج، لـذا سـتكون الخطـوة القادمة هي برمجة هذا الجزء.

6.4.2 الخطوة 2: فصل الأسطر وإضافة النجمة

ناتج استدعاء `pyperclip.paste()` يعيد النص الموجود في الحافظة كسلسة نصـية واحـدة، إذ سـيبدو مثال «قائمة القوائم» الذي نسخناه على الشكل التالي:

```
'Lists of animals\nLists of aquarium life\nLists of biologists by
author
abbreviation\nLists of cultivars'
```

يوجد المحرف `\n` في السلسلة النصية السابقة لعرضها على عدة أسطر حين لصقها من الحافظة، لاحـظ أن السلسلة النصية السابقة «متعددة» الأسطر لوجود محرف التهريب `\n`. ستحتاج إلى إضافة نجمة إلى بدايـة كـل سطر من الأسطر السابقة.

يمكنك أن تكتب شيفرة تبحث عن المحرف `\n` وتضيف نجمةً قبله، أو أن تستعمل التابع `split()` لإعادة قائمة من السلاسل النصية يمثل كل عنصر فيها سطرًا من السلسلة النصية الأصلية، وبعد ذلك تضـيف النجمـة قبل كل عنصر:

```
#! python3
# bulletPointAdder.py - إضافة قائمة منقطة لويكيبيديا في البداية

# لكل سطر من النص في الحافظة
import pyperclip
text = pyperclip.paste()
```

```
# فصل الأسطر وإضافة النجمة
lines = text.split('\n')
for i in range(len(lines)):     # المرور على جميع عناصر القائمة
  lines[i] = '* ' + lines[i] # إضافة نجمة وفراغ قبل كل عنصر

pyperclip.copy(text)
```

قسمنا النص في مكان السطر الجديد ليمثل كل عنصر في القائمة سطرًا واحدًا، وخزّنا الناتج في المتغير `lines`، ثم مررنا على عناصر القائمة `lines`، ولكل عنصر أضفنا نجمة وفراغًا في بداية السطر. أصبحت لدينا الآن قائمة باسم `lines` فيها عناصر القائمة وقبل كل عنصر رمزُ النجمة.

6.4.3 الخطوة 3: دمج الأسطر المعدلة

تحتوي القائمة `lines` على الأسطر المعدلة التي تبدأ بالنجمة، لكن الدالة `pyperclip.copy()` تتعامل مع السلاسل النصية وليس مع القوائم، لذا نحتاج إلى تحويل القائمة إلى سلسلة نصية، وذلك بجمع عناصر مع بعضها بعضًا عبر التابع `join()`:

```
#! python3
# bulletPointAdder.py - إضافة قائمة منقطة لويكيبيديا في البداية

# لكل سطر من النص في الحافظة
import pyperclip
text = pyperclip.paste()

# فصل الأسطر وإضافة النجمة
lines = text.split('\n')
for i in range(len(lines)):     # المرور على جميع عناصر القائمة
  lines[i] = '* ' + lines[i] # إضافة نجمة وفراغ قبل كل عنصر

text = '\n'.join(lines)
pyperclip.copy(text)
```

حين تشغيل البرنامج السابق فسيبدل محتويات الحافظة ويضيف نجمةً قبل كل سطر موجود فيها.

أصبح البرنامج جاهزًا للتجربة!

حتى لو لم تكن بحاجة إلى أتمتة هذه المهمة البسيطة (فهنالك محرر مرئي لويكيبيديا مثلًا) لكن قد تستفيد من برامج مشابهة لأتمتة عمليات بسيطة لمعالجة النصوص، مثل إزالة الفراغات، أو تحويل حالة الأحرف، أو أيًّا كان غرضك من تعديل محتويات الحافظة.

6.5 أسئلة للتدريب

1. ما هي محارف التهريب escape characters؟
2. ماذا تمثل المحارف `\n` و `\t`؟
3. كيف تطبع الخط المائل الخلفي `\`؟
4. السلسلة النصية `"Howl's Moving Castle"` صحيحة في بايثون، لماذا لم نهرِّب علامة الاقتباس المفردة في الكلمة `H owl's`؟
5. كيف تكتب سلسلة نصية متعددة الأسطر دون استعمال المحرف `\n`؟
6. ما هي نتيجة التعابير البرمجية الآتية؟

```
'Hello, world!'[1]

'Hello, world!'[0:5]

'Hello, world!'[:5]

'Hello, world!'[3:]
```

7. ما هي نتيجة التعابير البرمجية الآتية؟

```
'Hello'.upper()

'Hello'.upper().isupper()

'Hello'.upper().lower()
```

8. ما هي نتيجة التعابير البرمجية الآتية؟

```
'Remember, remember, the fifth of November.'.split()

'-'.join('There can be only one.'.split())
```

9. ما هي التوابع التي يمكنها أن تحاذي السلسلة النصية إلى اليمين وإلى اليسار وإلى المنتصف؟

10. كيف تحذف الفراغات البيضاء من بداية أو نهاية السلسلة النصية؟

6.6 مشاريع للتدريب

لكي تتدرب، اكتب برامج لتنفيذ المهام الآتية.

6.6.1 طابع جداول

اكتب دالةً اسمها `printTable()` تقبل قائمةً فيها قوائم تحتوي على سلاسل نصية، وتعرضها في جدول منسق تكون فيه محاذاة الأعمدة إلى اليمين.افترض أن لكل القوائم الداخلية العدد نفسه من السلاسل النصية.

```
tableData = [['apples', 'oranges', 'cherries', 'banana'],
     ['Alice', 'Bob', 'Carol', 'David'],
     ['dogs', 'cats', 'moose', 'goose']]
```

يجب أن تطبع الدالة `printTable()` ما يلي:

```
 apples Alice  dogs
oranges   Bob  cats
cherries Carol moose
 banana David goose
```

تلميح: يجب أن تبحث عن أطول سلسلة نصية في كل قائمة داخلية، كي يتسع العمود لجميع السلاسل النصية. يمكنك تخزين العرض الأقصى لكل عمود في قائمة من الأرقام.

يمكن أن تبدأ الدالة `printTable()` بالسطر `colWidths = [0] * len(tableData)` الذي سينشِئ قائمةً تحتوي على الرقم `0` يساوي عدد القوائم الداخلية الموجودة في القائمة `tableData`، وبالتالي ستخزن عرض أطول سلسلة نصية في `tableData[0]` في العنصر `colWidths[0]`، وعرض أطول سلسلة نصية في القائمة `tableData[1]` في العنصر `colWidths[1]` وهلم جرًا... ثم يمكنك الحصول على أكبر قيمة في القائمة `colWidths` لتعرف القيمة التي ستمررها كعرض إلى التابع `rjust()`.

6.7 الخلاصة

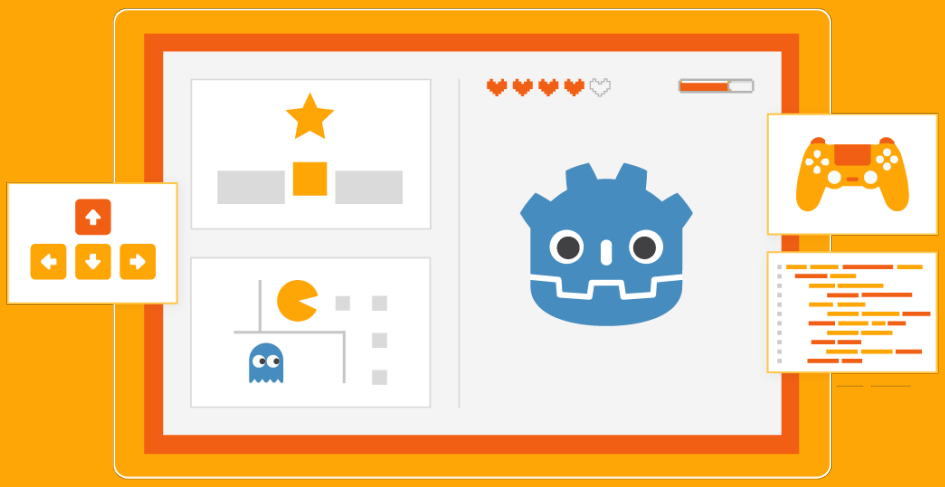
النصوص هي أكثر أنواع البيانات شيوعًا، وتأتي بايثون مع مجموعة من توابع معالجة النصوص المفيدة. ستستخدم الفهرسة والتقسيم وتوابع السلاسل النصية في كل برنامج بايثون تكتبه تقريبًا.

صحيحٌ أن البرامج التي تكتبها الآن لا تبدو معقدة جدًا، فهي لا تحتوي على واجهات مستخدم رسومية فيها صور ونصوص ملونة، فكل ما نستعمله هو الدالة `print()` لطباعة النصوص و `input()` لقبول مدخلات المستخدم؛ لكن يستطيع المستخدم إدخال مجموعة كبيرة من النصوص عبر نسخها إلى الحافظة، مما يفيد كثيرًا في معالجة كميات كبيرة من النصوص. وصحيحٌ أن هذه البرامج لا تملك واجهات رسومية جميلة لكنها تفعل الكثير بجهدٍ قليل.

طريقة أخرى لمعالجة كمية كبيرة من النصوص هي كتابة الملفات وقراءتها من نظام الملفات المحلي مباشرةً، وسنتعلم كيفية فعل ذلك في الفصل التاسع من هذا الكتاب. لقد شرحنا حتى الآن المفاهيم الأساسية في برمجة بايثون! ستتعلم مفاهيم جديدة في بقية هذه الكتاب، لكن يفترض أنك تعرف ما يكيفك لكتابة برامج مفيدة تستطيع عبرها أتمتة المهام.

قد تظن أنك لا تمتلك المعرفة الكافية في بايثون لتنزيل صفحات الويب أو تحديث جداول البيانات أو إرسال الرسائل البريدية، لكن هنا يأتي دور الوحدات الخارجية في بايثون، التي توفر لك ما تحتاج إليه لفعل كل ذلك وأكثر منه، والتي سنتعلمها سويةً في الفصول القادمة.

دورة تطوير الألعاب



مميزات الدورة

- بناء معرض أعمال قوي بمشاريع حقيقية
- وصول مدى الحياة لمحتويات الدورة
- تحديثات مستمرة على الدورة مجانًا
- شهادة معتمدة من أكاديمية حسوب
- إرشادات من المدربين على مدار الساعة
- من الصفر دون الحاجة لخبرة مسبقة

اشترك الآن

7. التعابير النمطية في لغة بايثون

من المرجح أنك تعرف كيف تبحث عن النصوص بالضغط على `Ctrl+f` وإدخال الكلمات التي تريد البحث عنها، في حين أن التعابير النمطية Regular expressions تنقل الأمور إلى مرحلة أعلى، فهي تسمح لك بتحديد «نمط» النص الذي تبحث عنه، فلو لم تكن تعرف رقم هاتف الشركة في الورقة أمامك، لكنك تعيش في الولايات المتحدة أو كندا، فستعلم أن أرقام الهاتف تتألف من 3 أرقام ثم شرطة `-`، ثم 4 أرقام أخرى (وقد يبدأ برمز المنطقة وهو 3 أرقام في البداية). أي أنك ستعرف أن ما يلي هو رقم هاتف 415-555-1234 لكن 4,155,551,234 ليس رقمًا هاتفيًا.

يمكننا التعرف على مختلف أنماط النصوص بسهولة: فعناوين البريد الإلكتروني تحتوي الرمز `@` في منتصفها، وعناوين URL للمواقع تحتوي على نقط وخطوط مائلة، والعناوين الأخبارية تستعمل نسق العنوان Title case، والتاغات في وسائل التواصل الاجتماعي تبدأ برمز `#` ولا تحتوي على فراغات ...إلخ.

التعابير النمطية مفيدة جدًا، لكن قلّة من غير المبرمجين يعرفونها على الرغم من أن أغلبية المحررات النصية الحديثة ومعالجات النصوص مثل مايكروسوفت وورد وليبرأوفيس رايتر يمتلكون خاصيات بحث واستبدال تستعمل التعابير النمطية. والتعابير النمطية تسر ع معالجة النصوص لدرجة كبيرة، فلا يستفيد منها مستخدمو برمجيات التحرير فقط، بل المبرمجون أيضًا، ويقول الكاتب الشهير كوري دكتورو أن علينا تدريس التعابير النمطية قبل تدريس البرمجة:

> «معرفة التعابير النمطية تعني الفرق بين حل المشكلة في 3 خطوات وحلها في 3,000 خطوة. إذا كنت تستعملها فستنسى أن المشكلة التي حللتها بضغطات قليلة تأخذ من الناس كمية كبيرة من العمل المضني»
> كوري دكتورو -بتصرف-.

سنبدأ هذا الفصل بكتابة برنامج لمطابقة نمط من النصوص دون استخدام التعابير النمطية، ثم سنرى كيف يمكننا كتابة شيفرة بسيطة باستخدام التعابير النمطية؛ وسنبدأ بالتعابير البسيطة ثم ننتقل إلى الميزات

المتقدمة أكثر، مثل استبدال النصوص وفئات الحروف، ثم سنكتب في نهاية الفصل مشروعًا بسيطًا يستخرج أرقام الهواتف وعناوين البريد الإلكتروني من مستند نصي.

7.1 مطابقة الأنماط دون استخدام التعابير النمطية

لنقل أننا نريد البحث عن رقم هاتف يتبع نظام الكتابة الأمريكي في سلسلة نصية، أي يحتوي 3 أرقام ثم شرطة ثم 3 أرقام ثم شرطة ثم 4 أرقام، مثل 415-555-4242.

لنكتب دالةً باسم `isPhoneNumber()` للتحقق إن كانت تمثل السلسلة النصية رقم هاتف، وتعيد القيمة `True` أو `False`.

احفظ الشيفرة الآتية في ملف باسم `isPhoneNumber.py`:

```
def isPhoneNumber(text):
 ❶ if len(text) != 12:
    return False
  for i in range(0, 3):
   ❷ if not text[i].isdecimal():
      return False
 ❸ if text[3] != '-':
    return False
  for i in range(4, 7):
   ❹ if not text[i].isdecimal():
      return False
 ❺ if text[7] != '-':
    return False
  for i in range(8, 12):
   ❻ if not text[i].isdecimal():
      return False
 ❼ return True
```

```
print('Is 415-555-4242 a phone number?')
print(isPhoneNumber('415-555-4242'))
print('Is ABC a phone number?')
print(isPhoneNumber(ABC))
```

حين تجربة المثال السابق فسينتج ما يلي:

```
Is 415-555-4242 a phone number?
True
Is ABC a phone number?
False
```

تحتوي الدالة `isPhoneNumber()` على شيفرة تجري عدة عمليات تحقق للتأكد أن السلسلة النصية في المعامل `text` هي رقم هاتف صالح، وإذا فشلت أي تحقق من هذه التحققات فستعيد الدالة القيمة `False`.

تبدأ الشيفرة بالتحقق أن السلسلة النصية بطول 12 محرفًا تمامًا ❶، ثم تتحقق أن رقم المنطقة (أي أول 3 محارف في `text`) تحتوي على محارف رقمية فقط ❷، بقية الدالة تتحقق أن السلسلة النصية تتبع نمط أرقام الهواتف: يجب أن تكون أول شرطة في الرقم بعد رمز المنطقة ❸، ثم 3 أرقام ❹، ثم شرطة ❺، ثم 4 أرقام ❻، وإن أنهينا كل عمليات التحقق بنجاح فستعيد الدالة `True` ❼.

استدعاء الدالة `isPhoneNumber()` مع الوسيط `'415-555-4242'` سيعيد `True`، بينما استدعاؤها مع `'ABC'` سيعيد `False`، إذ ستفشل أول عملية تحقق لأن السلسلة النصية `'ABC'` ليست بطول 12 محرفًا.

إذا أردت العثور على رقم هاتف في سلسلة نصية طويل، فستحتاج إلى كتابة المزيد من الشيفرات لمطابقة نمط رقم الهاتف.

بدِّل استدعاء الدالة `print()` في المثال السابق إلى ما يلي:

```
message = 'Call me at 415-555-1011 tomorrow. 415-555-9999 is my
office.'
for i in range(len(message)):
    ❶ chunk = message[i:i+12]
    ❷ if isPhoneNumber(chunk):
        print('Phone number found: ' + chunk)
print('Done')
```

سينتج البرنامج الناتج الآتي:

```
Phone number found: 415-555-1011
Phone number found: 415-555-9999
Done
```

في كل دورة لحلقة `for` سنأخذ 12 محرفًا من المتغير `message` ونسندها إلى المتغير `chunk` ❶، فمثلًا في أول دورة لحلقة التكرار تكون قيمة `i` هي `0`، وستسند القيمة `message[0:12]` إلى المتغير `chunk` (أي السلسلة النصية `'Call me at 4'`)، وفي الدورة التالية تكون قيمة `i` تساوي `1`، وستسند القيمة `message[1:13]` إلى `chunk` (أي السلسلة النصية `'all me at 41'`)، بعبارة أخرى: في كل دورة من حلقة `for` ستتغير قيمة `chunk` بزيادة مكان بدء عملية الاقتطاع بمقدار واحد، ويكون طولها 12 محرفًا:

- `'Call me at 4'`
- `'all me at 41'`
- `'ll me at 415'`
- `'l me at 415-'`
- وهلم جرًا...

سنمرر بعدئذٍ المتغير `chunk` إلى الدالة `isPhoneNumber()` من أجل لتحقق مما إذا كان يطابق نمط أرقام الهواتف ❷، وإذا طابقها فسنطبع القيمة المطابقة.

سنستمر في تنفيذ حلقة التكرار التي ستمر على السلسلة النصية كلها، وستختبر كل 12 محرفًا فيها على حدة، وتطبع قيمة `chunk` إن أعادت الدالة `isPhoneNumber()` القيمة `True`، وبعد المرور على جميع محارف السلسلة النصية `message` فسنطبع الكلمة `Done`.

على الرغم من أن السلسلة النصية في هذا المثال `message` قصيرة، لكنها قد تكون طويلة جدًا وفيها ملايين المحارف، وسيعالجها البرنامج في أقل من ثانية؛ لكن إن كتبنا برنامجًا يطابق أرقام الهواتف باستخدام التعابير النمطية فسيعمل بسرعة مشابهة تقريبًا، لكن برمجته أسهل بكثير.

7.2 العثور على تعبير نصي باستخدام التعابير النمطية

يعمل البرنامج الذي كتبناه في القسم السابق عملًا صحيحًا، لكننا احتجنا إلى كتابة شيفرة كثيرة للقيام بأمر بسيط؛ فطول الدالة `isPhoneNumber()` هو 17 سطرًا لكنها تستطيع مطابقة نمط واحد من أرقام الهواتف. فماذا عن الأرقام المكتوبة بالشكل `415.555.4242` أو `(415) 555-4242`؟ ماذا لو كان هنالك رمز تحويل بعد رقم الهاتف مثل `415-555-4242 x99`؟ ستفشل الدالة `isPhoneNumber()` بمطابقتها. صحيحٌ أنك تستطيع كتابة شيفرات إضافية للتحقق من هذه الحالات، لكن هنالك طريقة أسهل بكثير.

التعابير النمطية Regular Expression أو اختصارًا Regex تصف نمط النصوص (ومن هنا أتى اسمها). فمثلًا `\d` في صياغة التعابير النمطية تعني محرف رقمي، أي رقم مفرد من 0 حتى 9. ويستعمل التعبير النمطي `\d\d\d-\d\d\d-\d\d\d\d` في بايثون لمطابقة النص الذي تطابقه الدالة `isPhoneNumber()` السابقة: سلسلة من 3 أرقام، ثم شرطة، ثم 3 أرقام، ثم شرطة، ثم 4 أرقام. ولن تطابق أي سلسلة نصية لها نمط آخر التعبير النمطي `\d\d\d-\d\d\d-\d\d\d\d`.

لكن يمكن أن تكون التعابير النمطية أعقد من ذلك بكثير، فمثلًا إضافة الرقم 3 بين قوسين مجعدين `{3}` يعني «طابق هذا النمط 3 مرات»، وبالتالي يمكننا كتابة التعبير النمطي السابق باختصاره إلى `\d{3}-\d{3}-\d{4}` وستكون النتيجة نفسها.

7.2.1 إنشاء كائنات Regex

جميع دوال التعابير النمطية موجودة في الوحدة `re`، وعلينا استيرادها أولًا:

```
>>> import re
```

> **ملاحظة**: أغلبية أمثلة هذا الفصل تستعمل الوحدة `re`، لذا من المهم أن تتذكر استيرادها في بداية كل سكربت أو حينما تعيد تشغيل محرر Mu، وإلا فستحصل على رسالة الخطأ NameError: name 're' is not defined.

تمرير سلسلة نصية تمثل التعبير النمطي إلى `re.compile()` سيعيد كائن `Regex`.

لإنشاء كائن `Regex` يطابق نمط أرقام الهواتف السابق، فأدخل ما يلي إلى الطرفية التفاعلية. تذكر أن `\d` تعني «محرف رقمي»، و `\d\d\d-\d\d\d-\d\d\d\d` هو التعبير النمطي الذي سيطابق رقم الهاتف.

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
```

يحتوي المتغير `phoneNumRegex` الآن على كائن `Regex`.

7.2.2 مطابقة كائنات Regex

يمتلك الكائن `Regex` التابع `search()` الذي يبحث في السلسلة النصية التي مررت إليها لأي مطابقة للتعبير النمطية. سيعيد التابع `search()` القيمة `None` إذا لم يُطابَق التعبير النمطي في السلسلة النصية، وإذا عُثر على التعبير النمطي فسيعيد التابع `search()` كائن `Match`، الذي فيه التابع `group()` الذي يعيد النص الذي جرت مطابقته مع التعبير النمطي (سنشرح المجموعات لاحقًا فلا تقلق):

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> print('Phone number found: ' + mo.group())
Phone number found: 415-555-4242
```

اسم المتغير `mo` هو اسم عام لكائنات `Match`. قد يبدو المثال السابق معقدًا لكنه بصراحة أقصر بكثير من برنامج `isPhoneNumber.py`.

بدايةً مررنا التعبير النمطي الذي نريده إلى `re.compile()` وحفظنا كائن `Regex` الناتج في المتغير `phoneNumRegex`، ثم استدعينا التابع `search()` على `phoneNumRegex` ومررنا السلسلة النصية التي نريد البحث فيها عن النمط إلى التابع `search()`.

ستخزن نتيجة البحث في المتغير `mo`، ونحن نعرف في هذا المثال أن التابع سيعيد الكائن `Match`، ولمعرفتنا أن المتغير `mo` سيحتوي على كائن `Match` وليس قيمة فارغة `None`، فاستدعينا التابع `group()` على `mo` لإعادة الناتج، ولأننا كتبنا `mo.group()` داخل الدالة `print()` فسنعرض الناتج الذي جربت مطابقته `415-555-4242`.

7.2.3 مراجعة لعملية لمطابقة التعابير النمطية

هنالك عدة خطوات لاستعمال التعابير النمطية في بايثون، وكل واحدة منها بسيطة وسهلة:

1. استيراد الوحدة `Regex` بكتابة `import re`.
2. إنشاء كائن `Regex` مع دالة `re.compile()`. لا تنس استعمال سلسلة نصية خام `raw` بكتابة `r` قبلها.
3. تمرير السلسلة النصية التي تريد البحث فيها إلى التابع `search()` وسيعيد كائن من النوع `Match`.
4. استدعاء الدالة `group()` للكائن `Match` التي تعيد السلسلة النصية المطابقة.

> **ملاحظة**: صحيحٌ أنني أنصحك أن تجرب الشيفرات في الطرفية التفاعلية لكنك تستطيع استخدام تطبيقات ويب لاختبار التعابير النمطية التي تظهر لك بوضوح كيف يطابق التعبير النمطي النص الذي أدخلته. أنصحك بتجربة pythex.org.

7.3 ميزات إضافية لمطابقة النصوص عبر التعابير النمطية

لقد تعلمنا الخطوات الأساسية لإنشاء والبحث عبر التعابير النمطية في بايثون، وأصبحنا جاهزين لتجربة ميزات أقوى لها.

7.3.1 التجميع مع الأقواس

لنقل أنك تريد أن تفصل رقم المنطقة من بقية رقم الهاتف. إذا أضفنا أقواسًا في التعابير النمطية فستنشِئ مجموعات groups كما في `(\d\d\d-\d\d\d\d)-(\d\d\d)`، ثم يمكننا استخدام التابع `group()` للكائن `Match` للحصول على النص المطابق من مجموعة معينة.

ستخزن القيمة المطابقة من أول مجموعة في المجموعة 1، والمجموعة الثانية في 2... وإذا مررنا القيمة 1 أو 2 إلى التابع group() فسنحصل على أجزاء مختلفة من النص الذي جرت مطابقته. أما تمرير القيمة 0 أو عدم تمرير أي قيمة إلى التابع group() فسيعيد كامل النص الذي جرت مطابقته من التعبير النمطي:

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> mo.group(1)
'415'
>>> mo.group(2)
'555-4242'
>>> mo.group(0)
'415-555-4242'
>>> mo.group()
'415-555-4242'
```

إذا أردت الحصول على جميع المجموعات في آن واحد، فاستعمل التابع groups() (لاحظ أن اسم التابع بالجمع وليس المفرد):

```
>>> mo.groups()
('415', '555-4242')
>>> areaCode, mainNumber = mo.groups()
>>> print(areaCode)
415
>>> print(mainNumber)
555-4242
```

يعيد التابع mo.groups() صفًا tuple فيه أكثر من قيمة، ويمكنك استعمال الإسناد المتعدد لإسناد كل قيمة إلى متغير كما في السطر الآتي:

```
areaCode, mainNumber = mo.groups()
```

للأقواس معنى خاص في التعابير النمطية، لكن ماذا نفعل لو أردنا مطابقة الأقواس في النص؟ لنقل مثلًا أن رمز المنطقة في أرقام الهواتف التي تريد مطابقتها موجودة بين قوسين، وفي هذه الحالة سنحتاج إلى تهريب المحرفين (و) عبر شرطة مائلة خلفية:

```
>>> phoneNumRegex = re.compile(r'(\(\d\d\d\)) (\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('My phone number is (415) 555-4242.')
>>> mo.group(1)
```

```
'(415)'
>>> mo.group(2)
'555-4242'
```

سيطابق المحرفان المهربان (و) في السلسلة النصية الخام الممررة إلى الدالة `re.compile()` الأقواس في السلسلة النصية التي سنبحث فيها. هنالك معانٍ خاصة للمحارف الآتية في التعابير النمطية:

```
. ^ $ * + ? { } [ ] \ | ( )
```

في حال أردت مطابقة أحد تلك المحارف في النص الذي تريد البحث فيه، فعليك تهريبها بوضع خط مائل خلفي قبلها:

```
\. \^ \$ \* \+ \? \{ \} \[ \] \\ \| \( \)
```

تأكد أنك لم تخلط بين الأقواس المهربة وبين أقواس المجموعات في التعابير النمطية، إذا ظهرت لك رسالة خطأ حول «قوس ناقص» أو «أقواس غير متوازنة» فاعلم أنك نسيت إغلاق قوس مجموعة غير مهرب كما يلي

```
>>> re.compile(r'(\(Parentheses\)')
Traceback (most recent call last):
  --snip--
re.error: missing ), unterminated subpattern at position 0
```

تقول لك رسالة الخطأ أن هنالك قوس مفتوح في الفهرس 0 من السلسلة النصية `r'((Parentheses)'` الذي لا يوجد له قوس إغلاق.

7.3.2 مطابقة أكثر من مجموعة باستخدام الخط العمودي

محرف الخط العمودي | والذي يسمى كذلك بالأنبوب Pipe، يستعمل في أي مكان تريد فيه مطابقة أحد التعابير الموفرة. أي لنقل أن لدينا مثلًا التعبير النمطي `r'Batman|Yasmin'` الذي سوف يطابق `'Batman'` أو `'Yasmin'`.

فلو كانت الكلمتان Batman و Yasmin موجودةً في السلسلة النصية التي سنبحث فيها، فستعاد أول قيمة تطابق إلى الكائن `Match`:

```
>>> heroRegex = re.compile (r'Batman|Yasmin')
>>> mo1 = heroRegex.search('Batman and Yasmin')
>>> mo1.group()
'Batman'
```

```
>>> mo2 = heroRegex.search('Yasmin and Batman')
>>> mo2.group()
'Yasmin'
```

ملاحظة: يمكنك العثور على جميع حالات المطابقة باستخدام التابع `findall()` المشروحة في هذا الفصل.

يمكنك أيضًا استخدام الخط العمودي لمطابقة تعابير فرعية مختلفة، فمثلًا لو قلنا أننا نريد مطابقة أي سلسلة نصية من `'Batman'` و `'Batmobile'` و `'Batcopter'` و `'Batbat'`، ولأن كل هذه السلاسل النصية تبدأ بالكلمة `Bat` فمن المنطقي كتابة هذه السابقة مرة واحدة، ويمكننا فعل ذلك عبر الأقواس كما يلي:

```
>>> batRegex = re.compile(r'Bat(man|mobile|copter|bat)')
>>> mo = batRegex.search('Batmobile lost a wheel')
>>> mo.group()
'Batmobile'
>>> mo.group(1)
'Mobile'
```

استدعاء التابع `mo.group()` يعيد النص `'Batmobile'` بينما `mo.group(1)` يعيد النص المطابق داخل مجموعة الأقواس الأولى، أي `'mobile'`. استخدام الخط العمودي `|` يتيح لنا مطابقة أحد التعابير النمطية الموفرة، وإذا أردنا أن نطابق الخط العمودي نفسه في السلسلة النصية المبحوث فيها فلا ننسى تهريبه `|\`.

7.3.3 المطابقة الاختيارية عبر إشارة الاستفهام

قد ترغب أحيانًا بمطابقة نمط اختياريًا، أي أن التعبير النمطي سيطابق السلسلة النصية بغض النظر إن احتوت السلسلة النصية على ذاك النمط أم لا. وتشير علامة الاستفهام `?` أن النمط الذي يسبقها اختياري:

```
>>> batRegex = re.compile(r'Bat(wo)?man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'

>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'
```

لاحظ أن الجزء `(wo)?` يعني أن النمط `wo` هو مجموعة اختيارية، فسيطابَق التعبير النمطي لو احتوت السلسلة النصية على `0` أو `1` من السلسلة النصية `wo` داخلها. وهذا يعني أن التعبير النمطي سيطابق `'Batman'` و `'Batwoman'` معًا.

لو عدنا إلى مثال أرقام الهواتف السابق، يمكننا تعديل التعبير النمطي لكي يبحث عن أرقـام الهواتـف الـتي تحتوي أو لا تحتوي على رمز المنطقة:

```
>>> phoneRegex = re.compile(r'(\d\d\d-)?\d\d\d-\d\d\d\d')
>>> mo1 = phoneRegex.search('My number is 415-555-4242')
>>> mo1.group()
'415-555-4242'
>>> mo2 = phoneRegex.search('My number is 555-4242')
>>> mo2.group()
'555-4242'
```

يمكنك أن تقول أن `?` يعني «طابق النمط الفرعي السابق `0` مـرة أو مـرة واحـدة»، وإذا أردنـا مطابقـة علامـة الاستفهام فلا ننسى تهريبها بكتابة `?\`.

7.3.4 المطابقة صفر مرة أو أكثر باستخدام رمز النجمة

رمز النجمة `*` asterisk يعني «طابق صفر مرة أو أكثر»، فاستعمال النجمـة يعـني أن التعبـير النمطي الـذي يسبقها سيطابق لأي عدد من المرات في السلسلة النصية؛ فقد لا يكون موجودًا أو يكون مكررًا مرات كثيرة:

```
>>> batRegex = re.compile(r'Bat(wo)*man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'

>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'

>>> mo3 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo3.group()
'Batwowowowoman'
```

لاحظ أن التعبير النمطي `(wo)*` سيُطابَق `0` مـرة في `'Batman'`، ومـرة واحـدة في `'Batwoman'`، وأربـع مرات في `'Batwowowowoman'`. إذا أردنا مطابقة النجمة نفسها فلا ننسى تهريبها بكتابة `*\`.

7.3.5 المطابقة مرة واحدة أو أكثر باستخدام إشارة الجمع

على خلاف إشارة النجمة * التي تطابق النمط صفر مرة أو أكثر، تعني إشارة الجمع أو إشارة الزائد + «مطابقة النمط مرة واحدة أو أكثر»، هذا يعني أن النمط الذي يسبق إشارة + يجب أن يكون موجودًا على الأقل مرة واحدة، على خلاف إشارة النجمة التي تسمح بألّا يكون النمط موجودًا في السلسلة النصية.

قارن بين أثر رمز النجمة في القسم السابق والمثال الآتي:

```
>>> batRegex = re.compile(r'Bat(wo)+man')
>>> mo1 = batRegex.search('The Adventures of Batwoman')
>>> mo1.group()
'Batwoman'

>>> mo2 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo2.group()
'Batwowowowoman'

>>> mo3 = batRegex.search('The Adventures of Batman')
>>> mo3 == None
True
```

التعبير النمطي `Bat(wo)+man` لا يطابق السلسلة النصية `'The Adventures of Batman'` لأن من المطلوب وجود التعبير الفرعي `wo` مرة واحدة على الأقل. إذا أردنا مطابقة إشارة الزائد فلا ننسى تهريبها `+`.

7.3.6 مطابقة النمط لعدد معين من المرات باستخدام الأقواس المجعدة

إذا كانت لديك مجموعة تريد تكرارها لعدد معين من المرات، فأتبع تلك السلسلة برقم محاط بأقواس `{}`. فمثلًا التعبير النمطي `(Ha){3}` يطابق السلسلة النصية `'HaHaHa'` لكنه لن يطابق `'HaHa'` لأنها تحتوي على تكرارين للتعبير `Ha` فقط.

وبدلًا من كتابة رقم واحد، يمكننا تحديد مجال من التكرارات بكتابة الحد الأدنى، ثم فاصلة، ثم الحد الأقصى ضمن القوسين، مثلًا التعبير النمطي `(Ha){3,5}` سيطابق التعبير `'HaHaHa'` و `'HaHaHaHa'` و`'HaHaHaHaHa'`.

يمكنك ألّا تحدد الرقم الأدنى أو الأقصى ضمن القوسين لتترك المجال مفتوحًا، فمثلًا {3,}(Ha) سيطبق 3 تكرارات أو أكثر من المجموعة `(Ha)`، بينما `(Ha){,5}` سيطابق المجموعة `(Ha)` من 0 حتى 5 تكرارات.

ستساعدنا الأقواس المجعدة على تقصير التعبير النمطي، إذ يتساوى التعبيران النمطيان الآتيان:

```
(Ha){3}
(Ha)(Ha)(Ha)
```

وسيتساوي أيضًا:

```
(Ha){3,5}
((Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha)(Ha))
```

جرب ما يلي في الطرفية التفاعلية:

```
>>> haRegex = re.compile(r'(Ha){3}')
>>> mo1 = haRegex.search('HaHaHa')
>>> mo1.group()
'HaHaHa'

>>> mo2 = haRegex.search('Ha')
>>> mo2 == None
True
```

سيطابق `(Ha){3}` السلسلة النصية `'HaHaHa'` لكنه لن يطابق `'Ha'`، وبالتالي سيعيد التابع `search()` القيمة None.

7.4 المطابقة الجشعة والمطابقة غير الجشعة

لمّا كان التعبير `(Ha){3,5}` يطابق 3 أو 4 أو 5 تكرارات من `Ha` في السلسلة النصية `'HaHaHaHaHa'` فستتسائل لماذا يعيد استدعاء التابع `group()` على الكائن `Match` السلسلة النصية `'HaHaHaHaHa'` بدلًا من الاحتمالات الأخرى، ففي النهاية `'HaHaHa'` و `'HaHaHaHa'` هي مطابقات صالحة في التعبير النمطي `(Ha){3,5}`.

تكون التعابير النمطية في بايثون جشعة greedy افتراضيًا، وهذا يعني أنه في الحالات غير المحددة ستحاول التعابير النمطية مطابقة أطول سلسلة نصية ممكنة؛ أما المطابقة غير الجشعة non-greedy (وتسمى أحيانًا بالمطابقة الكسولة lazy) تطابق أقصر سلسلة نصية ممكنة، ونستطيع تحديد أننا نريد مطابقة غير جشعة عبر وضع علامة استفهام بعد قوس الإغلاق المجعد.

جرب التالي ولاحظ الاختلاف بين النسخة الجشعة وغير الجشعة وما ستطابقه في السلسلة النصية نفسها:

```
>>> greedyHaRegex = re.compile(r'(Ha){3,5}')
>>> mo1 = greedyHaRegex.search('HaHaHaHaHa')
```

```
>>> mo1.group()
'HaHaHaHaHa'

>>> nongreedyHaRegex = re.compile(r'(Ha){3,5}?')
>>> mo2 = nongreedyHaRegex.search('HaHaHaHaHa')
>>> mo2.group()
'HaHaHa'
```

لاحظ أن علامة الاستفهام لها معنيان في التعابير النمطية، الأول هو المطابقة غير الجشعة، والثاني هو جعل المجموعة اختيارية. وهذان المعنيان غير مرتبطين ببعضهما.

7.5 التابع findall()

بالإضافة إلى التابع `search()` تمتلك كائنات `Regex` التابع `findall()`، وفي حين أن التابع `search()` يعيد كائن `Match` لأول جزء مطابَق من السلسلة النصي التي نبحث فيها، يعيد التابع `findall()` جميع المطابقات في السلسلة النصية.

لنرى كيف يعيد التابع `search()` الكائن `Match` على أول سلسلة نصية مطابقة:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('Cell: 415-555-9999 Work: 212-555-0000')
>>> mo.group()
'415-555-9999'
```

وفي المقابل لن يعيد التابع `findall()` الكائن `Match`، بل قائمة فيها سلاسل نصية، لطالما لم تكن هنالك مجموعات فرعية في التعبير النمطي؛ وتمثل كل سلسلة نصية في القائمة المعادة الجزء من النص المطابق للتعبير النمطي:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') # لا توجد تعابير
فرعية
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
['415-555-9999', '212-555-0000']
```

أما لو كانت هنالك مجموعات أو تعابير فرعية في التعبير النمطي، فسيعيد التابع `findall()` قائمةً من الصفوف tuples، ويمثل كل صف مطابقةً، وعناصر الصف هي السلاسل النصية المطابقة لكل تعبير فرعي في التعبير النمطي.

لنرى الكلام السابق عمليًا لنفهمه. جرب الآتي في الطرفية التفاعلية ولاحظ وجود أنماط فرعية على مستوى التعبير النمطي:

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)') # توجد تعابير فرعية
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
[('415', '555', '9999'), ('212', '555', '0000')]
```

لنلخص ما الذي يعيد التابع `findall()`:

عند استدعائه على تعبير نمطي لا يحتوي على مجموعات، مثل `\d\d\d-\d\d\d-\d\d\d\d` فسيعيد التابع `findall()` قائمةً من السلاسل النصية مثل `['415-555-9999', '212-555-0000']`. عند استدعائه على تعبير نمطي يحتوي على مجموعات أو تعابير نمطية فرعية، مثل `(\d\d\d)-(\d\d\d-(\d\d\d\d)` فسيعيد التابع `findall()` قائمةً من الصفوف التي تحتوي على سلاسل نصية، سلسلة نصية لكل مجموعة، مثل

```
[('415', '555', '9999'), ('212', '555', '0000')]
```

7.6 فئات المحارف

تعلمنا في مثال أرقام الهواتف السابق أن \d يطابق أي رقم، أي أنه اختصار للتعبير `(0|1|2|3|4|5|6|7|8|9)`. هنالك عدد من فئات المحارف المختصرة، والتي تستطيع معرفتها من الجدول الآتي.

اختصار فئة المحارف	يمثل
`\d`	أي محرف يمثل رقمًا من 0 حتى 9
`\D`	أي محرف ليس رقمًا من 0 حتى 9
`\w`	أي حرف أو رقم أو الشرطة السفلية، يمكننا أن نقول أنه يطابق محارف «الكلمات»
`\W`	أي محرف ليس حرفًا أو رقمًا أو شرطةً سفلية (عكس `\w`)
`\s`	أي مسافة فارغة أو مسافة جدولة أو سطر جديد، يمكننا أن نقول أنه يطابق «الفراغات»
`\S`	أي محرف ليس فراغًا أو مسافة جدولة أو سطر جديد

الجدول 10: أي محرف يمثل رقمًا من 0 حتى 9

تساعد فئات المحارف Character classes على اختصار التعابير النمطية، ففئة المحارف `[0-5]` تطابق الأرقام من `0` إلى `5` فقط، وهذا أكثر اختصارًا من كتابة `(0|1|2|3|4|5)`.

لاحظ أننا نملك `\d` لمطابقة الأرقام فقط، لكن `\w` يطابق الأرقـام والأحـرف والشـرطة السـفلية _، ولا يوجـد اختصار لمطابقة الأحرف فقط، لكنك تستطيع أن تكتب `[a-zA-Z]` التي سنشرحها لاحقًا.

```
>>> xmasRegex = re.compile(r'\d+\s\w+')
>>> xmasRegex.findall('12 drummers, 11 pipers, 10 lords, 9 ladies, 8
maids, 7
swans, 6 geese, 5 rings, 4 birds, 3 hens, 2 doves, 1 partridge')
['12 drummers', '11 pipers', '10 lords', '9 ladies', '8 maids', '7
swans', '6
geese', '5 rings', '4 birds', '3 hens', '2 doves', '1 partridge']
```

سيطابق التعبير النمطي `\d+\s\w+` أي نص فيـه رقم واحـد أو أكـثر `+\d` يكـون متبوعًـا بفـراغ `\s` ويكـون متبوعًا برقم أو حرف أو شرطة سفلية `+\w`.

سيعيد التابع `findall()` السلاسل النصية المطابقة من التعبير النمطي في قائمة.

7.7 كتابة فئات محارف مخصصة

هنالك حالات نحتاج فيها إلى استخدام مجموعة من المحارف لكن الفئات المختصرة الشائعة (مثل `\d` و`\w` و `\s` ...إلخ.) غير مناسبة لحالتنا؛ لذا يمكننا تعريف فئـات المحـارف بأنفسـنا باسـتعمال الأقـواس المربعـة `[]`. فمثلًا فئة المحارف `[aeiouAEIOU]` ستطابق أي حرف صوتي سواءً كان بالحالة الصغيرة أو الكبيرة:

```
>>> vowelRegex = re.compile(r'[aeiouAEIOU]')
>>> vowelRegex.findall('RoboCop eats baby food. BABY FOOD.')
['o', 'o', 'o', 'e', 'a', 'a', 'o', 'o', 'A', 'O', 'O']
```

يمكنك أيضًا تضمين مجال من الأرقام أو الأحرف باستخدام الشـرطة، فمثلًا فئـة المحـارف `[a-zA-Z0-9]` ستطابق جميع الأحرف الصغيرة والأحرف الكبيرة والأرقام.

لاحظ أن رموز التعابير النمطية العادية لن تفسر داخل الأقواس المربعة، فلا حاجة إلى تهريب المحارف . أو * أو ? أو `()` بخـط مائـل خلفي. فمثلًا فئـة المحـارف `[0-5.]` تطـابق الأعـداد من 0 إلى 5 ونقطـة. ولا حاجـة إلى تهريبها وكتابة `[0-5\.]`.

إذا وضعنا رمز القبعة caret ^ بعد قوس بداية فئة المحارف فسيعني «الـرفض»، أي سـيعكس محتويـات فئة المحارف، أي أنها ستطابق أي محرف ليس موجودًا في فئة المحارف المحددة:

```
>>> consonantRegex = re.compile(r'[^aeiouAEIOU]')
>>> consonantRegex.findall('RoboCop eats baby food. BABY FOOD.')
['R', 'b', 'C', 'p', ' ', 't', 's', ' ', 'b', 'b', 'y', ' ', 'f', 'd',
'.', '', 'B', 'B', 'Y', ' ', 'F', 'D', '.']
```

فبدلًا من مطابقة الأحرف الصوتية، أصبحت فئة المحارف تطابق كل شيء عدا الأحرف الصوتية.

7.8 رمز القبعة ورمز الدولار

يمكن أيضًا استخدام رمز القبعة ^ ببداية التعبير النمطي للإشارة أن المطابقة يجب أن تبدأ من بداية السلسلة النصية التي نبحث فيها عن النمط. و يمكننا استخدام رموز الدولار $ بنهاية التعبير النمطي للإشارة أنه يجب أن تنتهي السلسلة النصية بالتعبير النمطي. يمكننا استعمال الرمزين ^ و $ معًا للإشارة إلى أن كل السلسلة النصية يجب أن تطابق التعبير النمطي، فلا يسمح بإجراء مطابقة جزئية على السلسلة النصية فقط.

لنأخذ مثالًا بسيطًا التعبير النمطي `r'^Hello'` الذي سيطابق أي سلسلة نصية تبدأ بالكلمة `'Hello'`:

```
>>> beginsWithHello = re.compile(r'^Hello')
>>> beginsWithHello.search('Hello, world!')
<re.Match object; span=(0, 5), match='Hello'>
>>> beginsWithHello.search('He said hello.') == None
True
```

التعبير النمطي `r'\d$'` يطابق السلاسل النصية التي تنتهي برقم من 0 إلى 9:

```
>>> endsWithNumber = re.compile(r'\d$')
>>> endsWithNumber.search('Your number is 42')
<re.Match object; span=(16, 17), match='2'>
>>> endsWithNumber.search('Your number is forty two.') == None
True
```

أما التعبير `r'^\d+$'` فهو يطابق السلاسل النصية التي تحتوي على رقم واحد على الأقل:

```
>>> wholeStringIsNum = re.compile(r'^\d+$')
>>> wholeStringIsNum.search('1234567890')
<re.Match object; span=(0, 10), match='1234567890'>
>>> wholeStringIsNum.search('12345xyz67890') == None
True
>>> wholeStringIsNum.search('12  34567890') == None
True
```

لاحظ أن آخر استدعائين للتابع `search()` يوضحان كيف يجب أن تطابق السلسلة النصية كلها النمط، وليس جزءًا منها.

7.9 حرف البدل

تسمى النقطة `.` في التعابير النمطية بحرف البدل wildcard، وهي تطابق أي محرف عدا السطر الجديد. فعلى سبيل المثال:

```
>>> atRegex = re.compile(r'.at')
>>> atRegex.findall('The cat in the hat sat on the flat mat.')
['cat', 'hat', 'sat', 'lat', 'mat']
```

تذكر أن النقطة ستطابق محرفًا واحدًا فقط، وهذا هو السبب في أن التعبير النمطي طابق `lat` في السلسلة النصية `flat` فقط. إذا أردنا البحث عن رمز النقطة فلا ننسى تهريبه عبر خط مائل خلفي `.\`.

7.9.1 مطابقة كل شيء مع النقطة والنجمة

نحتاج أحيانًا إلى مطابقة كل شيء، فمثلًا لو أردنا مطابقة السلسلة النصية `'First Name:'` متبوعةً بأي نصف، فحينها سنستعمل النقطة والنجمة `.*` تذكر أن النقطة تعني أي حرف عدا السطر الجديد، والنجمة تعني تكرار النمط الذي يسبقها `0` مرة أو أكثر.

```
>>> nameRegex = re.compile(r'First Name: (.*) Last Name: (.*)')
>>> mo = nameRegex.search('First Name: Al Last Name: Sweigart')
>>> mo.group(1)
'Al'
>>> mo.group(2)
'Sweigart'
```

تجري النقطة والنجمة مطابقةً جشعةً، أي أنها تحاول أن تطابق النصوص أكثر ما يمكن، ولجعلها مطابقة غير جشعة يمكننا استخدام النقطة والنجمة وإشارة الاستفهام `.*?`. بالتالي ستطابق بايثون النمط مطابقةً غير جشعة.

جرب المثال الآتي لترى الفرق بين النسخة الجشعة وغير الجشعة:

```
>>> nongreedyRegex = re.compile(r'<.*?>')
>>> mo = nongreedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man>'

>>> greedyRegex = re.compile(r'<.*>')
```

```
>>> mo = greedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man> for dinner.>'
```

يمكننا ترجمة التعبيرين النمطيين السابقين إلى «طابق قوس بدء زاوية ثم أي شيء ثم قوس إغلاق زاويـة»، لكن السلسلة النصية `'<To serve man> for dinner.>'` فيها مطـابقتين لقـوس إغلاق زاويـة، ففي النسخة غير الجشعة تطابق بايثون أقصر سلسلة نصية ممكنة `'<To serve man>'`؛ أما في النسخة الجشعة فتحاول بايثون مطابقة أطول سلسلة نصية ممكنة: `'<To serve man> for dinner.>'`.

7.9.2 مطابقة الأسطر الجديدة مع رمز النقطة

لقد تعلمنا أن النقطة والنجمة ستطابق كل المحارف باستثناء السطر الجديـد، لكن بتمريـر وسـيط ثـاني إلى الدالة `re.compile()` هو `re.DOTALL` سنتمكن من استعمال النقطة لمطابقة جميع المحـارف بمـا ف ذلـك السطر الجديد:

```
>>> noNewlineRegex = re.compile('.*')
>>> noNewlineRegex.search('Serve the public trust.\nProtect the
innocent.
\nUphold the law.').group()
'Serve the public trust.'
>>> newlineRegex = re.compile('.*', re.DOTALL)
>>> newlineRegex.search('Serve the public trust.\nProtect the
innocent.
\nUphold the law.').group()
'Serve the public trust.\nProtect the innocent.\nUphold the law.'
```

التعبـير النمطي `noNewlineRegex` لم نمـرر إليـه `re.DOTALL` حين اسـتدعاء `re.compile()` لـذا سـيطابق النمـط كـل شـيء حـتى الوصـول إلى محـرف السـطر الجديـد؛ بينمـا `newlineRegex` مررنـا إليـه `re.DOTALL` حين اسـتدعاء `re.compile()` لـذا سـيطابق كـل شـيء، لهـذا أعـاد اسـتدعاء `newlineRegex.search()` السلسلة النصية كاملة بما فيها الأسطر الجديدة.

7.10 مراجعة لرموز التعابير النمطية

شرحنا الكثير من الرموز في هذا الفصل، لنراجع سريعًا ما الذي تعلمناه حول أساسيات التعابير النمطية:

- يطابق `?` المجموعة التي تسبقه 0 مرة أو 1 مرة.
- يطابق `*` المجموعة التي تسبقه 0 مرة أو أكثر.

- يطابق `+` المجموعة التي تسبقه 1 مرة أو أكثر.
- يطابق `{n}` المجموعة التي تسبقه n مرة تمامًا.
- يطابق `{n,}` المجموعة التي تسبقه n مرة أو أكثر.
- يطابق `{, m}` المجموعة التي تسبقه 0 مرة حتى m مرة.
- يطابق `{n, m}` المجموعة التي تسبقه n مرة على الأقل و m على الأكثر.
- تجري `{n,m}?` أو `*?` أو `+?` مطابقة غير جشعة.
- يطابق `^spam` السلاسل النصية التي تبدأ بالكلمة spam.
- يطابق `spam$` السلاسل النصية التي تنتهي بالكلمة spam.
- يطابق الرمز `.` أي محرف عدا السطر الجديد.
- تطابق فئة المحارف `\d` و `\w` و `\s` رقمًا أو كلمةً أو فراغًا على التوالي وبالترتيب.
- تطابق فئة المحارف `\D` و `\W` و `\S` أي شيء عدا أن يكون رقمًا أو كلمةً أو فراغًا على التوالي وبالترتيب.
- تطابق فئة المحارف `[abc]` أي شيء بين القوسين المربعين (مثل a أو b أو c).
- تطابق فئة المحارف `[^abc]` أي شيء ليس بين القوسين المربعين.

7.11 المطابقة غير الحساسة لحالة الأحرف

تطابق التعابير النمطية النص بنفس الحالة المحددة، إذ تطابق التعابير النمطية الآتية على سبيل المثال سلاسل نصية مختلفة:

```
>>> regex1 = re.compile('RoboCop')
>>> regex2 = re.compile('ROBOCOP')
>>> regex3 = re.compile('robOcop')
>>> regex4 = re.compile('RobocOp')
```

لكن في بعض الأحيان لا يهمنا ما هي حالة الأحرف، وكل ما نريده هو مطابقة النص بغض النظر عن حالته، فحينها نريد جعل التعابير النمطية غير حساسة لحالة الأحرف، وذلك بتمرير وسيطٍ ثانٍ للدالة `re.compile()` هو `re.IGNORECASE` أو `re.I`:

```
>>> robocop = re.compile(r'robocop', re.I)
>>> robocop.search('RoboCop is part man, part machine, all
cop.').group()
```

```
'RoboCop'
>>> robocop.search('ROBOCOP protects the innocent.').group()
'ROBOCOP'
>>> robocop.search('Al, why does your programming book talk about
robocop so much?').group()
'robocop'
```

7.12 استبدال السلاسل النصية عبر التابع sub()

لا تستعمل التعابير النمطية للعثور على أنماط من النصوص فقط، وإنما لاستبدالها أيضًا.

يقبل التابع `sub()` في كائنات `Regex` معاملين، الأول هو السلسلة النصية الـتي نريـد تبـديل المطابقـات إليها، والثاني هو السلسلة النصية التي نريـد البحث فيهـا عن مطابقـات لتبـديلها. يعيـد التـابع `sub()` سلسـلةً نصية بعد تطبيق جميع عمليات الاستبدال:

```
>>> namesRegex = re.compile(r'Agent \w+')
>>> namesRegex.sub('CENSORED', 'Agent Alice gave the secret documents
to Agent Bob.')
'CENSORED gave the secret documents to CENSORED.'
```

قد نحتاج أحيانًا إلى استخدام النص المطابق كجزء من النص الجديد، لذا يمكننا استخدام `\1` و `\2` و `\3` في الوسيط الأول الممرر إلى التابع `sub()` للوصول إلى المجموعات الفرعية المطابقة.

لنقل أننا نريد إخفاء أسماء الأشخاص في مثالنا السابق، لكننا نريد إظهار الحرف الأول من اسمهم فقط، في هذه الحالية سيكون بإمكاننا استخدام التعبير النمطي `Agent (\w)\w*` وتمرير `r'\1****'` كأول معامل إلى التابع `sub()`، وستشير `\1` إلى السلسلة النصـية المطابقـة من النمـط الفـرعي الأول، أي المجموعـة `(\w)` في التعبير النمطي:

```
>>> agentNamesRegex = re.compile(r'Agent (\w)\w*')
>>> agentNamesRegex.sub(r'\1****', 'Agent Alice told Agent Carol that
Agent
Eve knew Agent Bob was a double agent.')
A**** told C**** that E**** knew B**** was a double agent.'
```

7.13 التعامل مع التعابير النمطية المعقدة

التعابير النمطيـة الـتي تطـابق نصًـا بسـيطًا تكـون سـهلة الفهم، لكن إذا أردنـا مطابقـة النصـوص المعقـدة فستصبح التعابير النمطية معقدة وطويلة، يمكننا التعامل مع هذا الإشكال بالطلب من الدالة `re.compile()`

أن تتجاهل الفراغات والتعليقات داخل السلسلة النصية التي تمثل التعبير النمطي، وهذا النمط يسمى verbose mode، ويمكن تفعيله بتمرير `re.VERBOSE` إلى المعامل الثاني في الدالة `re.compile()`.

فبدلًا من محاولة قراءة تعبير نمطي صعب مثل هذا:

```
phoneRegex = re.compile(r'((\d{3}|\(\d{3}\))?(\s|-|\.)?\d{3}(\s|-|\.)
\d{4}
(\s*(ext|x|ext.)\s*\d{2,5})?)')
```

نستطيع تقسيم التعبير النمطي إلى عدة أسطر مع تعليقات توضح وظيفة كل قسم:

```
phoneRegex = re.compile(r'''(
  (\d{3}|\(\d{3}\))?            # رمز المنطقة
  (\s|-|\.)?                    # فاصل
  \d{3}                         # أول 3 أرقام
  (\s|-|\.)                     # فاصل
  \d{4}                         # آخر 4 أرقام
  (\s*(ext|x|ext.)\s*\d{2,5})?  # اللاحقة
  )''', re.VERBOSE)
```

لاحظ أننا استعملنا علامة الاقتباس الثلاثية لإنشاء سلسلة نصية متعددة الأسطر، لكي نستطيع تقسيم التعبير على عدة أسطر وتسهيل قراءته.

تتبع التعليقات داخل التعابير النمطية بنفس قواعد التعليقات في بايثون، حيث سيتم تجاهل كل شيء بعد الرمز #. لاحظ أن الفراغات الزائدة في السلسلة النصية متعددة الأسطر لا تؤثر على معنى التعبير النمطي ولا تعد جزءًا منه، لذا يمكنك استخدام الفراغات كيفما تشاء لتسهيل قراءة التعبير النمطي.

7.14 استخدام re.IGNORECASE و re.DOTALL و re.VERBOSE

ماذا لو أردنا استخدام `re.VERBOSE` لكتابة تعليقات في التعابير النمطية لكننا نريد أن نتجاهل حالة الأحرف أيضًا عبر `re.IGNORECASE`؟

للأسف لا تقبل الدالة `re.compile()` غير قيمة واحدة كثاني وسيط لها، لكن يمكننا تجاوز هذه المحدودية بجمع القيم `re.IGNORECASE` و `re.DOTALL` و `re.VERBOSE` مع بضعها عبر الخط العمودي | وهو يسمى في هذا السياق بعامل OR الثنائي Bitwise OR.

أي أننا لو أردنا كتابة تعبير نمطي غير حساس لحالة الأحرف ويسمح بمطابقة محرف السطر الجديد برمز النقطة، فحينها سيكون استدعاء الدالة `re.compile()` كما يلي:

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL)
```

ولو أردنا تضمين الخيارات كلها فنكتب:

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL |
re.VERBOSE)
```

أصل هذه الصيغة من الإصدارات القديمة من بايثون، وشرح العوامل الثنائية خارج سياق هذه السلسلة. لاحظ أن هنالك خيارات أخرى يمكن تمريرها إلى الدالة `re.compile()` لكنها غير شائعة الاستخدام، ويمكنك القراءة عنها في التوثيق الرسمي.

7.15 مشروع: برنامج استخراج أرقام الهواتف وعناوين البريد الإلكتروني

لنقل أنك موكل بمهمة مملة هي العثور على جميع أرقام الهواتف وعناوين البريد الإلكتروني في صفحة ويب طويلة أو مستند كبير. إذا كنت ستتصفح تلك الصفحة وتبحث عنها يدويًا فستأخذ منك وقتًا طويلًا وجهدًا كبيرًا؛ لكن إن كان لديك برنامج يمكنه البحث في الحافظة عن أرقام الهواتف وعناوين البريد الإلكتروني فسيكون ذلك رائعًا، فكل ما عليك فعله هو الضغط على `Ctrl+A` لتحديد كل النص ثم `Ctrl+C` لنسخه إلى الحافظة، بعد ذلك شغل برنامجك وسيعدل محتويات المحافظة ويضع فيها أرقام الهواتف وعناوين البريد الإلكتروني التي سيعثر عليها.

قد يغريك حينما تبدأ بمشروع جديد أن تبدأ كتابة الشيفرات فورًا، لكن من الأفضل أن تنظر نظرةً شاملة على البرنامج قبل البدء بكتابته، وأنصحك أن ترسم خطةً بسيطةً لما يجب على برنامجك فعله، دون التفكير بالشيفرات وكيفية كتابتها، وإنما اكتب العناوين العريضة فقط.

فمثلًا لو أردنا العمل على برنامج استخراج أرقام الهواتف وعناوين البريد الإلكتروني فسنحتاج إلى:

1. الحصول على النص من الحافظة.
2. العثور على جميع أرقام الهواتف وعناوين البريد الإلكتروني.
3. لصق الناتج في الحافظة.

يمكنك أن تبدأ الآن بالتفكير كيفية كتابة ذلك في بايثون. سيحتاج برنامجك إلى:

1. استخدام الوحدة `pyperclip` لنسخ ولصق النصوص من الحافظة.
2. إنشاء تعبيران نمطيان، واحد لمطابقة أرقام الهواتف والثاني لمطابقة عناوين البريد الإلكتروني.

3. العثور على جميع المطابقات لكلي التعبيرين النمطيين، وليس أول مطابقة فقط.
4. تنسيق الناتج في سلسلة نصية واحدة جاهزة ولصقها في الحافظة.
5. عرض رسالة خطأ إن لم يعثر على مطابقات في النص.

هذا هو المخطط العام للمشروع، وسنركز على كيفية حل كل خطوة حين كتابة الشيفرات. لاحظ أننا تعلمنا كيفية التعامل مع كل خطوة من الخطوات السابقة في بايثون.

7.15.1 الخطوة 1: إنشاء تعبير نمطي لأرقام الهواتف

علينا في البداية إنشاء تعبير نمطي للبحث عن أرقام الهواتف، لننشِئ ملفًا باسم `phoneAndEmail.py` ونكتب فيه:

```
#! python3
# phoneAndEmail.py - البحث عن أرقام الهواتف وعناوين البريد في الحافظة

import pyperclip, re
phoneRegex = re.compile(r'''(
  (\d{3}|\(\d{3}\))?            # رمز المنطقة
  (\s|-|\.)?                    # فاصل
  (\d{3})                       # أول 3 أرقام
  (\s|-|\.)                     # فاصل
  (\d{4})                       # آخر 4 أرقام
  (\s*(ext|x|ext.)\s*(\d{2,5}))?     # اللاحقة
  )''', re.VERBOSE)

# TODO: إنشاء التعبير النمطي لعناوين البريد الإلكتروني

# TODO: العثور على المطابقات في الحافظة

# TODO: لصق الناتج في الحافظة
```

التعليقات التي تبدأ بالكلمة TODO هي لبنية التطبيق، وسنبدلها إلى شيفرات لاحقًا.

يبدأ رقم الهاتف برمز منطقة اختياري، لذا نجعل المجموعة الفرعية التي تدل على رمز المنطقة اختياريةً بإضافة علامة استفهام `?`. ولأن رمز المنطقة يمكن أن يكون 3 أرقام (أي `\d{3}`) أو 3 أرقام ضمن أقواس (أي `\`

(\d{3}\)) فاستعملنا الخط العمودي | للفصل بينهما. لاحظ أننا استطعنا إضافة التعليقات إلى التعبير النمطي متعدد الأسطر لكي نتذكر ماذا يفعل كل سطر.

يمكن أن يكون الفاصل في أرقام الهواتف فراغًا \s أو شرطة - أو نقطة . ، لذا فصلنا بين هذه الاحتمالات بخط عمودي.

بقية أقسام التعبير النمطي واضحة وسهلة: 3 أرقام، يتبعها فاصل، يتبعها 4 أرقام، وآخر قسم هو اللاحقة الاختيارية التي تبدأ بأي عدد من الفراغات يليها الكلمة ext أو x أو ext.، ويليها رقمين إلى 5 أرقام.

> **ملاحظة**: من السهل أن نخلط بين التعابير النمطية التي تحتوي على مجموعات عبر استخدام الأقواس ( )، وبين الأقواس المُهربة (و). تذكر أن تتأكد أنك تستخدم النوع الصحيح من الأقواس إن حصلت على رسالة خطأ تشير إلى قوسٍ ناقص.

7.15.2 الخطوة 2: إنشاء تعبير نمطي لعناوين البريد الإلكتروني

علينا الآن إنشاء تعبير نمطي لمطابقة عناوين البريد الإلكتروني:

```
#! python3
# phoneAndEmail.py - البحث عن أرقام الهواتف وعناوين البريد في الحافظة
import pyperclip, re
phoneRegex = re.compile(r'''(
--مقتطع--


# إنشاء التعبير النمطي لعناوين البريد الإلكتروني
emailRegex = re.compile(r'''(
 ❶ [a-zA-Z0-9._%+-]+       # اسم المستخدم
 ❷ @                       # @ إشارة
 ❸ [a-zA-Z0-9.-]+          # اسم النطاق
  (\.[a-zA-Z]{2,4})        # اللاحقة
  )''', re.VERBOSE)
# TODO: العثور على المطابقات في الحافظة
# TODO: لصق الناتج في الحافظة
```

يكون اسم المستخدم جزءًا من البريد الإلكتروني ❶، وفيه محارف واحدة أو أكثر مما يلي: الأحرف الكبيرة والصغيرة، والأرقام، والنقطة، والشرطة السفلية، وإشارة النسبة المئوية، وإشارة زائد، وشرطة عادية. يمكننا جمع كل ما سبق في فئة المحارف الآتية `[a-zA-Z0-9._%+-]`.

يفصل بين اسم المستخدم والنطاق برمز `@` ❷؛ ولا يسمح اسم النطاق بنفس المحارف التي يسمح بها اسم المستخدم، ولذا يجب أن ننشِئ فئة محارف فيها الأرقام والأحرف والنقط والشرطات `[a-zA-Z0-9.-]` ❸.

آخر جزء هو اللاحقة، مثل com. (يسمى تقنيًا بالنطاق في أعلى مستوى top-level domain) الذي هو رمز النقطة ويلي أي شيء، ويفترض أن يكون بين 2 و 4 محارف.

هنالك قواعد كثيرة جدًا لمطابقة عناوين البريد الإلكتروني، والتعبير النمطي الذي كتبناه لن يطابق جميع عناوين البريد الصالحة قاعديًا، لكنه سيطابق جميع عناوين البريد الشائعة.

7.15.3 الخطوة 3: العثور على جميع المطابقات في الحافظة

أصبحت لدينا الآن التعابير النمطية التي ستطابق أرقام الهواتف وعناوين البريد الإلكتروني عبر الوحدة `re`، واستطعنا الوصول إلى محتويات الحافظة عبر الدالة `paste()` في الوحدة `pyperclip`. نحتاج الآن إلى استخدام التابع `findall()` للكائن `Regex` لإعادة قائمة من الصفوف. تأكد أن برنامجك يشبه البرنامج الآتي:

```
#! python3
# phoneAndEmail.py - البحث عن أرقام الهواتف وعناوين البريد في الحافظة
import pyperclip, re
phoneRegex = re.compile(r'''(
--مقتطع--

# العثور على المطابقات في الحافظة
text = str(pyperclip.paste())
❶ matches = []
❷ for groups in phoneRegex.findall(text):
    phoneNum = '-'.join([groups[1], groups[3], groups[5]])
    if groups[8] != '':
        phoneNum += ' x' + groups[8]
```

```
    matches.append(phoneNum)
❸ for groups in emailRegex.findall(text):
    matches.append(groups[0])
  # TODO: لصق الناتج في الحافظة
```

هنالك صف لكل مطابقة ناتجة، وكل صف يحتوي على السلاسل النصية في كل مجموعة في التعبير النمطي، تذكر أن المجموعة 0 تطابق كامل التعبير النمطي، لذا ما يهمنا من الناتج هو المجموعة في الفهرس 0.

سنخزن المطابقات في قائمة باسم matches ❶، وسنبدأ برنامجنا بقائمة فارغة، ثم لدينا حلقتَي for. ففي الحلقة الخاصة بالبريد الإلكتروني نضيف محتويات المجموعة 0 إلى القائمة matches ❸؛ أما في الحلقة الخاصة بأرقام الهواتف فلا نريد أن نضيف ناتج المجموعة 0 إلى القائمة. صحيحٌ أن برنامجنا يستطيع مطابقة أرقام الهواتف بأكثر من صيغة، لكننا نريد إضافة أرقام الهواتف بصيغة واحدة معيارية، لذا ننشِئ المتغير phoneNum الذي يحتوي على قيمة المجموعات 1 و 3 و 5 و 8 من النص المطابق، وهي تمثل رقم المنطقة، وأول 3 أرقام، وآخر 4 أرقام، واللاحقة على التوالي وبالترتيب.

7.15.4 الخطوة 4: جمع المطابقات في سلسلة نصية ونسخها إلى الحافظة

أصبحت لدينا الآن عناوين البريد الإلكتروني وأرقام الهواتف كقائمة من السلاسل النصية المخزنة في matches. إذا أردنا نسخ هذه القائمة إلى الحافظة فسنستعمل الدالة pyperclip.copy() التي تقبل سلسلة نصية، لكن المطابقات موجودة لدينا في قائمة، ولهذا نحن بحاجة إلى استخدام التابع join() على القائمة matches.

للتأكد أن برنامجنا يعمل كما ينبغي له، فلنطبع أي مطابقات إلى الطرفية، وإذا لم تطابق أي عناوين بريدية أو أرقام هواتف فسيخبر البرنامجُ المستخدمَ بذلك.

```
#! python3
# phoneAndEmail.py - البحث عن أرقام الهواتف وعناوين البريد في الحافظة

--مقتطع--
for groups in emailRegex.findall(text):
  matches.append(groups[0])


# Copy results to the clipboard
if len(matches) > 0:
```

```
    pyperclip.copy('\n'.join(matches))
    print('Copied to clipboard:')
    print('\n'.join(matches))
else:
    print('No phone numbers or email addresses found.')
```

7.15.5 تشغيل البرنامج

من أجل التجربة، افتح المتصفح وتوجه إلى صفحة تواصل معنا الآتية nostarch.com/contactus، ثم حددها كلها بالضغط على Ctrl+A ثم انسخها إلى الحافظة Ctrl+C، ثم شغل البرنامج. يجب أن يكون الناتج كما يلي:

```
Copied to clipboard:
800-420-7240
415-863-9900
415-863-9950
info@nostarch.com
media@nostarch.com
academic@nostarch.com
info@nostarch.com
```

7.15.6 أفكار لمشاريع مشابهة

هنالك تطبيقات كثيرة لعملية مطابقة أنماط من النص (واستبدلها عبر التابع sub()، فمثلًا يمكنك:

العثور على جميع روابط URL التي تبدأ بالسابقة http:// أو https://. العثور على التواريخ بصيغها المختلفة مثل 3/14/2022 أو 03-14-2022 أو 2022/3/14 وتبديلها إلى صيغة واحدة قياسية موحدة. إزالة البيانات الحساسة مثل أرقام البطاقات البنكية من المستندات. العثور على الأخطاء الشائعة مثل الفراغات المكررة بين الكلمات، أو الكلمات المكررة خطأً، أو علامات الترقيم المكررة.

7.16 أسئلة للتدريب

1. ما هي الدالة التي تنشِئ كائنات Regex؟
2. لماذا نستعمل السلاسل النصية الخام حين إنشاء كائنات Regex؟
3. ماذا يعيد التابع search()؟

4. كيف نحصل على السلسلة النصية المطابقة من الكائن `Match`؟
5. في التعبير النمطي `r'(\d\d\d)-(\d\d\d-\d\d\d\d)'` ماذا ستكون محتويات المجموعة `0`؟ والمجموعة `1`؟ والمجموعة `2`؟
6. تمتلك الأقواس والنقط معانٍ خاصة في التعابير النمطية. كيف يمكننا أن نخبر بايثون أننا نقصد مطابقة الأقواس والنقط مباشرةً؟
7. التابع `findall()` يعيد قائمةً من السلاسل النصية أو قائمةً من الصفوف فيها سلاسل نصية. ما الذي يجعلها تعيد واحدًا من الاثنين؟
8. إلى ما يرمز الخط العمودي `|` في التعابير النمطية؟
9. هنالك وظيفتين لإشارة الاستفهام `?` في التعابير النمطية، ما هما؟
10. ما الفرق بين إشارة الزائد `+` والنجمة `*` في التعابير النمطية؟
11. ما الفرق بين `{3}` و `{3,5}`؟
12. إلى ما تشير فئات المحارف `\d` و `\w` و `\s`؟
13. إلى ما تشير فئات المحارف `\D` و `\W` و `\S`؟
14. ما الفرق بين `.*` و `.*?`؟
15. ما هي فئة المحارف التي تطابق جميع الأرقام والأحرف بالحالة الصغيرة؟
16. كيف نجعل التعابير النمطية غير حساسةٍ لحالة الأحرف؟
17. ما الذي يطابقه المحرف `.` عادة؟ وماذا يحدث لو مررنا `re.DOTALL` كوسيط ثانٍ إلى الدالة `re.compile()`؟
18. إذا كان `numRegex = re.compile(r'\d+')` فماذا سيعيد التعبير `numRegex.sub('X', '12 drummers, 11 pipers, five rings, 3 hens')`؟
19. ماذا يحدث إذا مررنا `re.VERBOSE` إلى الدالة `re.compile()`؟
20. كيف تكتب تعبيرًا نمطيًا يطابق عددًا فيه فواصل بين كل ثلاثة أرقام؟ يجب أن يطابق الأعداد الآتية:
    - `'42'`
    - `'1,234'`
    - `'6,368,745'`

لكن لن يطابق:

- `'12,34,567'` (هنالك رقمان فقط بين الفواصل)
- `'1234'` (ليس فيها فواصل)

21. كيف تكتب تعبيرًا نمطيًا يطابق الاسم الكامل لأحد من عائلة Watanabe؟ يمكنك أن تفترض أن الاسم الأول يأتي قبل الاسم الأخير، ويتألف من كلمة واحدة ويبدأ بحرف كبيرة. يجب أن يطابق:
    - `'Haruto Watanabe'`
    - `'Alice Watanabe'`
    - `'RoboCop Watanabe'`

    لكنه لن يطابق:
    - `'ayman Sewd'` (الاسم الأول لا يبدأ بحرف كبير)
    - `'Mr. Sewd'` (الكلمة التي تسبق الاسم الأخير فيها رمز)
    - `'Sewd'` (ليس هنالك اسم أول)
    - `'Ayman sewd'` (الاسم الأخير لا يبدأ بحرف كبير)

22. كيف تكتب تعبيرًا نمطيًا يطابق الجملة التي تبدأ بالكلمة Alice أو Bob أو Carol؛ وتكون الكلمة الثانية هي eats أو pets أو throws، والكلمة الثالثة هي apples أو cats أو baseballs، وتنتهي الجملة بنقطة؟ يجب أن يكون التعبير النمطي غير حساس لحالة الأحرف، ويجب أن يطابق:
    - `'Alice eats apples.'`
    - `'Bob pets cats.'`
    - `'Carol throws baseballs.'`
    - `'Alice throws Apples.'`
    - `'BOB EATS CATS.'`

    لكنه لن يطابق:
    - `'RoboCop eats apples.'`
    - `'ALICE THROWS FOOTBALLS.'`
    - `'Carol eats 7 cats.'`

7.17 مشاريع للتدريب

لكي تتدرب، اكتب برامج لتنفيذ المهام الآتية.

7.17.1 التحقق من التاريخ

اكتب تعبيرًا نمطيًا يستطيع التعرف على التواريخ بالصيغة DD/MM/YYYY، افترض أن الأيام تتراوح بين 01 إلى 31، والأشهر من 01 إلى 12، والسنوات من 1000 إلى 2999. لاحظ أنه إذا كان اليوم أو الشهر متألف من رقم واحد فيجب أن يسبق بصفر 0.

لا حاجة أن يتعرف التعبير النمطي على الأرقام الصحيحة لكل شهر من أشهر السنة، أو إذا كانت السنوات كبيسة؛ فلا بأس أن يقبل تاريخًا مثل 31/02/2022 أو 31/04/2022.

خزن السلاسل النصية الناتج في متغيرات باسم `month` و `day` و `year`، ثم اكتب شيفرة تتحقق إن كان التاريخ صالحًا، فأشهر أبريل ويونيو وسبتمبر ونوفمبر لها 30 يومًا، وشهر فبراير له 28 يومًا، وبقية الأشهر 31 يومًا.

يكون طول شهر فبراير 29 يومًا في السنوات الكبيسة، والسنة الكبيرة هي كل سنة تكون قابلة للقسمة على 4، عدا السنوات القابلة للقسمة على 100؛ ما لم تكن السنة قابلة للقسم على 400. لاحظ أن هذه العمليات الحسابية تجعل من المستحيل كتابة تعبير نمطي يمكنه فعل كل ذلك.

7.17.2 التحقق من كلمة مرور قوية

اكتب دالةً تستخدم التعابير النمطية للتأكد أن كلمة المرور الممررة إليها قوية. يمكننا تعريف كلمة المرور القوية أنها بطول 8 محارف على الأقل، وتحتوي على أحرف كبيرة وصغيرة، وفيها رقم واحد على الأقل.

قد تحتاج إلى اختبار كلمة المرور على أكثر من تعبير نمطي للتأكد من قوتها.

7.17.3 نسخة من الدالة ()strip باستخدام التعابير النمطية

اكتب دالةً تقبل سلسلةً نصيةً وتفعل فيها كما تفعله الدالة `strip()` التي تعلمناها في الفصل السابق.

إذا لم يمرر إليها أي وسائل بخلاف السلسلة النصية، فسنحذف جميع الفراغات من بداية ونهاية السلسلة النصية؛ وإلا فسنحذف المحارف المحددة في الوسيط الثاني الممرر إلى الدالة.

7.18 الخلاصة

صحيحٌ أن الحواسيب قادرة على البحث عن النصوص بسرعة، لكن يجب أن نخبرها تحديدًا ما الذي عليها البحث عنه. تسمح لنا التعابير النمطية أن نحدد نمطًا من المحارف الذي نريد البحث عنه، وتستخدم التعابير النمطية في البرمجة أو حتى في محررات النصوص أو مع برامج معالجة جداول البيانات.

تأتي الوحدة `re` مع أساس لغة بايثون، وتسمح لنا ببناء كائنات `Regex`، التي تمتلك عددًا من التوابع: التابع `search()` للبحث عن مطابقة واحدة، والتابع `findall()` للعثور على جميع المطابقات، والتابع `sub()` لإجراء عمليات البحث والاستبدال.

ستجد المزيد من المعلومات في توثيق بايثون الرسمي عن التعابير النمطية، وفي موقع regular-expressions.info وفي توثيق موسوعة حسوب.

هل تطمح لبيع منتجاتك الرقمية عبر الإنترنت؟

استثمر مهاراتك التقنية وأطلق منتجًا رقميًا
يحقق لك دخلًا عبر بيعه على متجر بيكاليكا

أطلق منتجك الآن

8. التحقق من المدخلات عبر بايثون

شيفرات التحقق من المدخلات تتأكد أن ما يدخله المستخدم، مثل النصوص الآتية من الدالة `input()`، هي مكتوبة كتابةً صحيحةً؛ فمثلًا حينما نطلب من المستخدمين إدخال أعمارهم، فلا يفترض أن يقبل برنامجك أجوبةً غير منطقية على السؤال، مثل الأرقام السالبة (التي هي خارج المجال المنطقي للأعمار) أو الكلمات (نوع البيانات خطأ).

يمنع التحقق من المدخلات حدوث العلل والمشاكل الأمنية، ففي حال كانت لدينا على سبيل الثال دالة باسم `widthdrawFromAccount()` وستأخذ وسيطًا هو المبلغ الذي يجب اقتطاعه من حساب المستخدم، فيجب علينا أن نحرص على أن يكون المبلغ قيمةً إيجابية، وفي حال إن لم نتحقق من ذلك ومررنا قيمةً سالبة إلى الدالة `widthdrawFromAccount()` وطرحت هذه الدالة القيمة السالبة من حسابنا فسنحصل على أموال بدل سحبها!

من الشائع أن نتحقق من مدخلات المستخدم ونستمر بسؤاله عن مدخلات صحيحة حتى يدخل نصًا صالحًا كما في المثال الآتي:

```
while True:
  print('Enter your age:')
  age = input()
  try:
    age = int(age)
  except:
```

```
    print('Please use numeric digits.')
    continue
  if age < 1:
    print('Please enter a positive number.')
    continue
  Break

print(f'Your age is {age}.')
```

إذا شغلنا هذا البرنامج فسيكون الناتج كما يلي:

```
Enter your age:
five
Please use numeric digits.
Enter your age:
-2
Please enter a positive number.
Enter your age:
30
Your age is 30.
```

إذا شغلت الشيفرة السابقة، فستُسأل عن عمرك حتى تدخل رقمًا صالحًا، وهذا يضمن أن قيمة المتغير age ستكون صالحة حينما ينتهي تنفيذ حلقة `while`، ولن تسبب قيمة هذا المتغير بخطأ لاحقًا في البرنامج.

لكن كتابة شيفرات للتحقق لكل استدعاء للدالة `input()` في برنامجك هو أمر ممل وصعب، ومن المرجح أنك ستغفل عن بعض الحالت مما يؤدي إلى مرور مدخلات خطأ إلى برنامجك. لذا سنتعلم كيف نستعمل الوحدة `PyInputPlus` في هذا الفصل للتحقق من المدخلات.

8.1 الوحدة PyInputPlus

تحتوي الوحدة `PyInputPlus` على عدد من الدوال الشبيه بالدالة `input()` لعدد من أنواع البيانات: الأرقام، والتواريخ، وعناوين البريد الإلكتروني، والمزيد.

إذا أدخل المستخدم قيمةً غير صالحة، كتاريخ فيه خطأ في الصياغة أو رقم خارج المجال المحدد، فستعيد الوحدة `PyInputPlus` طلب المدخلات من المستخدم كما في الشيفرة السابقة. وتمتلك أيضًا بعض الميزات

الأخرى المفيدة مثل وضع حد لعدد المرات التي سيعاد سؤال المستخدم فيها، ووضع زمن انتظار يجب أن يدخل المستخدم فيه المدخلات.

الوحدة `PyInputPlus` ليست جزءًا من المكتبة القياسية في بايثون، لذا يجب عليك تثبيتها بشكل منفصل عبر استخدام مدير الحزم pip؛ وذلك بتشغيل الأمر `pip install --user pyinputplus` من سطر الأوامر. يشرح الفصل الأول من هذا الكتاب بالتفصيل خطوات تثبيت الوحدات الخارجية. للتحقق من سلامة تثبيت الوحدة `PyInputPlus` يمكننا تجربة استيرادها في الطرفية التفاعلية:

```
>>> import pyinputplus
```

إذا لم تظهر أي أخطاء حين استيراد الوحدة، فهذا يعني أنها مثبتة تثبيتًا صحيحًا.

تمتلك الوحدة `PyInputPlus` عددًا من الدوال للتعامل مع مختلف أنواع المدخلات:

- `inputStr()` تشبه الدالة `input()` لكنها تمتلك ميزات وحدة `PyInputPlus` العامة، كما أن بإمكاننا تمرير دالة تحقق مخصصة.
- `inputNum()` تتحقق أن المستخدم يدخل رقمًا، وتعيد عددًا صحيحًا `int` أو عشريًا `float`، اعتمادًا إذا كان الرقم فيه فاصلة عشرية أم لا.
- `inputChoice()` تتحقق أن المستخدم اختار أحد الخيارات الموفرة له.
- `inputMenu()` شبيهة بالدالة `inputChoice()` لكنها توفر قائمة مع خيارات لها أرقام أو أحرف.
- `inputDatetime()` تتحقق أن المستخدم أدخل تاريخًا ووقتًا.
- `inputYesNo()` تتحقق أن المستخدم أدخل yes أو no.
- `inputBool()` تشبه الدالة `inputYesNo()` لكنها تختلف في كونها تقبل القيمة `True` أو `False` وتعيد قيمة منطقية.
- `inputEmail()` تتحقق أن المستخدم أدخل بريدًا إلكترونيًا صالحًا.
- `inputFilepath()` تتأكد أن المستخدم أدخل مسارًا صالحًا لأحد الملفات، ويمكن أن تتحقق اختياريًا أن هنالك ملف بهذا الاسم.
- `inputPassword()` كما في الدالة المبنية في بايثون `input()` لكنها تظهر نجومًا `*` بدل إظهار مدخلات المستخدم لكي لا تظهر المدخلات الحساسة مثل كلمات المرور على الشاشة.

ستعيد هذه الدوال طلب إدخال مدخلات صالحة من المستخدم في حال أدخل قيمةً خطأ:

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum()
five
'five' is not a number.
42
>>> response
42
```

كتابة `as pyip` في عبارة `import` توفر علينا وقتًا في كتابة `pyinputplus` في كل مرة نريد استدعاء دالة من الوحدة `PyInputPlus`، وسنكتب بدلًا منها `pyip`.

إذا نظرت إلى المثال السابق فستجد أن الدوال تعيد القيمة `int` أو `float` على عكس `input()` التي تعيد سلاسل نصية مثل `'42'`.

وكما كنا نمرر سلسلة نصية إلى `input()` لاستعمالها كمحث prompt، فيمكننا تمرير سلسلة نصية إلى دوال `PyInputPlus` عبر استخدام الوسائط ذات الكلمات المفتاحية Keyword arguments، واستخدام الوسيط prompt لعرض المحث:

```
>>> response = input('Enter a number: ')
Enter a number: 42
>>> response
'42'
>>> import pyinputplus as pyip
>>> response = pyip.inputInt(prompt='Enter a number: ')
Enter a number: cat
'cat' is not an integer.
Enter a number: 42
>>> response
42
```

يمكنك استخدام الدالة `help()` في بايثون لتعرف المزيد من المعلومات حول أي دالة من تلك الدوال، فمثلًا كتابة `help(pyip.inputChoice)` سيعرض معلومات حول الدالة `inputChoice()`. يمكنك أن تقرأ التوثيق كاملًا عبر https://pyinputplus.readthedocs.io. على خلاف الدالة المبنية في بايثون `input()`، تمتلك دوال الوحدة `PyInputPlus` عددًا من الميزات الإضافية للتحقق من المدخلات، كما سنشرح في القسم التالي.

8.1.1 وسائط ذات الكلمات المفتاحية min و max و greaterThan و lessThan

تمتلك الدوال `inputNum()` و `inputInt()` و `inputFloat()` التي تقبل الأرقام الصحيحة `int` والعشرية `float` الوسائط ذات الكلمات المفتاحية `min` و `max` و `greaterThan` و `lessThan` لتحديد مجال من القيم الصالحة:

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum('Enter num: ', min=4)
Enter num:3
Input must be at minimum 4.
Enter num:4
>>> response
4
>>> response = pyip.inputNum('Enter num: ', greaterThan=4)
Enter num: 4
Input must be greater than 4.
Enter num: 5
>>> response
5
>>> response = pyip.inputNum('>', min=4, lessThan=6)
Enter num: 6
Input must be less than 6.
Enter num: 3
Input must be at minimum 4.
Enter num: 4
>>> response
4
```

هذه الوسائط ذات الكلمات المفتاحية هي اختيارية، لكن إذا ضبطناها فلا يمكن أن تكون مدخلات المستخدم أقل من min أو أكبر من max (لكن يمكن أن تكون المدخلات مساويةً لها)؛ ويجب أن تكون المدخلات أيضًا أكبر من قيمة `greaterThan` وأقل من قيمة `lessThan` (وأيضًا يمكن أن تكون المدخلات مساويةً لها).

8.1.2 الوسيط ذو الكلمة المفتاحية blank

افتراضيًا لا يُسمَح بالقيم الفارغة ما لم يضبط الوسيط `blank` إلى `True`:

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum('Enter num: ')
Enter num:(blank input entered here)
Blank values are not allowed.
Enter num: 42
>>> response
42
>>> response = pyip.inputNum(blank=True)
(blank input entered here)
>>> response
''
```

استخدام `blank=True` إذا أردت أن تجعل مدخلات المستخدم اختيارية.

8.1.3 الوسائط ذات الكلمات المفتاحية limit و timeout و default

ستستمر دوال الوحدة `PyInputPlus` سؤال المستخدم عن مدخلات صالحة للأبد (ما دام البرنامج يعمل بالطبع) افتراضيًا.

لكن إذا أردنا أن تتوقف الدالة عن سؤال المستخدم بعد عدد من المحاولات أو بعد وقتٍ محدد، فيمكننا استخدام الوسيط `limit` و `timeout`.

يمكننا تمرير قيمة إلى الوسيط ذي الكلمة المفتاحية `limit` لتحديد عدد المرات التي ستحاول الدالة فيها الحصول على مدخل صحيح قبل أن تتوقف عن المحاولة، وتمرير رقم إلى الوسيط ذي الكلمة المفتاحية `timeout` سيحدد عدد الثواني التي ستنتظرها الدالة لمدخلات المستخدم قبل أن تتوقف عن المحاولة.

إذا فشل المستخدم بإدخال قيمة صالحة فسيؤدي ذلك إلى رمي الاستثناء `RetryLimitException` أو `TimeoutException` على التوالي. فمثلًا جرب إدخال ما يلي في الطرفية التفاعلية:

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum(limit=2)
blah
'blah' is not a number.
Enter num: number
'number' is not a number.
Traceback (most recent call last):
  --snip--
pyinputplus.RetryLimitException
```

```
>>> response = pyip.inputNum(timeout=10)
(entered after 10 seconds of waiting)
Traceback (most recent call last):
  --snip--
pyinputplus.TimeoutException
```

حين استخدام الوسائل السابقة يمكننا أيضًا استخدام الوسيط ذي الكلمة المفتاحية `default`، مما يجعل الدالة تعيد قيمةً افتراضيةً بدلًا من رمي استثناء:

```
>>> response = pyip.inputNum(limit=2, default='N/A')
hello
'hello' is not a number.
world
'world' is not a number.
>>> response
'N/A'
```

فبدلًا من رمي الاستثناء `RetryLimitException` فستعيد الدالة `inputNum()` السلسلة النصية `'N/A'`.

8.1.4 الوسيط ذو الكلمة المفتاحية allowRegexes و blockRegexes

يمكنك استخدام التعابير النمطية للتحقق إن كانت المدخلات مسموحٌ بها أم لا. فالوسيطان `allowRegexes` و `blockRegexes` يأخذها قائمةً من التعابير النمطية لتحديد إن كانت دالة `PyInputPlus` ستسمح بالمدخلات على أنها مقبولة أو ترفضها.

جرب مثلًا الشيفرة الآتية في الطرفية التفاعلية لكي تقبل الدالة `inputNum()` الأرقام الرومانية بالإضافة إلى الأرقام العادية:

```
>>> import pyinputplus as pyip

>>> response = pyip.inputNum(allowRegexes=[r'(I|V|X|L|C|D|M)+',
r'zero'])
XLII
>>> response
'XLII'
>>> response = pyip.inputNum(allowRegexes=[r'(i|v|x|l|c|d|m)+',
r'zero'])
```

```
xlii
>>> response
'xlii'
```

ما سيؤثر عليه هذا التعبير النمطي هو الأحرف التي ستقبلها الدالة `inputNum()` من المستخدم، فالدالة حاليًا تقبل الأرقام الرومانية ذات الترتيب الخطأ مثل `'XVX'` أو `'MILLI'` لأن التعبير النمطي `r'(I|V|X|L|C|D|M)+'` يقبل هذه القيم.

يمكنك أيضًا تحديد قائمة بالتعابير النمطية التي سترفضها دالة `PyInputPlus` باستخدام الوسيط `blockRegexes`. جرب المثال الآتي بالطرفية التفاعلية لترى أن الدالة `inputNum()` لن تقبل الأرقام الزوجية:

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum(blockRegexes=[r'[02468]$'])
42
This response is invalid.
44
This response is invalid.
43
>>> response
43
```

إذا حددت قيمتين للوسيطين `allowRegexes` و `blockRegexes`، فإن قائمة السماح ستأخذ أولوية على قائمة الرفض.

جرب المثال الآتي الذي يسمح بالكلمتين `'caterpillar'` و `'category'` لكنه يرفض أي مدخلات فيها الكلمة `'cat'`:

```
>>> import pyinputplus as pyip
>>> response = pyip.inputStr(allowRegexes=[r'caterpillar',
'category'],
blockRegexes=[r'cat'])
cat
This response is invalid.
catastrophe
This response is invalid.
category
>>> response
'category'
```

توابع الوحدة PyInputPlus يمكنها أن توفر علينا كتابة شيفرات كثيرة مملة، تذكر أن تعود إلى التوثيق الرسمي للوحدة PyInputPlus لمزيد من المعلومات حول الوسائط التي يمكن تمريرها إلى دوالها.

8.1.5 تمرير دالة تحقق خاصة إلى inputCustom()

يمكننا كتابة دالة خاصة بنا لتجري عملية التحقق، ونمررها إلى الدالة inputCustom(). لنقل مثلًا أننا نريد من المستخدم أن يدخل سلسلةً من الأرقام التي يجب أن يكون مجموعها 10؛ فلن نجد دالةً باسم pyinputplus.inputAddsUpToTen() لكننا نستطيع إنشاء دالة خاصة بنا التي:

- تقبل معاملًا واحدًا هو السلسلة النصية التي أدخلها المستخدم.
- ترمي استثناءً حين وقوع مشكلة في التحقق.
- تعيد None (أو لا تحتوي على عبارة return) إذا أردنا أن تعيد الدالة inputCustom() مدخلات المستخدم كما هي.
- تعيد قيمة ليست None إذا أردنا أن تعيد الدالة inputCustom() سلسلةً نصيةً مختلفةًتمامًا عمّا أدخله المستخدم.
- نمررها كأول وسيط إلى الدالة inputCustom().

سننشِئ في هذا المثال الدالة addsUpToTen() الخاصة بنا ومررناها إلى الدالة inputCustom(). لاحظ أن استدعاء الدالة سوف يكون على الشكل inputCustom(addsUpToTen) وليس على الشكل inputCustom(addsUpToTen()) لأننا نريد تمرير الدالة addsUpToTen() نفسها إلى الدالة inputCustom()، وليس استدعاء الدالة addsUpToTen() وتمرير القيمة المعادة منها.

```
>>> import pyinputplus as pyip
>>> def addsUpToTen(numbers):
...   numbersList = list(numbers)
...   for i, digit in enumerate(numbersList):
...     numbersList[i] = int(digit)
...   if sum(numbersList) != 10:
...     raise Exception('The digits must add up to 10, not %s.' %
(sum(numbersList)))
...   return int(numbers) # إعادة عدد صحيح
...
>>> response = pyip.inputCustom(addsUpToTen) # لا توجد أقواس بعد اسم الدالة
123
```

```
The digits must add up to 10, not 6.
1235
The digits must add up to 10, not 11.
1234
>>> response # inputStr() أعادت رقمًا وليس سلسلةً نصية
1234
>>> response = pyip.inputCustom(addsUpToTen)
hello
invalid literal for int() with base 10: 'h'
55
>>> response
```

الدالة `inputCustom()` تقبل أيضًا بقية ميزات `PyInputPlus` مثـل `blank` و `limit` و `timeout` و `default` و `allowRegexes` و `blockRegexes`.

سنستفيد جدًا من كتابة دالـة التحقـق الخاصـة بنـا إذا كـان من الصـعب أو المسـتحيل كتابـة تعبـير نمطي للتحقق من صحة مدخلات المستخدم، كمثالنا عن إدخال أرقام مجموعها 10.

8.2 مشروع: هل تريد معرفة حكمة اليوم؟

لنستخدم `PyInputPlus` لإنشاء مشروع بسيط يفعل ما يلي:

1. يسأل المستخدم إن كان يريد معرفة حكمة اليوم؟
2. إذا أدخل المستخدم `no` فسينتهي البرنامج بسلام
3. إذا أدخل المستخدم `yes` فسيذهب إلى الخطوة 1.

أجزم أنك لا تريد أن تعرف الحكمة من مثالنا 🙂.

لا نعرف إن كان سيدخل المستخدم أي سلسلة نصية خلاف "`yes`" و "`no`" لذا علينا إجراء عمليـة تحقـق من صـحة المـدخلات، وسـيكون جميلًا أن نسـمح للمسـتخدم بإدخـال "`y`" أو "`n`" بـدلًا من كتابـة كاملـة الكلمـة. تستطيع الدالة `inputYesNo()` فعل ذلـك، وسـتعيد لنـا السلسـلة النصـية `'yes'` أو `'no'` بغض النظـر عن طريقة الإيجاب أو الرفض التي استعملها المستخدم:

```
Want to know how to keep a 'wise' man busy for hours?
sure
'sure' is not a valid yes/no response.
Want to know how to keep a 'wise' man busy for hours?
```

```
yes
Want to know how to keep a 'wise' man busy for hours?
y
Want to know how to keep a 'wise' man busy for hours?
Yes
Want to know how to keep a 'wise' man busy for hours?
YES
Want to know how to keep a 'wise' man busy for hours?
YES!!!!!!
'YES!!!!!!' is not a valid yes/no response.
Want to know how to keep a 'wise' man busy for hours?
TELL ME HOW TO KEEP A WISE MAN BUSY FOR HOURS.
'TELL ME HOW TO KEEP A WISE MAN BUSY FOR HOURS.' is not a valid yes/no
response.
Want to know how to keep a 'wise' man busy for hours?
no
Thank you. Have a nice day.
```

افتح محرر النصوص وسمِّ ملفك باسم `idiot.py` وأدخل ما يلي:

```
import pyinputplus as pyip
```

سنستورد الوحدة `PyInputPlus`، لكننا نريد اختصار اسمها في برنامجنا إلى `pyip` لأنها أقصر من كتابة `pyinputplus` في كل مرة.

```
while True:
  prompt = 'Want to know how to keep a 'wise' man busy for hours?\n'

  response = pyip.inputYesNo(prompt)
```

سندخل في حلقة تكرار لا نهائية لا تنتهي إلا بالخروج من البرنامج أو الوصول إلى عبارة `break`. وسنستخدم الدالة `pyip.inputYesNo()` في هذه الحلقة لقبول مدخلات المستخدم والتحقق أنها `yes` أو `no`.

```
  if response == 'no':

    break
```

الدالة `pyip.inputYesNo()` مصممة لتعيد السلسلة النصية `yes` أو `no`؛ فإذا أعادت `no` فسنخرج من الحلقة اللانهائية ونكمل تنفيذ السطر الأخير من البرنامج الذي يشكر المستخدم على صبره:

```
print('Thank you. Have a nice day.')
```

وإلا فسيستمر تنفيذ حلقة التكرار.

يمكنك إنشاء نسخة من الدالة `inputYesNo()` في اللغات غير الإنكليزية بتمرير قيم للوسيطين ذوي الكلمات المفتاحية `yesVal` و `noVal`. فانظر إلى المثال الآتي باللغة الإسبانية:

```
 prompt = '¿Quieres saber cómo mantener ocupado a un idiota durante
horas?\n'
 response = pyip.inputYesNo(prompt, yesVal='sí', noVal='no')
 if response == 'sí':
```

يمكن للمستخدم الآن إدخال `sí` أو `s` (بأحرف كبيرة أو صغيرة) بدلًا من `yes` أو `y` للإيجاب بالموافقة.

8.3 مشروع: اختبار جدول الضرب

يمكننا استخدام ميزات الوحدة `PyInputPlus` لإنشاء اختبار لجدول الضرب له وقت معين. إذ نستطيع أن نضبط قيم للوسائط ذات الكلمات المفتاحية `allowRegexes` و `blockRegexes` و `timeout` و `limit` للدالة `pyip.inputStr()` ونترك أمر التحقق من صحة مدخلات المستخدم على الوحدة `PyInputPlus`.

تذكر دومًا أنه كلما كتبت شيفرة أقل ستصبح برامجك أسرع في التطوير والتنفيذ.

لننشِئ برنامجًا يعرض 10 أسئلة في جدول الضرب على المستخدم، ولا يجوز أن يدخل المستخدم سوى الأرقام الصحيحة. احفظ الشيفرات الآتية في ملف باسم `multiplicationQuiz.py`.

سنستورد في البداية الوحدات `pyinputplus` و `random` و `time`. وسنتتبع عدد الأسئلة التي يسألها برنامجنا وعدد الإجابات الصحيحة التي يوفرها المستخدم عبر المتغيرين `numberOfQuestions` و`correctAnswers`.

سنسأل المستخدم داخل حلقة `for` لعشر مرات.

```
import pyinputplus as pyip
import random, time

numberOfQuestions = 10
correctAnswers = 0
```

سنختار داخل حلقة for أرقامًا ذات خانة واحدة لضربها مع بضعها، وسننشِئ المحث prompt الذي سنسأل المستخدم فيه عن قيمة ناتج الضرب #Q: N × N = الذي تكون فيه Q هي رقم السؤال (من 1 إلى 10) و N هما الرقمان اللذان سنضربهما ببعضهما:

```
    # اختيار رقمين عشوائيين
    num1 = random.randint(0, 9)
    num2 = random.randint(0, 9)

    prompt = '#%s: %s x %s = ' % (questionNumber, num1, num2)
```

ستتولى الدالة pyip.inputStr() أغلبية خصائص البرنامج، فسنمرر لها المعامل allowRegexes الذي هو قائمة فيها عنصر واحد وهو السلسلة النصية '^%s$'، وسيبدل فيها s% إلى قيمة الجواب الصحيح، وسنستخدم ^ و $ للتحقق أن جواب المستخدم يبدأ وينتهي بالرقم الصحيح، وستحذف المكتبة PyInputPlus أي فراغات بيضاء في بداية ونهاية جواب المستخدم في حال أضاف مسافةً فارغةً خطأً.

القيمة التي سنمررها إلى المعامل blocklistRegexes هي قائمة بها عنصر هو صف قيمته ('.*','Incorrect!').

أول سلسلة نصية في الصف هي التعبير النمطي الذي يطابق أي شيء، بالتالي لو أدخل المستخدم أي جواب لا يطابق الجواب الصحيح فسيرفضه البرنامج وستظهر السلسلة النصية 'Incorrect!' وسنطلب من المستخدم إدخال قيمة مجددًا.

لاحظ أيضًا تمرير القيمة 8 إلى المعامل timeout و 3 إلى المعامل limit لكي نحرص أن المستخدم يملك 8 ثواني لإدخال إجابة و3 محاولات فقط لإدخال الرقم الصحيح.

```
    try:
      pyip.inputStr(prompt, allowRegexes=['^%s$' % (num1 * num2)],
              blockRegexes=[('.*', 'Incorrect!')],
              timeout=8, limit=3)
```

إذا لم يدخل المستخدم الإجابة خلال مهلة 8 ثواني، فسترمي pyip.inputStr() الاستثناء TimeoutException. إذا أدخل المستخدم 3 إجابات خطأ فسيرمى الاستثناء RetryLimitException. لاحظ أنهما جزء من الوحدة PyInputPlus لذا يجب أن نسبقهما بالبادئة pyip..

```
    except pyip.TimeoutException:
      print('Out of time!')
    except pyip.RetryLimitException:
      print('Out of tries!')
```

هل تذكر أن كتلة `else` تأتي بعد كتل `if` أو `elif`؟ يمكنها أيضًا أن تأتي بعد آخر كتلة `except`؛ وستنفذ الشيفرة الموجودة في كتلة `else` في حال عدم رمي أي استثناء في كتلة `try`، أي في حالتنا حين إدخال المستخدم الإجابة الصحيحة.

استخدام `else` في هذا السياق هو أمر اختياري، والسبب الرئيسي لاستخدامها بدلًا من كتابة بقية التعابير البرمجية في كتلة `try` هو تسهيل مقروئية النص.

```
    else:
      # ستنفذ هذه الكتلة في حال عدم رمي أي استثناء
      print('Correct!')
      correctAnswers += 1
```

ومهما كانت الرسالة التي ستظهر للمستخدم من الرسائل الثلاث "Out of time!" أو "Out of tries!" أو "Correct!" فسيمهل المستخدم لمدة 1 ثانية في نهاية حلقة `for` ليقرأها.

بعد سؤال المستخدم 10 مرات عبر حلقة `for` فسنعرض له كم إجابةً صحيحةً قد أجاب:

```
    time.sleep(1) # انتظر برهة ليستطيع المستخدم القراءة
  print('Score: %s / %s' % (correctAnswers, numberOfQuestions))
```

الوحدة `PyInputPlus` مرنة بما يكفي لاستخدامها في مختلف أنواع التطبيقات التي تقبل مدخلات المستخدم من سطر الأوامر كما رأينا في هذا الفصل.

8.4 أسئلة للتدريب

1. هل تأتي الوحدة `PyInputPlus` مع المكتبة القياسية في بايثون؟
2. لماذا نستورد الوحدة `PyInputPlus` عبر `import pyinputplus as pyip`؟
3. ما الفرق بين `inputInt()` و `inputFloat()`؟

4. كيف تستطيع التحقق أن المستخدم قد أدخل رقمًا بين 0 و 99 عبر `PyInputPlus`؟
5. ما الذي نمرره إلى المعاملين `allowRegexes` و `blockRegexes`؟
6. ما نتيجة `inputStr(limit=3)` إن كانت المدخلات فارغة 3 مرات؟
7. ما نتيجة `inputStr(limit=3, default='hello')` إن كانت المدخلات فارغة 3 مرات؟

8.5 مشاريع للتدريب

لكي تتدرب، اكتب برامج لتنفيذ المهام الآتية.

8.5.1 صانع الصندويشات

اكتب برنامجًا يسأل المستخدم عما يفضل وضعه في الصندويشة التي طلبها. استعمل الوحدة `PyInputPlus` للتأكد من صحة مدخلات المستخدم كما يلي:

- استخدم `inputMenu()` من أجل تحديد نوع الخبز: دقيق كامل wheat، أو خبز أبيض white، أو خبز مختمر sourdough.
- استخدم `inputMenu()` لنوع البروتين: دجاج chicken، أو ديك turkey، أو بقر beef، أو توفو (جبن نباتي من حليب الصويا) tofu.
- استخدم `inputYesNo()` لتسأله إذا كان يريد جبنة cheese أم لا.
- إذا كان يريد جبنة فاستخدم `inputMenu()` لتسأله عن نوعها: شيدر cheddar أو موزاريلا mozzarella أو جبنة سويسرية swiss.
- استخدم `inputYesNo()` لسؤاله إن كان يريد مايونيز mayo وخردل mustard وخس lettuce وطماطم tomato.
- استخدم `inputInt()` لتسأله كم صندويشة يريد، احرص أن يكون العدد أكبر من 1.

حدد من عندك أسعارًا للخيارات السابقة، واعرض للمستخدم السعر النهائي في نهاية البرنامج.

8.5.2 أعد كتابة اختبار جدول الضرب

لترى كم تسهل عليك الوحدة `PyInputPlus` عملية التحقق من المدخلات، حاول إعادة إنشاء اختبار جدول الضرب دون استخدامها؛ أي اكتب برنامجًا يسأل المستخدم 10 أسئلة حول جدول الضرب من 0 × 0 حتى 9 × 9. ستحتاج إلى برمجة الميزات الآتية:

- إذا أدخل المستخدم الإجابة الصحيحة فسيعرض لـه البرنـامج القيمـة "Correct!" لثانيـة ثم ينتقـل إلى السؤال الذي يليه.
- لدى المستخدم ثلاث محاولات قبل أن ينتقل البرنامج إلى السؤال التالي.
- إذا مرت 8 ثواني بعد عرض السؤال لأول مرة ولم يدخل المستخدم إجابة صحيحةً فسينتقل البرنامج إلى السؤال التالي دون احتساب إجابة السؤال.

قارن بين الشيفرة التي كتبتها وبين الشيفرة في قسم «مشروع: اختبار جدول الضرب».

8.6 الخلاصة

من السهل أن ننسى كتابة شيفرات التحقق من مدخلات المستخدم، لكن دونها سنجد ظهور مختلف العلل البرمجية في برامجنا بسبب اختلاف القيم التي نتوقـع أن يـدخلها المسـتخدم عن القيم الـتي القيم الـتي يمكنـه إدخالهـا؛ لـذا يجب أن تكـون برامجنـا مرنـةً بمـا يكفي للتعامـل مـع هـذه الحـالات الاسـتثنائية. يمكننا استخدام التعابير النمطية لإنشاء الشيفرات اللازمـة للتحقـق من المـدخلات، أو اسـتخدام مكتبـة خارجيـة جاهزة مثل `PyInputPlus` باستيرادها عبر `import pyinputplus as pyip`، وستستطيع استخدام اسم مختصر لها أيضًا.

تمتلك الوحدة `PyInputPlus` دوال مختلفة لأنواع متنوعة من المدخلات، بما في ذلك السلاسـل النصـية والأرقام والتواريخ، وأسئلة الإيجاب بالموافقة أو الـرفض، وأسـئلة `True` أو `False`، وعنـاوين البريـد الإلكـتروني، والملفات.

وفي حين أن الدالة `input()` المضمنة في بايثون تعيد سلسلةً نصيةً دومًا، لكن هذه الدوال تعيـد القيمـة مع نوع البيانات المناسب لها. تسمح لنا الدالة `inputChoice()` باختصـار أحـد الخيـارات الموجـودة مسـبقًا، بينما توفر `inputMenu()` قائمة مع خيارات لها أرقام أو أحرف لتسهيل الاختيار.

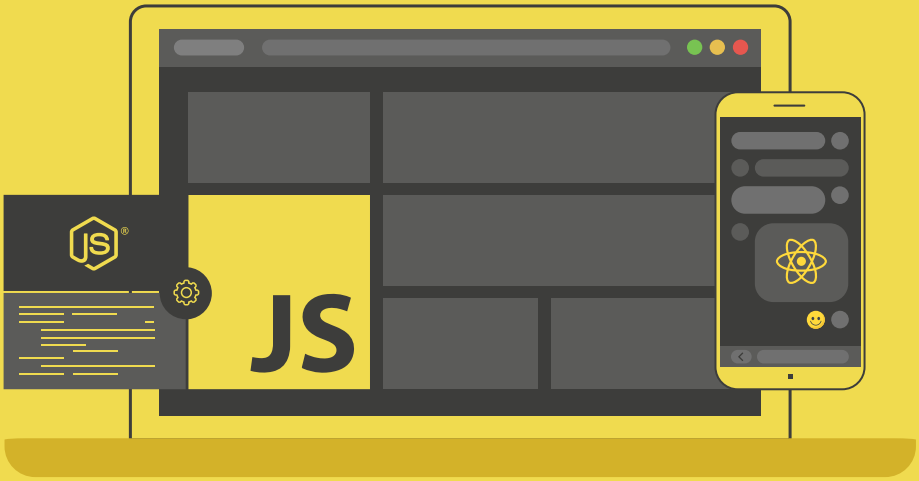
لكل تلك الدوال ميزات قياسـية: فهي تزيـل الفراغـات من بدايـة ونهايـة المـدخلات، وتضـبط مهلـةً وعـددًا لمحاولات الإدخال عبر المعاملين `timeout` و `limit`، ونستطيع تمرير قائمة فيها سلاسل نصـية تمثـل تعـابير نمطية إلى المعـاملين `allowRegexes` أو `blockRegexes` لتضـمين أو اسـتبعاد تعـابير نمطيـة محـددة من إجابـات المسـتخدم. وعلا ذلـك سـتوفر عليـك وقتـك الـذي ستقضـيه في كتابـة حلقـات `while` لإعـادة طلب المدخلات من المستخدم.

إذا لم تكن إحدى دوال الوحـدة `PyInputPlus` مناسـبةً لاحتياجاتـك، لكنـك مـا تـزال تريـد الاسـتفادة من الطيـف الواسـع من ميزاتهـا، فيمكنـك اسـتدعاء الدالـة `inputCustom()` وتمريـر دالـة تحقـق خاصـة بـك لاستخدامها. لا تنسَ العودة إلى توثيق الوحدة `PyInputPlus` لأي ميزات أو أمور ترغب في أن تستوضحها ولم تكن واضحةً لك في هذا الفصل.

لا حاجة أن تعيد اختراع العجلة كل مرة، من المهم أن تتعلم كيفية استخدام الوحدات والمكتبات الـتي كتبهـا غيرك بدلًا من إضافة الوقت في إعادة برمجة نفس الميزات.

بعد أن أصبحت لدينا المعرفة اللازمة في معالجة النصوص والتحقق من مدخلات المستخدم، لنتعلم كيفيـة القراءة والكتابة من نظام الملفات في حاسوبك.

دورة تطوير التطبيقات
باستخدام لغة JavaScript



مميزات الدورة

- بناء معرض أعمال قوي بمشاريع حقيقية
- وصول مدى الحياة لمحتويات الدورة
- تحديثات مستمرة على الدورة مجانًا
- شهادة معتمدة من أكاديمية حسوب
- إرشادات من المدربين على مدار الساعة
- من الصفر دون الحاجة لخبرة مسبقة

اشترك الآن

9. قراءة وكتابة الملفات باستخدام بايثون

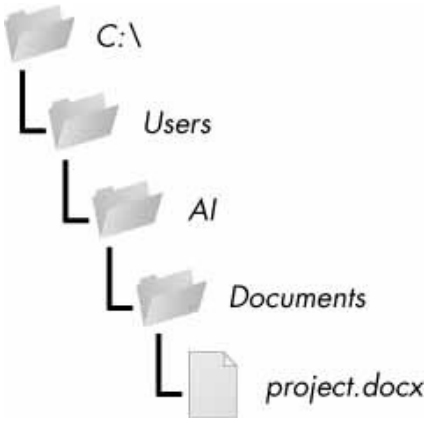
يمكننا تخزين المعلومات في المتغيرات في برنامجنا وستبقى موجودة طالما استمر تشغيل البرنـامج، لكنـا ماذا لو أردنا الحفاظ على البيانات بعد انتهاء تنفيذ البرنامج؟ سـنحتاج إلى حفظهـا إلى ملـف؛ وسـنتعلم في هـذا الفصل كيفية استخدام بايثون لإنشاء وقراءة وكتابة الملفات في حاسوبنا.

9.1 الملفات ومساراتها

يملك كل ملف خاصيتان أساسيتان: اسم الملف ومساره. يحـدد مسـار الملـف أين سـيظهر في حاسـوبك، فمثلًا هنالـك ملـف في حاسـوبي الـذي يعمـل بنظـام وينـدوز موجـود اسـمه project.docx في المسـار `C:\Users\Al\Documents`.

الجزء الذي يلي اسم الملف ويأتي بعد النقطة يسمى بامتداد الملـف `file extension` ويخبرنـا مـا هـو نـوع الملـف. فعلى سـبيل المثـال، الملـف `project.docx` هـو مسـتند وورد؛ بينمـا تشـير Users و Al وDocuments إلى مجلدات.

يمكن أن تحتوي المجلدات على ملفات ومجلدات أخرى، فمثلًا الملف `project.docx` موجود في المجلد Documents الذي بدوره موجود في المجلد Al الموجود في المجلد `Users`. يوضح الشكل التالي هذه البنية.



الشكل 36: ملف موجود في مجلد

الجزء `C:\` من المسار يسمى بالمجلد الجذر root، أي يحتوي على جميع المجلدات الأخرى. وهو المجلد `C:` في نظام ويندوز أو يسمى القرص `C:`؛ أما في ماك أو لينكس فيكون المجلد الجذر هو `/`. سنستعمل في هذه السلسلة نمط مسارات ويندوز لأنها أكثر شيوعًا، لكن إن كنت تستعمل ماك أو لينكس فاستعمل بنية المسارات المناسبة لنظامك.

تظهر أجهزة التخزين الأخرى مثل أقراص DVD أو وسائط تخزين USB بطرائق مختلف حسب نظام التشغيل، ففي ويندوز ستظهر على شكل قرص جديد له حرف مختلف مثل `D:\` أو `E:\`. بينما في ماك فستظهر كمجلد جديد في المجلد `/Volumes`، وفي لينكس ستظهر كمجلدات جديدة في المجلد `/mnt` (أو `/media` حسب التوزيعة عندك).

من المهم أن تلاحظ أن أسماء الملفات والمجلدات غير حساسة لحالة الأحرف في ويندوز وماك، لكنها حساسة لحالة الأحرف في لينكس.

> **ملاحظة**: من المؤكد أن بنية المجلدات وأسماء الملفات في حاسوبك ونظام تشغيلك تختلف عمّا هو عندي، لذا لن تستطيع اتباع أمثلة هذه السلسلة حرفيًا. لكن جرب المتابعة على الملفات والمجلدات الموجودة عندك.

9.1.1 الخطين المائل الخلفي \ و المائل الأمامي / في أنظمة التشغيل

تستعمل مسارات الملفات في أنظمة ويندوز الخط المائل الخلفي `\` الذي يفصل بين أسماء المجلدات؛ أما في ماك ولينكس فيستعمل الخط المائل الأمامي `/` فاصلًا بين المجلدات؛ ولو أردت أن تعمل برامجك التي تكتبها على جميع الأنظمة (وهذا أمر مهم أنصحك به) فعليك أن تتعامل مع كلا الحالتين.

لحسن الحظ هنالك دالة في الوحدة `pathlib` باسم `Path()`، التي نمرر إليها سلاسل نصية بأسماء المجلدات والملف المطلوب، وستعيد الدالة `Path()` سلسلةً نصيةً لمسار الملف يستعمل الفاصل الصحيح بين المجلدات وفقًا لنظام التشغيل:

```
>>> from pathlib import Path
>>> Path('spam', 'olive', 'eggs')

WindowsPath('spam/olive/eggs')
>>> str(Path('spam', 'olive', 'eggs'))
'spam\\olive\\eggs'
```

لاحظ أن من الشائع حين استيراد `pathlib` أن نكتب `from pathlib import Path` وإلا فسنحتاج إلى كتابة `pathlib.Path` في كل مرة نريد استعمال `Path` فيها.

سأشغل أمثلة هذا الفصل على نظام ويندوز، لذا ستعيد الدالة `Path('spam', 'olive', 'eggs')` الكائن `WindowsPath` للمسار النهائي `WindowsPath('spam/olive/eggs')`؛ وصحيحٌ أن ويندوز يستعمل الخط المائل الخلفي في المسارات، لكن تمثيل الكائن `WindowsPath` في الطرفية التفاعلية يستعمل الخط المائل الأمامي، هذا لأن مطوري البرمجيات مفتوحة المصدر يفضلون نظام لينكس ويستعملون العادات الخاصة به أثناء التطوير.

إذا أردنا الحصول على سلسلة نصية من المسار، فيمكننا تمرير الكائن `WindowsPath` إلى الدالة `str()` التي ستعيد في مثالنا السلسلة النصية `'spam\\olive\\eggs'`، لاحظ أن الخطوط المائلة الخلفية مضاعفة لأن كل خط مائل خلفي يحتاج إلى خطٍ مائل خلفي آخر لتهريبه. إذا استخدمنا الدالة السابقة في نظام لينكس فستعيد كائن `PosixPath`، الذي حين تمريره إلى الدالة `str()` فسيعيد السلسلة النصية `'spam/olive/eggs'` (كلمة `POSIX` تشير إلى مجموعة من المعايير الحاكمة للأنظمة الشبيهة بيونكس `Unix-like` مثل لينكس. إذا كنت لم تجرب لينكس بعد فأنصحك وبشدة أن تجربه).

يمكن تمرير كائنات `Path` (سواءً كانت `WindowsPath` أو `PosixPath` اعتمادًا على نظام تشغيلك) إلى دوال أخرى متعلقة بالتعامل مع الملفات والتي سنشرحها خلال هذا الفصل. لمثال الآتي يولد مجموعة من مسارات الملفات في أحد المجلدات:

```
>>> from pathlib import Path
>>> myFiles = ['accounts.txt', 'details.csv', 'invite.docx']
>>> for filename in myFiles:
    print(Path(r'C:\Users\Al', filename))

C:\Users\Al\accounts.txt
C:\Users\Al\details.csv
C:\Users\Al\invite.docx
```

يفصل الخط المائل الخلفي بين المجلدات في ويندوز، لذا لا يمكنك استخدامه في أسماء الملفـات، لكنـك تستطيع استخدام الخطوط المائلة الخلفية `\` في ماك ولينكس، فبينما يشير المسار `Path(r'spam\eggs')` إلى مجلـدين مختلفين (أو الملـف `eggs` في المجلـد `spam`) في ويـنـدوز، لكنـه يشـير إلى مجلـد أو ملـف باسـم `spam\eggs` في ماك ولينكس. لهذا السبب من المستحسن استخدام الخطوط المائلة الأمامية `/` في شيفرات بايثون دومًا، وسنفعل المثل في أمثلة الفصل، وستضمن لنا الوحدة `pathlib` أن المسـارات الـتي نسـتخدمها تعمل على جميع أنظمة التشغيل.

لاحظ أن الوحدة `pathlib` جديدة في بايثون 3.4 وأتت لتستبدل دوال `os.path` القديمة؛ وتـدعمها دوال المكتبـة القياسـية في بـايثون بـدءًا من الإصـدار 3.6. إذا كنت تعمـل مـع سـكربتات مكتوبـة بإصـدار بـايثون 2 فأنصحك أن تستعمل الوحدة `pathlib2` التي تـوفر إمكانيـة `pathlib` في بـايثون 2.7. يشـرح الفصـل الأول خطوات تثبيت `pathlib2` باستخدام `pip`.

سأوضح أي اختلافات واسـتخدامات للوحـدة os حين الحاجـة، فقـد تسـتفيد منهـا حين قـراءة السـكربتات القديمة أو التي كتبها غيرك.

9.1.2 استخدام العامل / لجمع المسارات

نستخدم العامل `+` عادةً لجمع عددين كما في التعبير `2 + 2`، الذي ينتج القيمة العددية `4`، لكن يمكننا أيضًا استخدام العامل `+` لجمع سلسـلتين نصـيتين كمـا في التعبـير `'Hello' + 'World'` الـذي ينتج السلسـلة النصية `'HelloWorld'`. وبشكل مشابه يستعمل العامل `/` للقسـمة لكن يمكنـه أيضًـا أن يجمـع بين كائنـات `Path` والسلاسل النصية، وهو يفيد في التعامل مع كائنات `Path` التي أنشأناها سابقًا عبر الدالة `Path()`:

```
>>> from pathlib import Path
>>> Path('spam') / 'olive' / 'eggs'
WindowsPath('spam/olive/eggs')
>>> Path('spam') / Path('olive/eggs')
WindowsPath('spam/olive/eggs')
>>> Path('spam') / Path('olive', 'eggs')
WindowsPath('spam/olive/eggs')
```

يسهل استخدام العامل `/` مع كائنات `Path` عملية جمع المسارات مع بعضها كما لو كـانت سلاسـل نصـية بسيطة، واستخدامه يُعَد أكثر أمانًا من إجـراء عمليـة جمـع للسلاسـل النصـية يـدويًا أو عـبر التـابع `join()` كمـا بالمثال الآتي:

```
>>> homeFolder = r'C:\Users\Al'
>>> subFolder = 'spam'
>>> homeFolder + '\\' + subFolder
```

```
'C:\\Users\\Al\\spam'
>>> '\\'.join([homeFolder, subFolder])
'C:\\Users\\Al\\spam'
```

الشيفرة السابقة ليست آمنة لأن الخطوط المائلة الخلفية لا تعمل إلا على ويندوز كما ناقشنا في الأقسام السابقة. يمكنك أن تضيف عبارةً شرطيةً `if` للتحقق من `sys.platform` (الذي يحتوي على سلسلة نصية تصف نظام التشغيل المستعمل) لتحديد ما هو نوع الخط المائل الذي نريد استخدامه. لكن استخدام هذه الشيفرة في كل مكان تريد التعامل مع مسارات الملفات فيه هو أمر متعب وغير متناسق ومن المرجح أن يسبب علل برمجية.

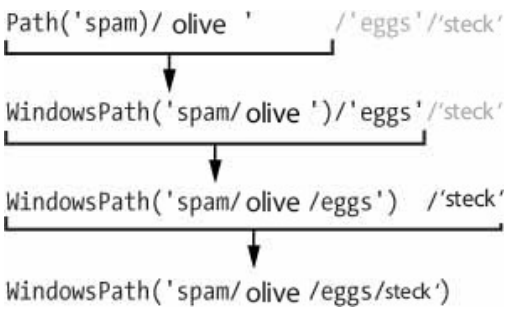
تحل الوحدة `pathlib` هذه المشاكل بإعادة استخدام معامل القسمة `/` ليجمع بين المسارات دون مشاكل بغض النظر عن نظام التشغيل المستعمل. يوضح المثال الآتي آلية استخدامه:

```
>>> homeFolder = Path('C:/Users/Al')
>>> subFolder = Path('spam')
>>> homeFolder / subFolder
WindowsPath('C:/Users/Al/spam')
>>> str(homeFolder / subFolder)
'C:\\Users\\Al\\spam'
```

أمر واحد مهم يجب أن نبقيه في ذهننا أثناء استخدام العامل `/` لجمع المسارات هو أن إحدى أول قيمتين من المسارات التي يجب جمعها يجب أن تكون كائن `Path`، وإلا فستظهر لك بايثون رسالة خطأ:

```
>>> 'spam' / 'olive' / 'eggs'
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

تقيّم بايثون التعبير البرمجي الذي يستعمل العامل `/` من اليسار إلى اليمين إلى كائن `Path`، لهذا يجب أن يكون أول أو ثاني قيمة من اليسار من النوع `Path` لكي تستطيع إنتاج كائن `Path` من التعبير البرمجي. هذه هي آلية العمل التي تتبعها بايثون للحصول على كائن `Path` النهائي:



الشكل 37: آلية تفسير المسارات مع العامل /

إذا ظهرت رسالة الخطأ التالية فهذا يعني أنه يجب علينا أن نضع الكائن `Path` في الطرف الأيسر من التعبير البرمجي:

```
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

يستبدل العامل / الدالةَ `os.path.join()` التي يمكنك معرفة المزيد عنها من التوثيق الرسمي https://docs.python.org/3/library/os.path.html#os.path.join.

9.1.3 مجلد العمل الحالي

يملك كل برنامج تشغله على حاسوب ما يسمى «مجلد العمل الحالي current working directory» أو اختصارًا cwd؛ تفترض بايثون أن أي ملفات أو مسارات لا تبدأ بالمجلد الجذر هي موجودة في مجلد العمل الحالي.

> ملاحظة: صحيح أننا نقول «مجلد» ترجمةً لكلمة directory التي تعني «موجِّه»، لكنها شائعة بين المستخدمين العرب أكثر؛ ولا أحد يقول current working folder. أغلبية الاصطلاحات البرمجية تستعمل directory بدلًا من folder، لذا ستجد هذه الكلمة مستعملةً في أسماء الدوال المشروحة تاليًا.

يمكننا الحصول على سلسلة نصية تمثل مجلد العمل الحالي باستخدام `Path.cwd()`، ويمكن تغييرها باستخدام `os.chdir()`:

```
>>> from pathlib import Path
>>> import os
>>> Path.cwd()
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python37')'
>>> os.chdir('C:\\Windows\\System32')
>>> Path.cwd()
WindowsPath('C:/Windows/System32')
```

لاحظ أن مجلد العمل الحالي هو:

```
C:\Users\Al\AppData\Local\Programs\Python\Python37
```

لذا إذا استعملنا اسم الملف `project.docx` فإنه سيشير إلى المسار:

```
C:\Users\Al\AppData\Local\Programs\Python\Python37\project.docx
```

حينما بدلنا مجلد العمل الحالي إلى `C:\Windows\System32` فسيفسر `project.docx` إلى المسار:

```
C:\Windows\System32\project.docx
```

ستظهر بايثون رسالة خطأ حين محاولة تغيير مجلد العمل الحالي إلى مجلد غير موجود:

```
>>> os.chdir('C:/ThisFolderDoesNotExist')
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
FileNotFoundError: [WinError 2] The system cannot find the file
specified:
'C:/ThisFolderDoesNotExist'
```

لا توجد دالة في `pathlib` لتغيير مجلد العمل الحالي، لأن تغييره أثناء تشغيل البرنامج قد يسبب علل برمجية نحن في غنى عنها.

الدالة `os.getcwd()` هي الطريقة القديمة للحصول على سلسلة نصية تمثل مجلد العمل الحالي.

9.1.4 مجلد المنزل

يمتلك جميع المستخدمون مجلدًا خاصًا بهم يسمى مجلد المنزل home directory، ويمكننا الحصول على كائن `Path` لمجلد المنزل باستدعاء `Path.home()`:

```
>>> Path.home()
WindowsPath('C:/Users/Al')
```

توجد مجلدات المنزل عادةً في مكان محدد يختلف حسب نظام تشغيلك:

- في ويندوز تكون في `C:\Users`.
- في ماك تكون في `/Users`.
- في لينكس تكون في `/home`.

من شبه المؤكد أن سكربتات بايثون التي تكتبها ستمتلك أذونات القراءة والكتابة في مجلد المنزل، لـذا من المستحسن أن تضع الملفات التي ستعالجها عبر بايثون فيه.

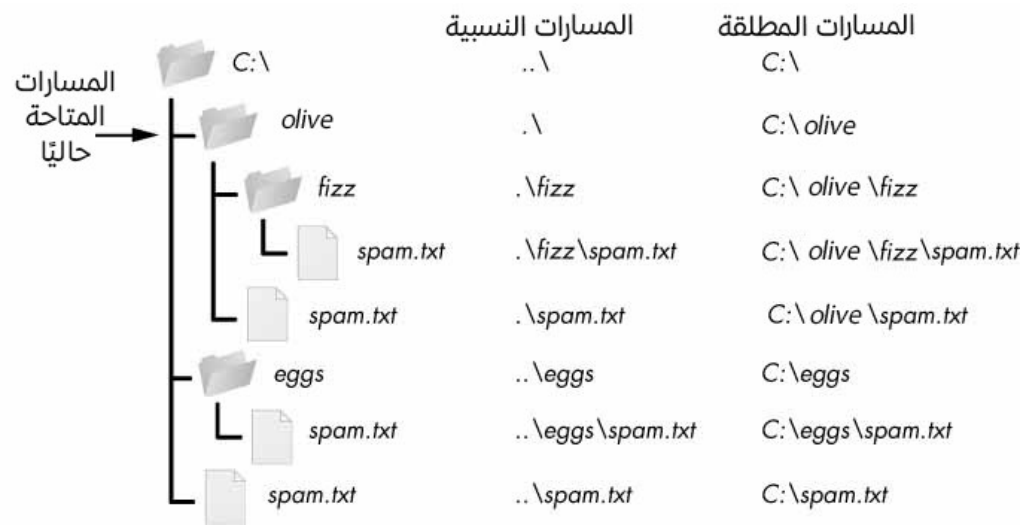
9.1.5 المسارات النسبية والمسارات المطلقة

هنالك طريقتان لتحديد مسار ملف:

- مسار مطلق absolute path، الذي يبدأ من المجلد الجذر.
- مسار نسبي relative path، الذي يبدأ من مجلد العمل الحالي للبرنامج.

هنالك النقطة . والنقطتـان .. حين التعامـل مـع المسـارات، وهي ليسـت مجلـدات حقيقـة ولكنهـا أسـماء خاصة، إذا تمثل النقطة . المجلد الحالي، بينما النقطتان .. تمثل المجلد الأب.

يوضح الآتي بعض الملفات والمجلدات ويكون مجلد العمـل الحـالي فيـه هـو `C:\olive`، وفيـه مسـارات الملفات والمجلدات كلها المطلقة والنسبية.



الشكل 38: المسارات المطلقة والنسبية

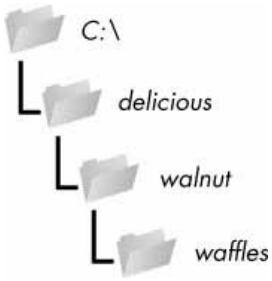
لاحظ أن .\ في بداية المسارات النسبية اختيارية، إذ يشير `.\spam.txt` و `spam.txt` إلى نفس الملف.

9.1.6 إنشاء مجلدات جديدة باستخدام الدالة os.makedires()

يمكن لبرامجك إنشاء مجلدات جديدة باستخدام الدالة `os.makedires()`:

```
>>> import os
>>> os.makedirs('C:\\delicious\\walnut\\waffles')
```

سينتشئ المثال السابق المجلد `C:\delicious` وبداخله المجلد `walnut` وبداخله المجلد `waffles`. فالدالة `os.makedires()` ستنشِئ أي مجلدات لازمة غير موجودة مسبقًا.



الشكل 39: ناتج تنفيذ
دالة os.makedirs()

لإنشاء مجلد من كائن `Path` فيمكننا استدعاء التابع `mkdir()`، فمثلًا سأنشِئ المجلد `spam` في مجلد المنزل في حاسوبي:

```
>>> from pathlib import Path
>>> Path(r'C:\Users\Al\spam').mkdir()
```

لاحظ أن التابع `mkdir()` يستطيع إنشاء مجلد واحد فقط، ولن ينشِئ مجلدات فرعية كما هو الحال في الدالة `os.makedirs()`.

9.1.7 التعامل مع المسارات النسبية والمطلقة

توفر الوحدة `pathlib` توابع للتحقق إن كان أحد المسارات نسبيًا أو مطلقًا، وتستطيع إعادة المسار المطلق من مسارٍ نسبي.

سيعيد استدعاء التابع `is_absolute()` على كائن `Path` القيمة `True` إذا كان يمثل مسارًا مطلقًا أو `False` إذا كان يمثل مسارًا نسبيًا. تذكر أن تستعمل مسارات موجودة في حاسوبك في الأمثلة القادمة:

```
>>> Path.cwd()
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python37')
>>> Path.cwd().is_absolute()
True
>>> Path('spam/olive/eggs').is_absolute()
False
```

للحصول على مسار مطلق من مسار نسبي يمكننا وضع `Path.cwd() /` قبل كائن `Path`، فحينما نقول «مسار نسبي» فهذا يعني أن المسار منسوب إلى مجلد العمل الحالي:

```
>>> Path('my/relative/path')
WindowsPath('my/relative/path')
>>> Path.cwd() / Path('my/relative/path')
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python37/my/
relative/
path')
```

أمـا إذا كـان المسـار النسـبي منسـوب إلى مسـار مختلـف عن مجلـد العمـل الحـالي، فعلينـا تبـديل `Path.cwd()` إلى ذاك المسار، ففي المثال الآتي سنحصل على المسار النسبي نسبةً إلى مجلد المنزل الخاص بنا بدلًا من مجلد العمل الحالي:

```
>>> Path('my/relative/path')
WindowsPath('my/relative/path')
>>> Path.home() / Path('my/relative/path')
WindowsPath('C:/Users/Al/my/relative/path')
```

تمتلك الوحدة `os.path` عددًا من الدوال المفيدة المتعلقة بالمسارات النسبية والمطلقة:

- ستعيد `os.path.abspath(path)` سلسلةً نصية فيها المسار المطلق للوسيط الممرر إليها، وهـذه أسهل طريقة لتحويل مسار نسبي إلى مسار مطلق.
- ستعيد `os.path.isabs(path)` القيمة `True` إن كان الوسـيط الممـرر مسـارًا مطلقًـا و `False` إذا كان مسارًا نسبيًا.
- ستعيد `os.path.relpath(path, start)` سلسلةً نصيةً للمسار النسبي للمسار `path` بدءًا من المسار `start`. إذا لم نوفر المعامل `start` فسيستعمل مجلد العمل الحالي بدلًا منه.

لنجرب الدوال السابقة:

```
>>> os.path.abspath('.')

'C:\\Users\\Al\\AppData\\Local\\Programs\\Python\\Python37'
>>> os.path.abspath('.\\Scripts')
'C:\\Users\\Al\\AppData\\Local\\Programs\\Python\\Python37\\Scripts'
>>> os.path.isabs('.')
False
>>> os.path.isabs(os.path.abspath('.'))
True
```

ولمّا كان المسار:

```
C:\Users\Al\AppData\Local\Programs\Python\Python37
```

هو مجلد العمل الحالي حين استدعاء الدالة `os.path.abspath()`، فسيكون المجلد . (نقطة واحدة) هو المسار المطلق لمجلد العمل الحالي:

```
'C:\\Users\\Al\\AppData\\Local\\Programs\\Python\\Python37'
```

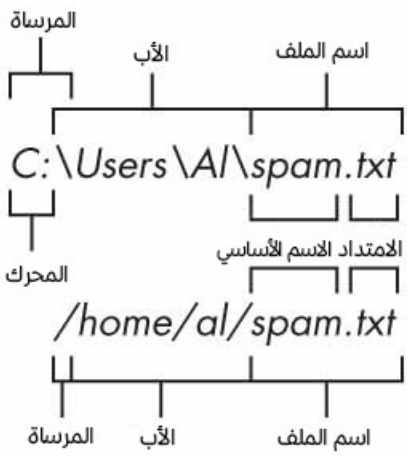
لنجرب الدالة `os.path.relpath()` في الصدفة التفاعلية:

```
>>> os.path.relpath('C:\\Windows', 'C:\\')
'Windows'
>>> os.path.relpath('C:\\Windows', 'C:\\spam\\eggs')
'..\\..\\Windows'
```

إذا امتلك المسار النسبي نفس الأب لمجلد العمل الحالي كما في `'C:\\Windows'` و `'C:\\spam\\eggs'` فيمكن استخدام النقطتين .. للوصول إلى المجلد الأب.

9.1.8 الحصول على أقسام المسار

يمكننا استخلاص مختلف أقسام المسار عبر خاصيات الكائن `Path`، وهذا يفيدنا في حال أردنا إنشاء مسار جديد اعتمادًا على مسار ملف موجود مسبقًا. الشكل التالي يوضح هذه الخاصيات:



الشكل 40: أجزاء المسار (ويندوز في الأعلى، ولينكس أو ماك في الأسفل)

تتألف مسارات الملفات من الأقسام الآتية:

- المرساة anchor وهي المجلد الجذر root directory في نظام الملفات.
- المحرك drive في ويندوز وهو حرف واحد يشير إلى القرص المستخدم.

- الأب parent وهو مسار المجلد الذي يحتوي على الملف.
- اسم الملف name وهو يتألف من الاسم الأساسي stem والامتداد أو اللاحقة suffix.

لاحظ أن كائنات `Path` تمتلك الخاصية `drive` في ويندوز، لكنها غير موجودة في ماك أو لينكس.

لاحظ أيضًا أن الخاصية `drive` لا تتضمن أول خط مائل خلفي.

نجرب هذه الخاصيات على أحد المسارات:

```
>>> p = Path('C:/Users/Al/spam.txt')
>>> p.anchor
'C:\\'
>>> p.parent # لن يعيد سلسلة نصية بل كائن Path
WindowsPath('C:/Users/Al')
>>> p.name
'spam.txt'
>>> p.stem
'spam'
>>> p.suffix
'.txt'
>>> p.drive
'C:'
```

ستعيد هذه الخاصيات سلاسل نصية باستثناء الخاصية `parent` التي ستعيد كائن `Path` آخر.

تمنحنا الخاصية `parents` (التي تختلف عن الخاصية `parent` السابقة) وصولًا إلى كائنات `Path` للمجلدات الأب مع فهرس رقمي:

```
>>> Path.cwd()
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python37')
>>> Path.cwd().parents[0]
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python')
>>> Path.cwd().parents[1]
WindowsPath('C:/Users/Al/AppData/Local/Programs')
>>> Path.cwd().parents[2]
WindowsPath('C:/Users/Al/AppData/Local')
>>> Path.cwd().parents[3]
WindowsPath('C:/Users/Al/AppData')
```

```
>>> Path.cwd().parents[4]
WindowsPath('C:/Users/Al')
>>> Path.cwd().parents[5]
WindowsPath('C:/Users')
>>> Path.cwd().parents[6]
WindowsPath('C:/')
```

تمتلك الوحدة `os.path` القديمة دوال مشابهة لما سبق للحصول على مختلف أقسام المسارات مثل سلسلة نصية.

ستعيد الدالة `os.path.dirname(path)` سلسلةً نصيةً فيها كل ما يسبق آخر خط مائل في المعامل `path`، بينما ستعيد `os.path.basename(path)` سلسلةً نصيةً فيها كل ما يلي آخر خط مائل في المعامل `path`. الشكل التالي يوضح مخرجات الدالتين السابقتين:

الشكل 41: اسم المجلد dirname واسم الملف basename في الوحدة os.path

لنجربها عمليًا في الطرفية التفاعلية:

```
>>> calcFilePath = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.basename(calcFilePath)
'calc.exe'
>>> os.path.dirname(calcFilePath)
'C:\\Windows\\System32'
```

إذا احتجت إلى اسم المجلد واسم الملف معًا، فيمكنك استدعاء الدالة `os.path.split()` للحصول على صف فيه سلسلتين نصيتين كما يلي:

```
>>> calcFilePath = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.split(calcFilePath)
('C:\\Windows\\System32', 'calc.exe')
```

لاحظ أنك تستطيع إنشاء نفس الصف السابق باستدعاء الدالتين `os.path.dirname()` و`os.path.basename()` بنفسك:

```
>>> (os.path.dirname(calcFilePath), os.path.basename(calcFilePath))
('C:\\Windows\\System32', 'calc.exe')
```

لكن استخدام `os.path.split()` سيوفر عليك بعض الوقت أحيانًا.

لاحظ أن الدالة `os.path.split()` لا تأخذ مسار أحد الملفات وتعيد قائمةً من السلاسل النصية تمثل كل مجلد، وإنما علينا استخدام الدالة `split()` الخاصة بالسلاسل النصية وتقسيم المسار عند كل فاصل `os.sep` (لاحظ أن الفاصل `sep` موجود في `os` وليس `os.path`).

يتمثل المتغير `os.sep` الفاصل بين المجلدات في نظام التشغيل المستخدم، وهو `'\\'` في ويندوز و `'/'` في ماك ولينكس:

```
>>> calcFilePath.split(os.sep)
['C:', 'Windows', 'System32', 'calc.exe']
```

سعيد المثال السابق جميع أقسام المسار كقائمة من السلاسل النصية. وسيكون أول عنصر في القائمة المعادة في أنظمة ماك ولينكس هو سلسلة نصية فارغة:

```
>>> '/usr/bin'.split(os. sep)
['', 'usr', 'bin']
```

9.1.9 الحصول على حجم ملف ومحتويات مجلد

بعد أن تعلمنا كيفية التعامل مع مسارات الملفات والمجلدات، يمكننا الان أن نبدأ بجمع المعلومات حولها. توفر الوحدة `os.path` ما يلزم لمعرفة الحجم التخزيني لملفٍ ما بالبايت، أو للملفات والمجلدات الموجودة في مجلد معين.

- استدعاء `os.path.getsize(path)` سيعيد الحجم التخزيني للملف الموجود في المسار `path`، وذلك بالبايت.
- استدعاء `os.listdir(path)` سيعيد قائمة list فيها أسماء كل الملفات الموجودة في المسار `path`، لاحظ أننا استعملنا هنا الوحدة `os` وليس `os.path`.

لنجرب هذه الدوال في الطرفية التفاعلية:

```
>>> os.path.getsize('C:\\Windows\\System32\\calc.exe')
27648
>>> os.listdir('C:\\Windows\\System32')
['0409', '12520437.cpx', '12520850.cpx', '5U877.ax', 'aaclient.dll',
--snip--
'xwtpdui.dll', 'xwtpw32.dll', 'zh-CN', 'zh-HK', 'zh-TW',
'zipfldr.dll']
```

كما هو واضح، حجم الملف `calc.exe` في حاسوبي هو 27,648 بايت، ولدي الكثير من الملفات في المسار `C:\Windows\system32`، وإذا أردت الحصول على الحجم التخزيني لكل الملفات في هذا المجلد فسأستخدم `os.path.getsize()` و `os.listdir()` معًا.

```
>>> totalSize = 0
>>> for filename in os.listdir('C:\\Windows\\System32'):
  totalSize = totalSize + os.path.getsize(os.path.join('C:\\Windows\
\System32', filename))
>>> print(totalSize)
2559970473
```

سأمر بحلقة التكرار على كل ملف موجود في المجلد `C:\Windows\System32` وسأزيد قيمة المتغير `totalSize` بمقدار الحجم التخزيني لكل ملف، لاحظ أنه حين استدعاء `os.path.getsize()` فنستخدم `os.path.join()` لإضافة اسم المجلد إلى اسم الملف الحالي.

ستضاف القيمة العددية المعادة من `os.path.getsize()` إلى قيمة `totalSize`، وبعد إنهاء المرور على جميع الملفات فسنطبع قيمة `totalSize` لمعرفة حجم المجلد `C:\Windows\System32` التخزيني.

9.1.10 الحصول على قائمة الملفات التي تطابق نمطًا معينًا باستخدام ()glob

إذا أردت العمل على ملفات محددة فمن الأسهل استخدام التابع `glob()` بدلًا من `listdir()`، إذ تملك كائنات `Path` التابع `glob()` للحصول على قائمة الملفات الموجودة داخل مجلد وفقًا لنمط يسمى Glob pattern، والتي تعرف أيضًا بالمحارف البديلة wildcard characters وهي نسخة مبسطة من التعابير النمطية التي يشيع استخدامها في سطر الأوامر (وتسمى هناك بالتوسعات expansion).

يعيد التابع `glob()` كائنًا مولدًا generator object (وهو خارج عن سياق هذا الكتاب)، الذي يمكننا تمريره إلى الدالة `list()` لتسهيل التعامل معه:

```
>>> p = Path('C:/Users/Al/Desktop')
>>> p.glob('*')
<generator object Path.glob at 0x000002A6E389DED0>
>>> list(p.glob('*'))
[WindowsPath('C:/Users/Al/Desktop/1.png'), WindowsPath('C:/Users/Al/
Desktop/22-ap.pdf'), WindowsPath('C:/Users/Al/Desktop/cat.jpg'),
 --snip--
WindowsPath('C:/Users/Al/Desktop/zzz.txt')]
```

رمز النجمة * يعني "مجموعة من أي نوع من المحارف"، وبالتالي سيعيد `p.glob('*')` مولدًا فيه جميع الملفات الموجودة في المسار المخزن في p.

وكما في التعابير النمطية، يمكننا كتابة أنماط معقدة بعض الشيء:

```
>>> list(p.glob('*.txt') # قائمة بكل الملفات النصية
[WindowsPath('C:/Users/Al/Desktop/foo.txt'),
 --snip--
WindowsPath('C:/Users/Al/Desktop/zzz.txt')]
```

سيعيد النمط `'*.txt'` كل الملفات التي تبدأ بأي مجموعة من المحارف طالما أنها تنتهي بالسلسلة النصية `'.txt'`، وهو عادةً امتداد الملفات النصية. أما رمز إشارة الاستفهام ? فيعني أي محرف واحد:

```
>>> list(p.glob('project?.docx')
[WindowsPath('C:/Users/Al/Desktop/project1.docx'),
WindowsPath('C:/Users/Al/
Desktop/project2.docx'),
 --snip--
WindowsPath('C:/Users/Al/Desktop/project9.docx')]
```

التعبير `'project?.docx'` سيعيد `'project1.docx'` أو `'project5.docx'`، لكنه لن يطابق `'project10.docx'` لأن رمز علامة الاستفهام ? سيطابق محرفًا واحدًا فقط، لذا لن يستطيع مطابقة محرفين `'10'`.

يمكنك أن تستعمل رمز النجمة وعلامة الاستفهام معًا لإنشاء تعابير مخصصة مثل:

```
>>> list(p.glob('*.?x?')
[WindowsPath('C:/Users/Al/Desktop/calc.exe'),
WindowsPath('C:/Users/Al/
Desktop/foo.txt'),
 --snip--
WindowsPath('C:/Users/Al/Desktop/zzz.txt')]
```

التعبير `'*.?x?'` سيعيد جميع الملفات التي لها أي اسم لكنها تنتهي بلاحقة تتألف من 3 محارف، والمحرف الوسط بينها هو `'x'`.

يسهل علينا التابع `glob()` تحديد الملفات التي نريدها باختيار الملفات التي يطابق اسمها نمطًا معينًا. سنستخدم هنا الحلقة `for` للمرور على المولد generator المولد من التابع `glob()`:

```
>>> p = Path('C:/Users/Al/Desktop')
>>> for textFilePathObj in p.glob('*.txt'):
...     print(textFilePathObj) # طباعة كائن Path كسلسلة نصية
...     # معالجة الملف النصي
...
C:\Users\Al\Desktop\foo.txt
C:\Users\Al\Desktop\spam.txt
C:\Users\Al\Desktop\zzz.txt
```

إذا أردت إجراء نفس العملية على جميع الملفات الموجودة في المجلد، فيمكنك أن تستعمل حينها `os.listdir(p)` أو `p.glob('*')`.

9.1.11 التأكد من المسارات

ستفشل أغلبية دوال بايثون التي تتعامل مع الملفات إذا أعطيناها مسارًا غير موجود، لكن لحسن الحظ هنالك توابع لكائنات `Path` للتحقق أن المسار المعطى موجود فعلًا، وهل هو ملف أم مجلد. فعلى فرض أن المتغير `p` يشير إلى كائن `Path`، فبالتالي سيعيد استدعاء:

- `p.exists()` القيمة `True` إذا كان المسار موجودًا، أو `False` إن لم يكن موجودًا.
- `p.is_file()` القيمة `True` إن كان المسار موجودًا ويشير إلى ملف، أو `False` خلاف ذلك.
- `p.is_dir()` القيمة `True` إذا كان المسار موجودًا ويشير إلى مجلد، أو `False` خلاف ذلك.

دعني أجرب هذه التوابع على حاسوبي الشخصي:

```
>>> winDir = Path('C:/Windows')
>>> notExistsDir = Path('C:/This/Folder/Does/Not/Exist')
>>> calcFile = Path('C:/Windows
/System32/calc.exe')
>>> winDir.exists()
True
>>> winDir.is_dir()
True
>>> notExistsDir.exists()
False
>>> calcFile.is_file()
True
>>> calcFile.is_dir()
False
```

إذا كنت تستعمل ويندوز فيمكنك أن تتحقق إن كان قرص التخزين المؤقت «الفلاشة» موصولًا إلى الحاسوب عبر التابع `exists()`، فمثلًا لو أردت التحقق أن القرص المسمى `D:\` موجود على حاسوبي:

```
>>> dDrive = Path('D:/')
>>> dDrive.exists()
False
```

يبدو أنني نسيت وصل القرص إلى الحاسوب.

الوحدة القديمة `os.path` تستطيع إنجاز نفس المهمة باستخدام `os.path.exists(path)` و`os.path.isfile(path)` و `os.path.isdir(path)`، التي تعمل ملف مكافأتها في كائنات `Path`. وبدءًا من الإصدار بايثون 3.6 أصبحت تقبل هذه التوابع كائنات `Path` إضافةً إلى سلاسل نصية تحتوي على مسارات الملفات.

9.2 عملية قراءة الملفات والكتابة إليها

بعد أن تصبح مرتاحًا بالتعامل مع المجلدات والمسارات النسبية، فستتمكن من تحديد موقع الملفات التي تريد قراءتها أو الكتابة إليها.

الدوال التي سنشرحها في الأقسام الآتية تعمل على الملفات النصية البسيطة، التي هي ملفات تحتوي على محارف نصية دون أن تحتوي على معلومات التنسيقات مثل الخطوط أو الألوان أو خلاف ذلك، ومن الأمثلة على الملفات النصية البسيطة هي ملفات txt أو py التي تحتوي على شيفرات بايثون.

يمكن فتح هـذه الملفـات باسـتخدام المفكـرة Notepad في ويـنـدوز، أو TextEdit في مـاك، أو Kate أو Gedit في لينكس. وتستطيع أن تفتح هذه الملفات في برامجك وتعاملها كسلاسل نصية عادية.

الملفات الثنائية هي نوع آخر من الملفات، مثل الملفات التي تنتجهـا بـرامج إنشـاء العـروض التقديميـة أو ملفات PDF أو الصور أو الملفات التنفيذية ...إلخ.

وإذا فتحتها بالمفكرة مثلًا فستجد أنها مجموعة من الرموز غير المفهومة:

الشكل 42: برنامج calc.exe مفتوح في المفكرة

ولأن كل نوع من الملفات الثنائية يجري التعامل معه بطريقة مختلفة، فلن ندخل بتفاصيل تعديل الملفـات الثانية مباشرةً في هذا الكتاب؛ وهنالك وحدات تسهل التعامل معها مثل الوحدة `shelve` التي ستتعامل معهـا لاحقًا في هذا الفصل.

التابع `read_text()` في الوحدة `pathlib` تعيد سلسلةً نصية فيها كل محتويات الملـف النصـي، بينمـا يكتب التابع `write_text()` ما يمرر إليه إلى ملف نصي جديد (أو يعيد الكتابة فوق ملف موجود مسبقًا):

```
>>> from pathlib import Path
>>> p = Path('spam.txt')
>>> p.write_text('Hello, world!')
13
>>> p.read_text()
'Hello, world!'
```

سننشئ ملفًا باسم `spam.txt` فيه المحتويات `'Hello, world!'`، لاحظ أن التـابع `write_text()` قد أعاد الرقم 13 الذي يشير إلى عدد المحارف التي كتبت إلى الملف (ونتجاهل تخزين هـذا الـرقم عـادةً)، ويقـرأ التابع `read_text()` محتويات الملف الجديد ويعيدها على شكل سلسلة نصية.

تذكر أن توابع الكائن `Path` توفر الأمور الأساسية في التعامل مع الملفات؛ والطريقة الأشيع لقراءة الملفات تكون عبر الدالة `open()` والكائن `File`. هنالك خطوات ثلاث لقراءة أو كتابة الملفات في بايثون:

1. استدعاء الدالة `open()` لإعادة الكائن `File`.
2. استدعاء التابع `read()` أو `write()` على الكائن `File`.
3. إغلاق الملف باستدعاء التابع `close()` على الكائن `File`.

سنشرح هذه الخطوات في الأقسام الآتية.

9.2.1 فتح الملفات عبر الدالة ()open

لفتح ملف باستخدام الدالة `open()` فنمرر سلسلة نصية تحتوي على مسار الملف الذي نريد فتحه؛ والذي يكون إما مسارًا مطلقًا absolute أو نسبيًا relative. ستعيد الدالة `open()` كائنًا من النوع `File`.

لنجربها بإنشاء ملف نصي بسيط اسمه `hello.txt` باستخدام المفكرة أو أي محرر نصوص، وكتابة `Hello, World!` داخلها وحفظه في مجلد المنزل، ثم كتابة ما يلي في الطرفية التفاعلية:

```
>>> helloFile = open(Path.home() / 'hello.txt')
```

تقبل الدالة `open()` السلاسل النصية أيضًا، فإذا كنت تستخدم ويندوز فاكتب:

```
>>> helloFile = open('C:\\Users\\your_home_folder\\hello.txt')
```

أما إذا كان نظامك ماك:

```
>>> helloFile = open('/Users/your_home_folder/hello.txt')
```

تذكر أن تبدل الكلمة your_home_folder باسم المستخدم في حاسوبك، فلو كان hsoub مثلًا فستدخل `'C:\\Users\\hsoub\\hello.txt'` في ويندوز؛ لاحظ أن الدالة `open()` أصبحت تقبل كائنات `Path` بدءًا من إصدار بايثون 3.6، وكان عليك استخدام السلاسل النصية فقط فيما سبق.

ستفتح هذه الأوامر الملف في وضع "قراءة الملفات النصية" أو اختصارًا "وضع القراءة"، وحينما يفتح الملف في وضع القراءة فتسمح لنا بايثون بقراءة الملفات من الملف فقط، ولا يمكنك أن تكتب عليه أو تعدله بأي شكل. لكن إذا أردت أن تحدد أنك تريد فتح الملف بوضع القراءة بوضوح فمرر القيمة `'r'` كثاني وسيط إلى الدالة `open()`، أي أن كلًا من الدالتين `open('/Users/Al/hello.txt', 'r')`، وأيضًا الدالة `open('/Users/Al/hello.txt')` متكافئتان تمامًا.

سيعيد استدعاء الدالة `open()` كائن `File`، ويمثل كائن `File` ملفًا على حاسوبك، وهو نوع مختلف من القيم في بايثون مثله كمثل القوائم أو القواميس التي تعرفت عليها مسبقًا.

خزنّا في المثال السابق كائن `File` في المتغير `helloFile`، ويمكنك أن تستسخدمه لأي عمليـات قـراءة أو كتابة مستقبلًا باستدعاء التوابع المناسبة على الكائن `File` المخزن في المتغير `helloFile`.

9.2.2 قراءة محتويات الملفات

أصبح لدينا الآن كائن `File`، ويمكننا أن نبدأ بقراءة كامل محتويات الملف كسلسلة نصية، وذلك عبر التـابع `read()`، لنكمل مثالنا السابق الذي فيه المتغير `helloFile` بكتابة ما يلي:

```
>>> helloContent = helloFile.read()
>>> helloContent
'Hello, world!'
```

لو تخيلت أن جميع محتويات الملـف هي سلسـلة نصـية كبـيرة، فسـيعيد لـك التـابع `read()` يعيـد تلـك السلسلة النصية.

بدلًا من ذلك، يمكنك استخدام التـابع `readlines()` للحصـول على قائمـة list فيهـا سلاسـل نصـية من الملف، وكل سلسلة نصية تمثل سطرًا فيه، فمثلًا لو كان لدينا ملف اسـمه `sonnet29.txt` في نفس المجلـد الذي فيه الملف `hello.txt` وكتبنا فيه النص الآتي:

```
When, in isgrace with fortune and men's eyes,
I all alone beweep my outcast state,
And trouble deaf heaven with my bootless cries,
And look upon myself and curse my fate,
```

تأكد أنك قد فصلت بين الأسطر الأربعة السابقة كلٌ في سطر مختلف، ثم أدخل الآتي بالطرفية التفاعلية:

```
>>> sonnetFile = open(Path.home() / 'sonnet29.txt')
>>> sonnetFile.readlines()
[When, in disgrace with fortune and men's eyes,\n', ' I all alone
beweep my
outcast state,\n', And trouble deaf heaven with my bootless cries,\n',
And
look upon myself and curse my fate,']
```

لاحظ أن كل عنصر من عناصر القائمة (عدا آخر واحد) هو سلسلة نصية تنتهي بمحرف السطر الجديد `\n`.

يكون في العادة من الأسهل التعامل مع قائمة من السلاسل النصية بدل سلسلة نصية واحدة كبيرة.

9.2.3 الكتابة إلى الملفات

تسمح لنا بايثون بكتابة محتوى إلى الملفات يشبه "كتابة" الدالة `print()` للسلاسل النصية إلى الشاشة.

لا يمكنك أن تكتب إلى ملف قد فتحته بوضع القراءة، لذا عليك أن تفتحه بوضع "الكتابة على الملفات النصية" أو "الإضافة إلى الملفات النصية" واختصارًا وضع الكتابة أو وضع الإضافة.

وضع الكتابة سيعيد الكتابة فوق ملف موجود ويبدأ من الصفر، كما لو أعدنا إسناد قيمة جديدة إلى متغير. يمكنك فتحت الملف بوضع الكتابة بتمرير `'w'` كثاني وسيط إلى الدالة `open()`.

أما وضع الإضافة فسيضيف النص إلى نهاية ملف موجود مسبقًا، كما لو أضفنا عنصرًا إلى قائمة موجودة في متغير بدلًا من إعادة الكتابة. مرر `'a'` كثاني وسيط إلى الدالة `open()` لفتح الملف في وضع الإضافة.

إذا لم يكن الملف الممرر إلى الدالة `open()` موجودًا فسينشأ ملف جديد في وضع الكتابة والإسناد. لا تنسَ أن تستدعي الدالة `close()` قبل إعادة فتح الملف مجددًا.

لنجرب هذه المفاهيم معًا بكتابة:

```
>>> oliveFile = open('olive.txt', 'w')
>>> oliveFile.write('Hello, world!\n')
13
>>> oliveFile.close()
>>> oliveFile = open('olive.txt', 'a')
>>> oliveFile.write('Olive is not a vegetable.')
25
>>> oliveFile.close()
>>> oliveFile = open('olive.txt')
>>> content = oliveFile.read()
>>> oliveFile.close()
>>> print(content)
Hello, world!
Olive is not a vegetable.
```

في البداية فتحنا الملف `becon.txt` بوضع الكتابة، ولعدم وجود الملف `becon.txt` بعد فإن بايثون تنشئه لنا، واستدعاء التابع `write()` وتمرير السلسلة النصية `'Hello, world! /n'` سيؤدي إلى كتابتها إلى الملف وإعادة عدد المحارف المكتوبة بما فيها محرف السطر الجديد. ثم أغلقنا في النهاية الملف.

لإضافة نص إلى المحتويات الموجودة لملف بدلًا من استبداله، فسنفتح الملف في وضع الإضافة، وأضفنا السلسلة النصية `'Olive is not a vegetable.'` إلى الملف وأغلقناه.

في النهاية نريد أن نطبع محتويات الملف فاستخدمنا الدالة `open()` لفتح الملف في الوضع الافتراضي وهو وضع القراءة، وخزنّا محتويات الملف في المتغير `content` ثم أغلقنا الملف وطبعنا محتوياته.

لاحظ أن التابع `write()` لا يضيف محرف السطر الجديد إلى نهاية السلسلة النصية مثلما تفعل الدالة `print()` لذا عليك أن تضيفه بنفسك. تذكر أنك تستطيع تمرير كائن `Path` إلى الدالة `open()` بدلًا من سلسلة نصية بسيطة بدءًا من إصدار بايثون 3.6.

9.3 حفظ المتغيرات باستخدام الوحدة shelve

يمكنك أن تحفظ المتغيرات الموجودة في برامجك إلى ملفات ثنائية باستخدام الوحدة `shelve`، وبالتالي يمكنك أن تستعيد البيانات إلى المتغيرات من ملف مخزن على حاسوبك.

تسمح لك الوحدة `shelve` بحفظ واستعادة البيانات إلى برنامجك، فلو ضبطتَ مثلًا بعض المتغيرات في برنامجك، يمكنك أن تحفظ تلك البيانات على الرف (`shelf`، ومن هنا أتى اسم الوحدة) ثم تستعيد تلك القيم حينما تشغل برنامجك مجددًا.

```
>>> import shelve
>>> shelfFile = shelve.open('mydata')
>>> cats = ['Zophie', 'Pooka', 'Simon']
>>> shelfFile['cats'] = cats
>>> shelfFile.close()
```

لقراءة وكتابة البيانات باستخدام الوحدة `shelve` عليك أن تستوردها أولًا، ثم تستدعي `shelve.open()` وتمرر إليها اسم الملف ثم تخزن القيم. لاحظ أنك تستطيع التعامل مع القيم كما لو أنها قاموس. بعد أن تنتهي لا تنسَ استدعاء `close()`.

أنشأنا في المثال السابق القائمة `cats` وكتبنا `shelfFile['cats'] = cats` لتخزين القائمة في `shelfFile` كقيمة مرتبطة مع المفتاح `'cat'` (كما في مفاتيح القواميس). ثم استدعينا `close()` على `shelfFile`، لاحظ أنه بدءًا من إصدار بايثون 3.7 سيكون عليك تمرير أسماء الملفات إلى `open()` كسلاسل نصية، ولا يمكنك تمرير كائن `Path`.

بعد تشغيل الشيفرة السابقة في ويندوز، ستجد ثلاثة ملفات جديدة في المجلد وهي `mydata.bak` و`mydata.dat` و`mydata.dir`؛ أما على ماك فسينشَأ ملف واحد باسم `mydata.db`.

تحتوي هذه الملفات الثنائية على البيانات التي خزنتها «على الرف»؛ ولا تهمك صيغة هذه الملفات الثانية، فكل ما تحتاج إلى معرفته هو ما تفعله الوحدة `shelve` وليس كيف تفعل ذلك. وهذه الوحدة تريح رأسك من القلق حول كيفية تخزين البرنامج للبيانات.

يمكن لبرامجك استخدام الوحدة `shelve` لإعادة فتح الملف والحصول على البيانات، ولا حاجة إلى تحديد إن كنت تريد قراءة البيانات أم كتابتها، ففتح الملف يسمح بكلي العمليتين.

```
>>> shelfFile = shelve.open('mydata')
>>> type(shelfFile)
<class 'shelve.DbfilenameShelf'>
>>> shelfFile['cats']
['Zophie', 'Pooka', 'Simon']
>>> shelfFile.close()
```

فتحنا في المثال السابق الملف الذي وضعنا فيه البيانات، وتأكدنا أن التخزين سليم إذ أعاد `shelfFile['cats']` نفس القائمة التي خزناها سابقًا، وفي النهاية أغلقنا الملف `close()`.

هنالك تابعان اسمهما `keys()` و `values()` تشبه تلك الموجودة في القواميس التي تعيد قيمةً شبيهة بالقوائم list-like للمفاتيح والقيم الموجودة في الرف. ولأن هذه التوابع تعيد قيمًا شبيهة بالقوائم وليست قوائم حقيقية فيجب عليك تمريرها إلى الدالة `list()` للحصول على قائمة حقيقية تتعامل معها.

```
>>> shelfFile = shelve.open('mydata')
>>> list(shelfFile.keys())
['cats']
>>> list(shelfFile.values())
[['Zophie', 'Pooka', 'Simon']]
>>> shelfFile.close()
```

الخلاصة أن الملفات النصية البسيطة مفيدة لتخزين البيانات النصية الأساسية، أما لو أردت حفظ بيانات من برنامج بايثون الذي كتبته، فيمكنك أن تستفيد من الوحدة `shelve`.

9.4 حفظ المتغيرات مع الدالة pprint.pformat()

إذا كنت تذكر في قسم «تجميل الطباعة» أن الدالة `pprint.pprint()` تطبع محتويات قائمة أو قاموس بتنسيق مخصص، بينما الدالة `pprint.pformat()` تنسق النص وتعيده بدلًا من طباعته مباشرةً.

وصحيحٌ أن النص المعاد من هذه الدالة سيكون منسقًا تنسيقًا جميلًا لتسهيل قراءته، لكنه في الواقع منسق كما لو أنه شفيرة بايثون.

لنقل مثلًا أن لديك قاموسًا مخزنًا في متغير وأردت حفظ هذا المتغير ومحتوياته للاستخدام مستقبلًا، فيمكنك أن تستفيد من الدالة `pprint.pformat()` لإعادة سلسلة نصية تكتبها إلى ملف `.py`. ويمكنك أن تستورد هذا الملف في أي مرة تريد استخدام المتغير المخزن فيه.

```
>>> import pprint
>>> cats = [{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka',
'desc': 'fluffy'}]
>>> pprint.pformat(cats)
"[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy', 'name':
'Pooka'}]"
>>> fileObj = open('myCats.py', 'w')
>>> fileObj.write('cats = ' + pprint.pformat(cats) + '\n')
83
>>> fileObj.close()
```

استوردنا في هذا المثال الوحدة `pprint` لكي نستطيع استخدام الدالة `pprint.pformat()`، ولدينا متغير `cats` فيه قائمة من القواميس؛ ولكي نحتفظ بالقائمة الموجودة في المتغير `cats` حتى بعد أن نغلق الصدفة التفاعلية فيمكننا استخدام الدالة `pprint.pformat()` لإعادته كسلسلة نصية، ثم بعد حصولنا على السلسلة النصية يمكننا كتابتها إلى ملف وليكن اسمه `myCats.py`.

تذكر أن الوحدات التي تستوردها العبارة `import` هي سكربتات بايثون عادية؛ وعندما نحفظ السلسلة النصية المأخوذة من `pprint.pformat()` إلى ملف `.py` فيمكن اعتبار هذا الملف على أنه وحدة يمكن استيرادها مثل أي وحدات بايثون الأخرى.

ولأن سكربتات بايثون هي ملفات نصية بسيط امتدادها `.py` فيمكن لبرامجك أن تولد برامج بايثون أخرى، ويمكنك استيراد تلك البرامج داخل برامجك:

```
>>> import myCats
>>> myCats.cats
[{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc':
'fluffy'}]
>>> myCats.cats[0]
{'name': 'Zophie', 'desc': 'chubby'}
>>> myCats.cats[0]['name']
'Zophie'
```

الفائدة من إنشاء ملف `.py` بسيط بدلًا من حفظ المتغيرات مع الوحدة `shelve` هو أن الناتج ملف نصي يمكن قراءته وتعديله من أي شخص بمحرر نصي بسيط.

لكن لأغلبية حالات الاستخدام يكون من المناسب حفظ البيانات باستخدام الوحدة `shelve`، إذا لا يمكن كتابة القيم إلى ملفات نصية بسيطة إلا إذا كانت قيمًا بسيطة مثل الأعداد والسلاسل النصية والقوائم والقواميس، بينما لا تستطيع تخزين الكائنات مثل `File` أو غيره كنص بسيط.

9.5 مشروع: توليد ملفات اختبارات عشوائية

لنفترض أنك أستاذ مادة الجغرافيـة ولـديك 35 طالبًـا في صـفحك، وتريـد إجـراء اختبـار لعواصـم الولايـات الأمريكية؛ وأنت تعرف طلابك حق المعرفة وتدرك أن بعضهم سيحاول أن يغش، لذا تريد أن تغيّر ترتيب الأسئلة في كل اختبار لكي تكون فريدة مما يجعل من الصعب جدًا نقل الإجابات من طالب آخر. عمل هذه النماذج يدويًا يأخذ وقتًا وجهدًا وسيكون أمرًا مملًا، لكن ستساعدك مهاراتك في بايثون هنا.

هذه هي وظيفة البرنامج:

1. إنشاء 35 اختبار مختلف.
2. إنشاء 50 سؤال اختيار من إجابات متعددة لكل اختبار، بترتيب عشوائي.
3. توفير الإجابة الصحيحة وثلاث إجابات خطأ لكل سؤال مرتبة ترتيبًا عشوائيًا.
4. كتابة الاختبارات إلى 35 ملف نصي.
5. كتابة مفاتيح الإجابات الصحيحة إلى 35 ملف نصي.

هذا يعني أن الشيفرة عليها أن:

1. تخزن أسماء الولايات في أمريكا وأسماء عواصمها في قاموس
2. تستدعي `open()` و `write()` و `close()` لكل ملف اختبار وإجابات
3. تستخدم `random.shuffle()` لترتيب الأسئلة والإجابات ترتيبًا عشوائيًا

9.5.1 الخطوة 1: تخزين بيانات الاختبار في قاموس

أول خطـوة هي إنشـاء بيئـة السـكربت الأساسـية وكتابـة بيانـات الاختبـار. أنشِـئ ملفًـا باسـم `randomQuizGenerator.py` وضع فيه المحتوى الآتي:

```
#! python3
# randomQuizGenerator.py - إنشاء اختبارات مع أسئلة وإجابات عشوائية
# مع تخزين مفتاح الحل

❶ import random

# بيانات الاختبار: أسماء الولايات الأمريكية وعواصمها
```

```
❷ capitals = {'Alabama': 'Montgomery', 'Alaska': 'Juneau', 'Arizona':
'Phoenix',
 'Arkansas': 'Little Rock', 'California': 'Sacramento', 'Colorado':
'Denver',
 'Connecticut': 'Hartford', 'Delaware': 'Dover', 'Florida':
'Tallahassee',
 'Georgia': 'Atlanta', 'Hawaii': 'Honolulu', 'Idaho': 'Boise',
'Illinois':
 'Springfield', 'Indiana': 'Indianapolis', 'Iowa': 'Des Moines',
'Kansas':
 'Topeka', 'Kentucky': 'Frankfort', 'Louisiana': 'Baton Rouge',
'Maine':
 'Augusta', 'Maryland': 'Annapolis', 'Massachusetts': 'Boston',
'Michigan':
 'Lansing', 'Minnesota': 'Saint Paul', 'Mississippi': 'Jackson',
'Missouri':
 'Jefferson City', 'Montana': 'Helena', 'Nebraska': 'Lincoln',
'Nevada':
 'Carson City', 'New Hampshire': 'Concord', 'New Jersey': 'Trenton',
'New
 Mexico': 'Santa Fe', 'New York': 'Albany',
 'North Carolina': 'Raleigh', 'North Dakota': 'Bismarck', 'Ohio':
'Columbus', 'Oklahoma': 'Oklahoma City',
 'Oregon': 'Salem', 'Pennsylvania': 'Harrisburg', 'Rhode Island':
'Providence',
 'South Carolina': 'Columbia', 'South Dakota': 'Pierre', 'Tennessee':
 'Nashville', 'Texas': 'Austin', 'Utah': 'Salt Lake City', 'Vermont':
 'Montpelier', 'Virginia': 'Richmond', 'Washington': 'Olympia', 'West
 Virginia': 'Charleston', 'Wisconsin': 'Madison', 'Wyoming':
'Cheyenne'}

  # تولید 35 اختبار عشوائي.

❸ for quizNum in range(35):
```

```
    # TODO: إنشاء ملفات الاختبار والإجابات.

    # TODO: كتابة ترويسة الاختبار.

    # TODO: تغيير ترتيب الولايات.

    # TODO: المرور على 50 سؤال وتوليد إجابات عشوائية.
```

لما كان هذا البرنامج يرتب الأسئلة والإجابات عشوائيًا، فعلينا اسـتيراد الوحـدة `random` ❶ لكي نستطيع استخدام الدوال الخاصة بها. يحتوي المتغير `capitals` ❷ على قاموس فيه أسماء الولايات الأمريكية كمفتاح وعاصمتها كقيمة. ولأننا نريد كتابة الشيفرة التي تولد ملفات الاختبار والإجابات (أضفناها على أنها TODO حاليًا) فسندخل إلى حلقة التكرار `for` بعدد 35 مرة ❸، ويمكننا تغيير الرقم وفق متطلبات البرنامج.

9.5.2 الخطوة 2: إنشاء ملف الاختبار وتغيير ترتيب الأسئلة عشوائيا

حان الوقت لبدء العمل على مهام TODO.

ستكرر الشيفرة الموجودة داخل حلقة `for` بعدد 35 مرة، مرة لكل اختبار، فعلينا أن نفكر بكيفية إنشاء اختبار واحدة وسيكرر الأمر على البقية.

بدايةً سنحتاج إلى إنشاء ملف الاختبار، ويجب أن يكون له اسم فريد، ويجب أن يحتوي على ترويسة قياسية فيها معلومات الاختبار ومكان ليضع الطالب اسمه وصفه وتاريخ اليوم. ثم ستكون هنالك قائمة الولايات بترتيب عشوائي، التي يمكن استخدامها لاحقًا لإنشاء الأسئلة والإجابات لكل اختبار.

أضف الأسطر الآتية إلى ملف `randomQuizGenerator.py`:

```
#! python3
# randomQuizGenerator.py - إنشاء اختبارات مع أسئلة وإجابات عشوائية

# مع تخزين مفتاح الحل.
--snip--
# توليد 35 اختبار عشوائي.
for quizNum in range(35):
  # إنشاء ملف الاختبارات والإجابات

 ❶ quizFile = open(f'capitalsquiz{quizNum + 1}.txt', 'w')

 ❷ answerKeyFile = open(f'capitalsquiz_answers{quizNum + 1}.txt', 'w')
```

```
    # كتابة الترويسة
❸ quizFile.write('Name:\n\nDate:\n\nPeriod:\n\n')
    quizFile.write((' ' * 20) + f'State Capitals Quiz (Form{quizNum + 1})')
    quizFile.write('\n\n')

    # تغيير ترتيب الولايات
    states = list(capitals.keys())
❹ random.shuffle(states)

    # TODO: المرور على 50 سؤال وتوليد إجابات عشوائية.
```

ستكون أسماء ملفات الاختبارات على الشكل `capitalsquiz<N>.txt` حيث `<N>` هو رقم فريد لكل اختبار يأتي من `quizNum` الذي هو عدّاد حلقة `for`. سيخزن ملف مفاتيح الإجابات للاختبار `capitalsquiz<N>.txt` في ملف باسم `capitalsquiz_answers<N>.txt`.

في كل دورة في حلقة `for` ستبدل قيمة `{quizNum + 1}` في كل من:

```
f'capitalsquiz{quizNum + 1}.txt'
```

و

```
f'capitalsquiz_answers{quizNum + 1}.txt'
```

برقم فريد، كما ستكون أسماء الملفات لأول مرور باسم `capitalsquiz1.txt` و`capitalsquiz_answers1.txt`. ستنشأ هذه الملفات حين استدعاء الدالة `open()` في السطرين ❶ و❷ حين تمرير الوسيط الثاني `'w'` لفتح الملفات في وضع الكتابة.

العبارات `write()` في القسم ❸ تكتب ترويسة الاختبار التي يجب على الطالب أن يملأها؛ ثم سننشِئ قائمة عشوائية فيها أسماء الولايات الأمريكية باستخدام الدالة `random.shuffle()` في السطر ❹، التي تغير ترتيب القيم في أي قائمة تمرر إليها.

9.5.3 الخطوة 3: إنشاء خيارات الإجابة

علينا الآن توليد خيارات إجابات كل سؤال، والتي ستكون اختيار من متعدد من A إلى D، أي أننا سنحتاج إلى إنشاء حلقة `for` أخرى لتوليد محتوى لكل سؤال من الأسئلة الخميس. ثم ستكون هنالك حلقة `for` ثالثة داخلها لتوليد الإجابات المحتملة لكل سؤال.

```
#! python3
# randomQuizGenerator.py - إنشاء اختبارات مع أسئلة وإجابات عشوائية

# مع تخزين مفتاح الحل
--snip--
  # المرور على الولايات الخمسين وتوليد سؤال لكل منها
  for questionNum in range(50):
    # الحصول على الإجابات الصحيحة والخطأ
    ❶ correctAnswer = capitals[states[questionNum]]
    ❷ wrongAnswers = list(capitals.values())
    ❸ del wrongAnswers[wrongAnswers.index(correctAnswer)]
    ❹ wrongAnswers = random.sample(wrongAnswers, 3)
    ❺ answerOptions = wrongAnswers + [correctAnswer]
    ❻ random.shuffle(answerOptions)
    # TODO: كتابة السؤال والإجابات إلى ملف الاختبار
    # TODO: كتابة مفتاح الحل إلى ملف الحلول.
```

من السهل الحصول على الجواب الصحيحة، فهو القيمة المخزنة في القاموس `capitals` ❶. ستمر حلقة التكرار على جميع الولايات الموجودة ضمن قائمة `states` المرتبة عشوائيًا، من `states[0]` حتى `states[49]`، ويجد كل ولاية في `capitals` ثم يخزن الجواب الصحيح في المتغير `correctAnswer`.

عملية إنشاء قائمة بالإجابات الخطأ أصعب بقليل، عليك أولًا أن تنسخ جميع القيم في قاموس `capitals` ❷، ثم تحذف الجواب الصحيح ❸، ثم تأخذ ثلاث قيم عشوائية من القائمة ❹؛ وتسهل علينا الدالة `random.sample()` ذلك، وتأخذ وسيطين: الأول هو القائمة التي تريد الاختيار منها، والثاني هو عدد القيم التي تريدها.

ستكون القائمة النهائية هي الإضافة الصحيحة إضافةً إلى إجابات خطأ ثلاثة ❺، ثم علينا جعل الإجابات عشوائية ❻ كيلا تكون الإجابة D هي الإجابة الصحيحة دومًا.

9.5.4 الخطوة 4: كتابة المحتوى إلى ملفات الاختبارات والإجابات الصحيحة

كل ما بقي فعله هو كتابة السؤال إلى ملف الاختبار، وكتابة الإجابة إلى ملف الإجابات الصحيحة.

```
#! python3
# randomQuizGenerator.py - إنشاء اختبارات مع أسئلة وإجابات عشوائية

# مع تخزين مفتاح الحل
--snip--


  # المرور على الولايات الخمسين وتوليد سؤال لكل منها.
  for questionNum in range(50):


  --snip--


    # كتابة السؤال والإجابات إلى ملف الاختبار.
    quizFile.write(f'{questionNum + 1}. What is the capital of
{states[questionNum]}?\n')
  ❶ for i in range(4):
    ❷ quizFile.write(f"    {'ABCD'[i]}. { answerOptions[i]}\n")
    quizFile.write('\n')


    # كتابة مفتاح الحل إلى ملف الحلول.
  ❸ answerKeyFile.write(f"{questionNum + 1}.
{'ABCD'[answerOptions.index(correctAnswer)]}")
  quizFile.close()
  answerKeyFile.close()
```

تمـر حلقـة `for` على الأعـداد 0 إلى 3 وتكتب الجـواب في القائمـة `answerOptions` ❶، أما التعبير `'ABCD'[i]` في ❷ أن السلسـلة النصـية `'ABCD'` سـتعامل كمصـفوفة وسـتكون قيمهـا هي `'A'` ثم `'B'` ثم `'C'` ثم `'D'` وفقًا لدورة حلقة التكرار.

في السطر الأخير ❸ سيعثر التعبير `answerOptions.index(correctAnswer)` على فهـرس الإجابـة الصحيحة في قائمة الإجابات المعشوائية، ثم ستكون نتيجة التعبير البرمجي الآتي:

```
'ABCD'[answerOptions.index(correctAnswer)]
```

هي حرف الجواب الصـحيح الـذي سـيكتب في ملـف الإجابـات. بعـد أن تشـغل البرنـامج فسـيبدو الملـف `capitalsquiz1.txt` كما يلي، مع الانتباه إلى أن الناتج سيكون مختلفًا عما عندك لأننا نأخذ القيم عشوائيًا:

```
Name:

Date:

Period:


        State Capitals Quiz (Form 1)
1.What is the capital of West Virginia?
  A. Hartford
  B. Santa Fe
  C. Harrisburg
  D. Charleston
2. What is the capital of Colorado?
  A. Raleigh
  B. Harrisburg
  C. Denver
  D. Lincoln


--snip--
```

وسيبدو ملف `capitalsquiz_answers1.txt` كما يلي:

```
1. D
2. C
3. A
4. C
--snip--
```

9.6 مشروع: تحديث لمشروع الحافظة

لنعد كتابة مشروع الحافظة من الفصل السابع من هذه السلسلة حول كيفية معالجة النصوص، لنستعمل الوحدة `shelve`، إذ سيتمكن المستخدم من حفظ سلاسل نصية جديدة وتحميلها إلى الحافظة دون تعديل الشيفرة المصدرية للتطبيق.

سنسمي مشروعنا `mcb.pyw`، فاختصار `mcb` هو `multi-clipboard` أي الحافظة المتعددة، والامتداد `.pyw` يعني أن بايثون لن تظهر نافذة الطرفية حين تشغيل البرنامج (راجع الفصل الأول لمزيد من التفاصيل).

سيحفظ البرنامج المحتويات النصية للحافظة تحت كلمة مفتاحية معينة، فمثلًا لو شغلت `py mcb.pyw save spam` فستحفظ المحتويات الحالية للحافظة مع الكلمة المفتاحية `spam`، ويمكن إعادة تحميل النص إلى الحافظة مجددًا بتشغيل `py mcb.pyw spam`، وإذا نسي المستخدم ما الكلمات المفتاحية التي استخدمها فيمكنه تشغيل `py mcb.pyw list` لنسخ قائمة فيها جميع الكلمات المفتاحية إلى الحافظة.

هذه هي آلية عمل المشروع:

1. التحقق من الوسيط الممرر عبر سطر الأوامر.
2. إذا كان save فستحفظ محتويات الحافظة إلى الكلمة المفتاحية المرسلة.
3. إذا كان list فستنسخ جميع الكلمات المفتاحية إلى الحافظة.
4. خلاف ذلك، سينسخ النص المرتبط بالكلمة المفتاحية إلى الحافظة.

هذا يعني أن على البرنامج:

- قراءة وسائط سطر الأوامر من `sys.argv`.
- القراءة والكتابة إلى الحافظة.
- حفظ وتحميل ملف `shelf`.

إذا كنت تستخدم ويندوز، فيمكنك ببساطة تشغيل السكربت من نافذة Run بإنشاء ملف `mcb.bat` فيه المحتوى الآتي:

```
@pyw.exe C:\Python34\mcb.pyw %*
```

9.6.1 الخطوة 1: البنية الأساسية وضبط عملية الحفظ والتحميل

لنبدأ بكتابة الهيكل الأساسي للسكربت مع بعض التعليقات وضبط أساسي لعملية الحفظ والتحميل:

```
#! python3

# mcb.pyw - حفظ وتحميل نصوص إلى الحافظة

❶ # Usage: py.exe mcb.pyw save <keyword> - حفظ الحاوية إلى keyword

#        py.exe mcb.pyw <keyword> -تحميل محتويات keyword إلى الحاوية

#        py.exe mcb.pyw list - تحميل كل الكلمات المفتاحية إلى الحاوية


❷ import shelve, pyperclip, sys

❸ mcbShelf = shelve.open('mcb')

# TODO: حفظ محتويات الحاوية

# TODO: إظهار كل الكلمات المفتاحية والمحتوى
```

من الشائع أن نضع معلومات الاستخدام في تعليقات في أول الملف ❶، ففي حال نسيت طريقة تشغيل البرنامج فيمكنك أن تلقي نظرة سريعة على هذه التعليقات لتتذكر طريقة الاستخدام، ثم استوردنا الوحدات اللازمة ❷ فعملية النسخ واللصق إلى الحاوية تحتاج إلى الوحدة `pyperclip`، وقراءة وسائط سطر الأوامر تحتاج إلى الوحدة `sys`، وستفيدنا الوحدة `shelve`: فكل مرة يحفظ فيها المستخدم محتويات الحافظة فسنكتبها إلى ملف، وحينما يريد تحميل نص إلى الحافظة فسنفتح الملف ونقرأه منه، وسنسمي هذا الملف باسم `mcb`.

9.6.2 الخطوة 2: حفظ محتويات الحافظة مع كلمة مفتاحية

يسلك البرنامج سلوكًا مختلفًا اعتمادًا على ما يريده المستخدم: حفظ محتويات الحاوية مع كلمة مفتاحية، أو تحميل نص إلى الحاوية، أو عرض جميع الكلمات المفتاحية. لنعالج الآن أول حالة:

```
#! python3

# mcb.pyw - حفظ وتحميل نصوص إلى الحافظة.

--snip--
```

```
# حفظ محتوى الحافظة.
❶ if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
    ❷ mcbShelf[sys.argv[2]] = pyperclip.paste()
elif len(sys.argv) == 2:
    ❸ # TODO: تحميل المحتوى وعرض الكلمات المفتاحية.

mcbShelf.close()
```

إذا كان أول وسيط من وسائط سطر الأوامر (الذي يكون دومًا بالفهرس 1 من قائمة `sys.argv`) هو `'save'` ❶ فإن الوسيط الثاني هو الكلمة المفتاحية التي يجب حفظ محتويات الحافظة إليها، والتي ستستخدم كمفتاح مع `mvbShelf`، وستكون قيمة هذا المفتاح هي محتويات الحافظة الحالية ❷.

أما إذا كان هنالك وسيط واحد ممرر من سطر الأوامر فهذا يعني أنه إما `'list'` أو كلمة مفتاحية نريد تحميل المحتوى النصي المرتبط بها إلى الحافظة. اترك تعليقًا الآن وسنكتب الشيفرة لاحقًا ❸.

9.6.3 الخطوة 3: تحميل الكلمات المفتاحية أو محتوى إحدى الكلمات

لنكتب الآن الشيفرة المناسبة للحالتين الباقيتين:

تحميل النص المرتبط بإحدى الكلمات المفتاحية إلى الحافظة، أو نسخ قائمة بجميع الكلمات المفتاحية إلى الحافظة:

```
#! python3
# mcb.pyw - حفظ وتحميل نصوص إلى الحافظة
--snip--
# حفظ محتوى الحافظة
if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
    mcbShelf[sys.argv[2]] = pyperclip.paste()
elif len(sys.argv) == 2:
    # تحميل المحتوى وعرض الكلمات المفتاحية.
    ❶ if sys.argv[1].lower() == 'list':
```

```
    ❷ pyperclip.copy(str(list(mcbShelf.keys())))

    elif sys.argv[1] in mcbShelf:

    ❸ pyperclip.copy(mcbShelf[sys.argv[1]])


mcbShelf.close()
```

إذا كان هنالك وسيط واحد في سطر الأوامر، فلنتأكد إن كان `'list'` ❶، فإذا كان كذلك فستنسخ سلسلة نصية تمثل قائمةً من الكلمات المفتاحية إلى الحافظة ❷، ويمكن للمستخدم أن يلصقها في أي محرر نصي أمامه ليقرأها.

فيما عدا ذلك فسنفترض أن الوسيط الممرر من سطر الأوامر هو كلمة مفتاحية، وإذا كانت الكلمة المفتاحية موجودة في `mcbShelf` فسنحمل القيمة إلى الحافظة ❸.

هذا كل ما يلزم لتطوير البرنامج! قد تختلف خطوات التشغيل اعتمادًا على نظام التشغيل الذي تستعمله، راجع الفصل الأول لتفاصيل.

من غير المنطقي أن تغير الشيفرة المصدرية لبرنامجك كلما احتجت إلى تحديث البيانات، لأن المستخدم العادي لا يرتاح لتعديل الشيفرات البرمجية؛ وكل مرة تعدل فيها البرنامج فأنت تخاطر بحدوث مشاكل جديدة وعلل لم تنتبه إليها. لذا من المهم فصل البيانات اللازمة لتشغيل التطبيق عن التطبيق نفسه، مما يجعل برامجك سهلة الاستخدام من الآخرين وتقلل احتمال حدوث مشاكل فيها.

9.7 أسئلة للتدريب

1. إلى ماذا ينسب المسار النسبي؟
2. كيف يبدأ المسار المطلق؟
3. ما هي نتيجة `Path('C:/Users') / 'Al'` في ويندوز؟
4. ما هي نتيجة `C:/Users' / 'Al''` في ويندوز؟
5. ماذا تفعل الدالتان `os.getcwd()` و `os.chdir()`؟
6. ما هما المجلدان . و .. ؟
7. ما هو اسم المجلد واسم الملف في المسار `C:\olive\eggs\spam.txt`؟
8. ما هي الأنماط التي يمكن تمريرها إلى الدالة `open()`؟
9. ماذا يحدث لو فتحنا ملفًا للكتابة `'w'` وكان موجودًا مسبقًا؟

10. ما الفرق بين التابعين `read()` و `readline()`؟

11. ما هو نوع البيانات الذي يحاول «الرف shelf» محاكاته؟

9.8 مشاريع للتدريب

لكي تتدرب، اكتب برامج لتنفيذ المهام الآتية.

9.8.1 توسعة تطبيق الحافظة

وسِّع تطبيق الحافظة الذي أنشأناه في هذا الفصل وأضف إليه الأمر delete الذي يحذف كلمة مفتاحية ومحتوياتها من «الرف». بعد ذلك أضف الأمر `delete` دون أي وسائط إضافية الذي سيؤدي إلى حذف جميع الكلمات المفتاحية.

9.8.2 لعبة Mad Libs

أنشِئ لعبة Mad Libs التي تقرأ ملفًا نصية وتسمح للمستخدم بإضافة النص الذي يريده في أي مكان تظهر فيه الكلمات المفتاحية ADJECTIVE و NOUN و ADVERB و VERB في الملف النصي. فمثلًا سيبدو الملف النصي كما يلي:

```
The ADJECTIVE panda walked to the NOUN and then VERB. A nearby NOUN
was
unaffected by these events.
```

سيقرأ البرنامج هذا الملف، ويبحث عن تلك الكلمات، ويطلب من المستخدم أن يدخل بديلًا عنها:

```
Enter an adjective:
silly
Enter a noun:
chandelier
Enter a verb:
screamed
Enter a noun:
pickup truck
```

وستكون نتيجة البرنامج هي:

```
The silly panda walked to the chandelier and then screamed. A nearby
pickup
truck was unaffected by these events.
```

ويجب أن تطبع هذه النتيجة على الشاشة وتحفظ في ملف نصي جديد باسم مناسب.

9.8.3 البحث عبر التعابير النمطية

اكتب برنامجًا يفتح جميع ملفات `txt` النصية في مجلد، ويبحث فيها عن أي سطر يحتوي على مطابقة لتعبير نمطي ضبطه المستخدم. يجب أن تعرض النتائج على الشاشة.

9.9 الخلاصة

تنظم الملفات في مجلدات (التي تسمى في بعض أنظمة التشغيل directory أي دليل)، ويصف مسارُ موقعَ الملف، ولكل برنامج يعمل في حاسوبك ما يسمى بمجلد العمل الحالي، مما يسمح بتحديد مسارات نسبية تبدأ من المجلد الحالي بدلًا من كتابة المسار الكامل للملف الذي يسمى أيضًا بالمسار المطلق. توفر الوحدتان `pathlib` و `os.path` عددًا من الدوال لإجراء عمليات على مسارات الملفات.

يمكن أن تتفاعل برامجك مباشرةً مع محتويات الملفات النصية، فالدالة `open()` تفتح الملفات للقراءة، وتستطيع أن تحصل على محتواها كسلسلة نصية واحدة كبيرة عبر التابع `read()` أو كقائمة من السلاسل النصية عبر التابع `readlines()`. يمكن أن تستخدم الدالة `open()` لفتح الملفات في وضع الكتابة أو الإضافة لإنشاء ملفات نصية جديدة أو الإضافة على ملفات موجودة مسبقًا، على التوالي.

تعلمنا في الفصول السابقة كيفية استخدام الحافظة لتخزين وتحميل كمية كبيرة من النصوص إلى برامجنا بدل من كتابتها يدويًا، لكننا الآن قادرون على جعل برامجنا تقرأ الملفات من ذاكرة التخزين أو الكتابة إليها، وهذا تحسين كبير عما سبق، وتكون وسيطة التخزين أكثر استقرارًا من الحافظة. سنتعلم في الفصل القادم كيفية التعامل مع الملفات نفسها، عبر نسخها وحذفها وإعادة تسميتها ونقلها ...إلخ.

أكبر موقع توظيف عن بعد في العالم العربي

أعلن عن وظائفك الشاغرة وابدأ بتوظيف
أفضل الكفاءات عن بعد

أضف وظيفة الآن

10. تنظيم الملفات باستخدام بايثون

تعلّمنا في فصلنا السابق كيفية إنشاء ملفات جديدة والكتابة فيها باستخدام لغة بايثون Python، ويمكن لبرامجك أيضًا تنظيم الملفات الموجودة مسبقًا على القرص الصلب. لا بد أنك جرّبتَ تصفح مجلدٍ مليء بالعشرات أو المئات أو حتى الآلاف من الملفات ونسخها أو إعادة تسميتها أو نقلها أو ضغطها جميعًا يدويًا، أو جرّبتَ مهامًا أخرى مثل المهام التالية:

إنشاء نُسخ من جميع ملفات PDF الموجودة في كل مجلدٍ فرعي من مجلدٍ ما.

إزالة الأصفار البادئة في أسماء الملفات لكل ملفٍ في مجلد يضم مئات الملفات المسمّاة مثلًا `spam001.txt` و `spam002.txt` وإلخ.

ضغط محتويات عدة مجلدات في ملف مضغوط ZIP واحد، والذي قد يكون نظام نسخ احتياطي بسيط.

يمكنك تنفيذ كافة هذه المهام المملة آليًا في شيفرة بايثون البرمجية، فإذا برمجتَ حاسوبك لإجراء هذه المهام، فيمكنك تحويله إلى محرر ملفات سريع في العمل ولا يرتكب أخطاءً أبدًا.

من المفيد رؤية امتداد الملف (مثل `.txt` و `.pdf` و `.jpg` وإلخ) بسرعة عند بدء العمل مع الملفات، إذ يُرجَّح أن يعرض متصفح ملفاتك الامتدادات تلقائيًا في نظامي ماك macOS ولينكس Linux، ولكن قد تكون امتدادات الملفات مخفية افتراضيًا في نظام ويندوز Windows، لذا يمكنك إظهار الامتدادات من خلال الانتقال إلى قائمة ابدأ Start، ثم لوحة التحكم Control Panel، ثم المظهر وإضفاء طابع شخصي Appearance and Personalization، ثم خيارات مستكشف الملفات Folder Options. ألغِ تحديد خانة الاختيار إخفاء ملحقات الملفات لأنواع الملفات المعروفة Hide extensions for known file types في تبويب عرض View ضمن الإعدادات المتقدمة Advanced Settings.

10.1 وحدة shutil

تحتوي وحدة `shutil` (التي هي اختصار لأدوات الصدفة المساعدة Shell Utilities) على دوالٍ تتيح لك نسخ الملفات ونقلها وإعادة تسميتها وحذفها في برامج بايثون الخاصة بك، ولكن يجب أولًا أن تستخدم التعليمة `import shutil` لاستخدام هذه الدوال.

10.1.1 نسخ الملفات والمجلدات

توفّر وحدة `shutil` دوالًا لنسخ الملفات والمجلدات الكاملة، حيث سيؤدي استدعاء الدالة `shutil.copy(source, destination)` إلى نسخ الملف من مسار المصدر `source` إلى المجلد الموجود في مسار الوِجهة `destination`، إذ يمكن أن يكون كلٌّ من `source` و `destination` سلاسلًا نصية أو كائنات `Path`. إذا كان `destination` اسم ملف، فسنستخدمه بوصفه اسمًا جديدًا للملف المنسوخ. تعيد هذه الدالة سلسلة نصية أو كائن `Path` للملف المنسوخ.

أدخِل مثلًا ما يلي في الصدفة التفاعلية Interactive Shell لترى كيفية عمل الدالة `shutil.copy()`:

```
>>> import shutil, os
>>> from pathlib import Path
>>> p = Path.home()
❶ >>> shutil.copy(p / 'spam.txt', p / 'some_folder')
'C:\\Users\\Al\\some_folder\\spam.txt'
❷ >>> shutil.copy(p / 'eggs.txt', p / 'some_folder/eggs2.txt')
WindowsPath('C:/Users/Al/some_folder/eggs2.txt')
```

ينسخ استدعاء الدالة `shutil.copy()` الأول الملف الموجود في `C:\Users\Al\spam.txt` إلى المجلد `C:\Users\Al\some_folder`، وتكون القيمة المُعادة هي مسار الملف المنسوخ، ولاحظ استخدام اسم الملف `spam.txt` الأصلي لاسم الملف المنسوخ الجديد عند تحديد مجلدٍ بوصفه الوِجهة ❶. ينسخ استدعاء الدالة `shutil.copy()` الثاني ❷ الملف الموجود في `C:\Users\Al\eggs.txt` إلى المجلد `C:\Users\Al\some_folder`، ولكنه يعطي الملف المنسوخ الاسم `eggs2.txt`.

تنسخ الدالة `shutil.copy()` ملفًا واحدًا، وتنسخ الدالة `shutil.copytree()` مجلدًا كاملًا مع جميع المجلدات والملفات الموجودة فيه، حيث يؤدي استدعاء الدالة `shutil.copytree(source, destination)` إلى نسخ المجلد الموجود في مسار المصدر `source` مع جميع ملفاته ومجلداته الفرعية إلى

المجلد الموجود في مسار الوِجهة destination، إذ تكون المعاملات source و destination سلاسلًا نصية. تعيد هذه الدالة سلسلة نصية تمثّل مسار المجلد المنسوخ.

لندخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import shutil, os
>>> from pathlib import Path
>>> p = Path.home()
>>> shutil.copytree(p / 'spam', p / 'spam_backup')
WindowsPath('C:/Users/Al/spam_backup')
```

ينشئ استدعاء الدالة shutil.copytree() السابق مجلدًا جديدًا بالاسم spam_backup الذي يحتوي على محتوى المجلد spam الأصلي نفسه، وبالتالي سنحصل بأمان على نسخة احتياطية من المجلد spam.

10.1.2 نقل وإعادة تسمية الملفات والمجلدات

سيؤدي استدعاء الدالة shutil.move(source, destination) إلى نقل الملف أو المجلد الموجود في مسار المصدر source إلى مسار الوجهة destination، وسيعيد سلسلة نصية للمسار المطلق الخاص بالموقع الجديد.

إذا أشار مسار الوجهة destination إلى مجلد، فسيُنقَل ملف المصدر source إلى الوجهة destination ويحتفظ باسم الملف الحالي. لندخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import shutil
>>> shutil.move('C:\\beef.txt', 'C:\\eggs')
'C:\\eggs\\beef.txt'
```

لنفترض أن المجلد الذي اسمه eggs موجودٌ مسبقًا في المجلد C:\، فسيمثّل استدعاء الدالة shutil.move() نقلَ الملف C:\beef.txt إلى المجلد C:\eggs. إذا كان الملف beef.txt موجودًا مسبقًا في المجلد C:\eggs، فسيُكتَب فوقه، لذا يجب عليك توخي الحذر عند استخدام الدالة move() لأنه من السهل الكتابة فوق الملفات عن طريق الخطأ باستخدام هذه الطريقة.

يمكن لمسار الوِجهة destination أيضًا تحديد اسم الملف، حيث سننقل ملف المصدر source ونعيد تسميته في المثال التالي:

```
>>> shutil.move('C:\\beef.txt', 'C:\\eggs\\new_beef.txt')
'C:\\eggs\\new_beef.txt'
```

يمثّل السطر السابق نقل الملف `C:\beef.txt` إلى المجلد `C:\eggs` وإعادة تسميته بالاسم `new_beef.txt`.

نفترض في المثالين السابقين وجود المجلد `eggs` في المجلد `C:\`، ولكن إن لم يوجَد المجلد `eggs`، فستعيد الدالة `move()` تسمية الملف `beef.txt` إلى ملفٍ اسمه `eggs`.

```
>>> shutil.move('C:\\beef.txt', 'C:\\eggs')
'C:\\eggs'
```

لم تتمكّن الدالة `move()` من العثور على مجلدٍ بالاسم `eggs` في المجلد `C:\`، وبالتالي تفترض أن الوجهة `destination` يجب أن تحدد اسم ملفٍ وليس اسم مجلد، لذلك أُعيدت تسمية الملف النصي `beef.txt` إلى `egg` (ملف نصي بدون امتداد الملف `.txt`)، وربما ليس هذا ما أردته. يمكن أن يكون ذلك خطأً يصعب اكتشافه في برامجك، لأن استدعاء الدالة `move()` يمكن أن يفعل شيئًا قد يكون مختلفًا تمامًا عمّا تتوقعه، وهذا سبب آخر لتوخي الحذر عند استخدام الدالة `move()`.

أخيرًا، يجب أن تكون المجلدات التي تشكّل الوِجهة موجودة فعليًا، وإلّا فسترمي شيفرة بايثون استثناءً. لندخِل الآن ما يلي في الصدفة التفاعلية:

```
>>> shutil.move('spam.txt', 'c:\\does_not_exist\\eggs\\meat')
Traceback (most recent call last):
 --snip--
FileNotFoundError: [Errno 2] No such file or directory: 'c:\
\does_not_exist\\
eggs\\meat'
```

تبحث شيفرة بايثون عن المجلدين `eggs` و `meat` ضمن المجلد `does_not_exist`، ولكنها لا تعثر على هذا المجلد، لذا لا يمكنها نقل الملف `spam.txt` إلى المسار الذي حدّدته.

10.1.3 حذف الملفات والمجلدات نهائيًا

يمكنك حذف ملف واحد أو مجلد واحد فارغ باستخدام دوال تابعة للوحدة `os`، ولكن يمكنك حذف مجلدٍ وجميع محتوياته باستخدام الوحدة `shutil`، وهذه الدوال هي:

سيؤدي استدعاء الدالة `os.unlink(path)` إلى حذف الملف الموجود في المسار `path`.

سيؤدي استدعاء الدالة `os.rmdir(path)` إلى حذف المجلد الموجود في المسار `path`، ولكن يجب أن يكون هذا المجلد خاليًا من أي ملفات أو مجلدات.

سيؤدي استدعاء الدالة `shutil.rmtree(path)` إلى إزالة المجلد الموجود في المسار `path`، وستُحذَف جميع الملفات والمجلدات الموجودة ضمنه.

كن حذرًا عند استخدام هذه الدوال في برامجك، إذ من الجيد أن تشغّل برنامجك أولًا مع تعليق هذه الاستدعاءات وإضافة استدعاءات الدالة `print()` لإظهار الملفات التي ستُحذَف. إليك فيما يلي برنامج بايثون الذي يهدف إلى حذف الملفات التي لها امتداد الملف `.txt`، ولكن يوجد به خطأ مطبعي يؤدي إلى حذف ملفات `.rxt` بدلًا من ذلك:

```
import os
from pathlib import Path
for filename in Path.home().glob('*.rxt'):
  os.unlink(filename)
```

إذا كان لديك أيّ ملفات مهمة تنتهي بالامتداد `.rxt` فستُحذَف نهائيًا عن طريق الخطأ، لذا يجب أولًا أن تشغّل البرنامج كما يلي:

```
import os
from pathlib import Path
for filename in Path.home().glob('*.rxt'):
  #os.unlink(filename)
  print(filename)
```

لاحظ أننا علّقنا استدعاء الدالة `os.unlink()`، لذا تجاهلته شيفرة بايثون، وستطبع اسم الملف المحذوف فقط، حيث سيؤدي تشغيل هذه النسخة من البرنامج أولًا إلى إظهار أنك طلبت من البرنامج عن طريق الخطأ حذف ملفات `.rxt` بدلًا من ملفات `.txt`.

تأكّد من عمل البرنامج بالطريقة الصحيحة، ثم احذف سطر التعليمة `print(filename)` وألغِ التعليق عند سطر التعليمة `os.unlink(filename)`، ثم شغّل البرنامج مرة أخرى لحذف الملفات فعليًا.

10.1.4 الحذف الآمن باستخدام وحدة send2trash

تحذف الدالة `shutil.rmtree()` المُدمَجة مع لغة بايثون الملفات والمجلدات نهائيًا، ولكن قد يكون استخدامها خطيرًا، لذا توجد طريقة أفضل بكثير لحذف الملفات والمجلدات، وهي استخدام الوحدة `send2trash` الخارجية. يمكنك تثبيت هذه الوحدة من خلال تشغيل الأمر `pip install --user send2trash` من نافذة الطرفية Terminal.

يُعَد استخدام الوحدة `send2trash` أكثر أمانًا من دوال الحذف العادية الخاصة بلغة بـايثون، لأنهـا سترسـل المجلدات والملفات إلى سلة المهملات أو سلة المحذوفات الخاصـة بحاسـوبك بـدلًا من حـذفها نهائيًـا. إذا أدّى خطأٌ ما في برنامجك إلى حذفٍ شيءٍ باستخدام الوحـدة `send2trash` ولا تريـد حذفـه، فيمكنـك اسـتعادته من سلة المحذوفات لاحقًا.

أدخِل مثلًا ما يلي في الصدفة التفاعلية بعد تثبيت الوحدة `send2trash`:

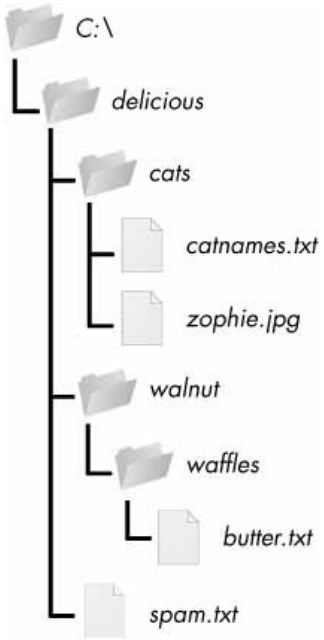
```
>>> import send2trash
>>> beefFile = open('beef.txt', 'a')    # إنشاء الملف
>>> beefFile.write('Beef is not a vegetable.')
25
>>> beefFile.close()
>>> send2trash.send2trash('beef.txt')
```

يجب دائمًا أن تستخدم الدالة `send2trash.send2trash()` لحذف الملفات والمجلدات، ولكن بـالرغم من أن إرسال الملفات إلى سلة المحذوفات يتيح لك استعادتها لاحقًـا، إلّا أنـه لن يـؤدي إلى تحريـر مسـاحةٍ من القرص الصلب كما يفعل الحذف النهائي، لذا إذا أردتَ أن يحرّر برنامجك مساحةً من القرص الصلب، فاسـتخدم دوال الوحدتين `os` و `shutil` لحذف الملفـات والمجلـدات. لاحـظ أن الدالـة `send2trash()` يمكنهـا إرسـال الملفات إلى سلة المحذوفات فقط، ولا يمكنها سحب الملفات منها.

10.2 المرور على شجرة مجلدات

لنفترض أنك تريد إعادة تسمية كل ملف في مجلدٍ ما وكل ملفٍ في كل مجلدٍ فرعي من هذا المجلـد، وهـذا يعني أنك تريد المرور على شجرة المجلدات، والتفاعل مع جميـع الملفـات أثنـاء المـرور عليهـا. قـد تكـون كتابـة برنامجٍ لذلك أمرًا صعبًا، ولكن توفّر بايثون الدالة `os.walk()` للتعامل مع هذه العملية نيابةً عنك.

أولًا، لنلقِ نظرة على المجلد `C:\delicious` ومحتوياته كما هو موضح في الشكل التالي:



الشكل 43: مثال لمجلد يحتوي
على ثلاثة مجلدات وأربعة
ملفات

إليك فيما يلي مثال لبرنامج يستخدم الدالة `os.walk()` مع شجرة المجلدات من الشكل السابق:

```
import os

for folderName, subfolders, filenames in os.walk('C:\\delicious'):
  print('The current folder is ' + folderName)

  for subfolder in subfolders:
    print('SUBFOLDER OF ' + folderName + ': ' + subfolder)

  for filename in filenames:
    print('FILE INSIDE ' + folderName + ': '+ filename)

  print('')
```

نمرّر قيمة سلسلة نصية واحدة تمثّل مسار المجلد إلى الدالة `os.walk()` التي يمكنك استخدامها في تعليمة حلقة `for` للمرور على شجرة المجلدات، حيث يشبه ذلك استخدام الدالة `range()` للمرور على مجالٍ من الأعداد، ولكن ستعيد الدالة `os.walk()` ثلاث قيم في كل تكرار من هذه الحلقة، وهذه القيم هي:

- سلسلة نصية تمثّل اسم المجلد الحالي.
- قائمة من السلاسل النصية التي تمثّل المجلدات الموجودة في المجلد الحالي.
- قائمة من السلاسل النصية التي تمثّل الملفات الموجودة في المجلد الحالي.

> **ملاحظة**: المجلد الحالي هو المجلد الخاص بالتكرار الحالي لحلقة `for`، ولم تغيّر الدالة `os.walk()` مجلد العمل الحالي للبرنامج.

يمكنك اختيار اسم المتغير `i` في شيفرة `for i in range(10):`، ويمكنك أيضًا اختيار أسماء المتغيرات للقيم الثلاث المذكورة سابقًا، ولكننا سنستخدم أسماء المتغيرات `foldername` و `subfolders` و`filenames` في أغلب الأحيان.

إذا شغّلنا البرنامج، فسينتج ما يلي:

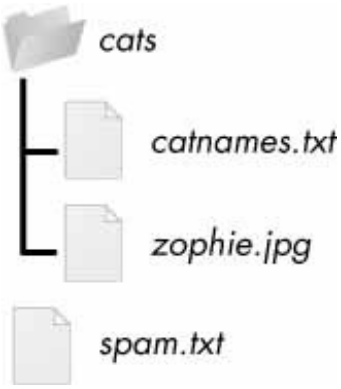
```
The current folder is C:\delicious
SUBFOLDER OF C:\delicious: cats
SUBFOLDER OF C:\delicious: walnut
FILE INSIDE C:\delicious: spam.txt

The current folder is C:\delicious\cats
FILE INSIDE C:\delicious\cats: catnames.txt
FILE INSIDE C:\delicious\cats: zophie.jpg

The current folder is C:\delicious\walnut
SUBFOLDER OF C:\delicious\walnut: waffles

The current folder is C:\delicious\walnut\waffles
FILE INSIDE C:\delicious\walnut\waffles: butter.txt.
```

تعيد الدالة `os.walk()` قوائمًا من السلاسل النصية التي تمثّل المتغيرات `subfolder` و `filename`، لذا يمكنك استخدام هذه القوائم في حلقات `for` الخاصة بها. ضع شيفرتك البرمجية مكان استدعاءات الدالة `print()`، أو احذف حلقتي `for` إن لم تكن بحاجة إليهما.

10.3 ضغط الملفات باستخدام الوحدة zipfile

قـد تكـون على درايـة بالملفـات المضـغوطة ZIP ذات امتـداد الملـف `.zip`، والـتي يمكنهـا الاحتفـاظ بالمحتويات المضغوطة للعديد من الملفات الأخرى، حيث يؤدي ضغط الملف إلى تقليل حجمه، ويُعَد ذلك أمـرًا مفيدًا عند نقله عبر الإنترنت. يمكن أن يحتوي ملف ZIP أيضًا على ملفـات ومجلـدات فرعيـة متعـددة، لـذا تُعَـد طريقة سهلة لحَزم ملفات متعددة في ملف واحد، ويمكن بعد ذلك مثلًا إرفـاق هـذا الملـف الـذي يسـمّى ملـف الأرشفة Archive File مع رسالة بريد إلكتروني.

يمكن لبرامج بايثون الخاص بك إنشـاء ملفـات ZIP وفتحهـا (أو فـك ضـغطها Extract) باسـتخدام الـدوال الموجودة في الوحدة `zipfile`. لنفترض أن لديك ملف ZIP بالاسم `example.zip` ويحتوي على المحتويـات الموضّحة في الشكل التالي:



الشكل 44: محتويات الملف example.zip

يمكنـك تنزيـل هـذا الملـف من موقـع nostarch أو المتابعـة باسـتخدام ملـف ZIP موجـود مسـبقًا على حاسوبك.

10.3.1 قراءة الملفات المضغوطة ZIP

يمكنك قراءة محتويات ملف مضغوط ZIP من خلال إنشـاء كـائن `ZipFile` أولًا (لاحـظ الأحـرف الكبـيرة Z وF)، حيث تتشابه كائنات `ZipFile` مع كائنات `File` التي تعيدها الدالة `open()`، فهي قيم يتفاعـل البرنـامج من خلالها مع الملف. ننشئ كائن `ZipFile` من خلال استدعاء الدالة `zipfile.ZipFile()` وتمرير سلسـلة نصية تمثّل اسم ملف `.ZIP` إليها. لاحظ أن `zipfile` هو اسم وحدة بايثون، وأن `ZipFile()` هو اسم الدالة.

أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import zipfile, os


>>> from pathlib import Path
>>> p = Path.home()
>>> exampleZip = zipfile.ZipFile(p / 'example.zip')
>>> exampleZip.namelist()
['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']
>>> spamInfo = exampleZip.getinfo('spam.txt')
>>> spamInfo.file_size
13908
>>> spamInfo.compress_size
3828
❶ >>> f'Compressed file is {round(spamInfo.file_size / spamInfo
.compress_size, 2)}x smaller!'
)
'Compressed file is 3.63x smaller!'
>>> exampleZip.close()
```

يحتوي كائن `ZipFile` على التابع `namelist()` الذي يعيد قائمةً من السلاسل النصية التي تمثّل جميع الملفات والمجلدات الموجودة في ملف ZIP، حيث يمكن تمرير هذه السلاسل النصية إلى التابع `getinfo()` لإعادة كائن `ZipInfo` لهذا الملف المحدّد. تمتلك كائنات `ZipInfo` سماتها Attributes الخاصة مثل السمات `file_size` و `compress_size` بالبايتات، والتي تحتوي على أعداد صحيحة لحجم الملف الأصلي وحجم الملف المضغوط على التوالي. يمثل كائن `ZipInfo` ملف أرشفة كامل، ولكن يحمل كائن `ZipInfo` أيضًا معلومات مفيدة حول ملف واحد في ملف الأرشفة.

يحسب الأمر الموجود في التعليمة ❶ مدى كفاءة ضغط الملف `example.zip` من خلال قسمة حجم الملف الأصلي على حجم الملف المضغوط ويطبع هذه المعلومات.

10.3.2 فك ضغط ملفات ZIP

يفك التابع `extractall()` الخاص بكائنات `ZipFile` ضغط جميع الملفات والمجلدات من ملف مضغوط ZIP إلى مجلد العمل الحالي.

```
>>> import zipfile, os
>>> from pathlib import Path
>>> p = Path.home()
>>> exampleZip = zipfile.ZipFile(p / 'example.zip')
>>> exampleZip.extractall()
>>> exampleZip.close()
```

❶

يؤدي تشغيل الشيفرة البرمجية السابقة إلى فَك ضغط محتويات الملف `example.zip` في المجلد `C:\`. يمكنك اختياريًا تمرير اسم مجلد إلى التابع `extractall()` لفك ضغط الملفات في مجلد آخر مختلفٍ عن مجلد العمل الحالي، وإذا كان المجلد الذي مرّرناه إلى التابع `extractall()` غير موجود، فسيُشَأ هذا المجلد، فمثلًا إذا وضعتَ الاستدعاء `exampleZip.extractall('C:\\delicious')` مكان الاستدعاء ❶، فسعمل الشيفرة البرمجية على فك ضغط الملفات من الملف `example.zip` إلى المجلد `C:\delicious` الذي أنشأناه.

يفك التابع `extract()` الخاص بكائنات `ZipFile` ضغط ملفٍ واحد من الملف المضغوط ZIP. تابع مثال الصدفة التفاعلية بما يلي:

```
>>> exampleZip.extract('spam.txt')
'C:\\spam.txt'
>>> exampleZip.extract('spam.txt', 'C:\\some\\new\\folders')
'C:\\some\\new\\folders\\spam.txt'
>>> exampleZip.close()
```

يجب أن تتطابق السلسلة النصية التي تمررها إلى التابع `extract()` مع إحدى السلاسل النصية الموجودة في القائمة التي يعيدها التابع `namelist()`، ويمكنك اختياريًا تمرير وسيطٍ ثانٍ إلى التابع `extract()` لفك ضغط الملف في مجلد آخر مختلف عن مجلد العمل الحالي، حيث إذا كان هذا الوسيط الثاني مجلدًا غير موجودٍ بعد، فستنشِئ شيفرة بايثون هذا المجلد. القيمة التي يعيدها التابع `extract()` هي المسار المطلق الذي فكينا ضغط الملف فيه.

10.3.3 إنشاء ملفات ZIP والإضافة إليها

يمكنك إنشاء ملفات ZIP المضغوطة من خلال فتح كائن `ZipFile` في وضع الكتابة مع تمرير `'w'` كوسيط ثانٍ، حيث يشبه ذلك فتح ملفٍ نصي في وضع الكتابة من خلال تمرير `'w'` إلى الدالة `open()`.

إذا مرّرتَ مسارًا إلى التابع `write()` مع كائن `ZipFile`، ستضغط شيفرة بايثون الملف الموجود في هذا المسار وتضيفه إلى ملف ZIP. الوسيط الأول للتابع `write()` هو سلسلة نصية تمثّل اسم الملف المراد إضافته، والوسيط الثاني هو معامل يمثّل نوع عملية الضغط، إذ يخبر هذا النوع الحاسوبَ بالخوارزمية التي يجب أن يستخدمها لضغط الملفات، حيث يمكنك دائمًا ضبط هذه القيمة على `zipfile.ZIP_DEFLATED` التي تحدّد خوارزمية الضغط Deflate التي تعمل على جميع أنواع البيانات.

أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import zipfile
>>> newZip = zipfile.ZipFile('new.zip', 'w')
>>> newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)
>>> newZip.close()
```

ستؤدي الشيفرة البرمجية السابقة إلى إنشاء ملف ZIP جديد بالاسم new.zip، حيث يحتوي هذا الملف على محتويات مضغوطة للملف `spam.txt`.

ضع في بالك أن وضع الكتابة سيؤدي إلى مسح جميع المحتويات الموجودة مسبقًا في ملف ZIP كما هو الحال مع الكتابة في الملفات. إذا أردتَ ببساطة إضافة ملفات إلى ملف ZIP موجود مسبقًا، فمرّر `'a'` كوسيطٍ ثانٍ إلى الدالة `zipfile.ZipFile()` لفتح ملف ZIP في وضع الإلحاق Append Mode.

10.4 تطبيق عملي: إعادة تسمية الملفات إلى تواريخ ذات نمط مختلف

لنفترض أن مديرك في العمل يرسل إليك عبر البريد الإلكتروني آلاف الملفات ذات تواريخ النمط الأمريكي (MM-DD-YYYY) الموجودة في أسماء هذه الملفات ويريد إعادة تسميتها إلى تواريخ النمط الأوروبي (DD-MM-YYYY)، إذ قد يستغرق إنجاز هذه المهمة المملة يدويًا وقتًا طويلًا، إذًا لنكتب برنامجًا ينّفذ هذه المهمة نيابةً عنك.

إليك الخطوات التي يفعلها هذا البرنامج:

- البحث في جميع أسماء الملفات الموجودة في مجلد العمل الحالي عن التواريخ ذات النمط الأمريكي.
- إعادة تسمية الملف مع التبديل بين الشهر واليوم لجعله على النمط الأوروبي عند العثور على أحد هذه الملفات.

وبالتالي يجب أن تطبّق شيفرتك البرمجية الخطوات التالية:

- إنشاء تعبير نمطي Regex يمكنه تحديد نمط النص للتواريخ ذات النمط الأمريكي.
- استدعاء التابع `os.listdir()` للعثور على جميع الملفات الموجودة في مجلد العمل.
- المرور ضمن حلقة على جميع أسماء الملفات باستخدام التعبير النمطي للتحقق من احتوائه على تاريخ.
- إذا احتوى اسم الملف على تاريخ، فيجب إعادة تسمية الملف باستخدام الدالة `shutil.move()`.
- افتح نافذةً جديدة في محرّرك من أجل إنشاء ملف جديد للمشروع واحفظ شيفرتك البرمجية بالاسم `renameDates.py`.

10.4.1 الخطوة الأولى: إنشاء تعبير نمطي للتواريخ ذات النمط الأمريكي

سيحتاج الجزء الأول من البرنامج إلى استيراد الوحدات الضرورية وإنشاء تعبير نمطي يمكنه تحديد تواريخ النمط الأمريكي MM-DD-YYYY. ستذكرك تعليقات `TODO` في النهاية بما تبقى لتكتبه في هذا البرنامج، حيث كتبناها ليسهل عليك العثور عليها باستخدام ميزة البحث `Ctrl-F` في محرّر Mu. اجعل شيفرتك البرمجية تبدو كما يلي:

```
#! python3
# renameDates.py - إعادة تسمية أسماء الملفات ذات تنسيق التاريخ الأمريكي إلى MM-DD-YYYY
# تنسيق التاريخ الأوروبي DD-MM-YYYY

❶ import shutil, os, re


# إنشاء تعبير نمطي يطابق الملفات ذات تنسيق التاريخ الأمريكي
❷ datePattern = re.compile(r"""^(.*?) # كل النص قبل التاريخ
    ((0|1)?\d)-                       # رقم أو رقمين للشهر
    ((0|1|2|3)?\d)-                   # رقم أو رقمين لليوم
    ((19|20)\d\d)                     # أربعة أرقام للسنة
    (.*?)$                            # كل النص بعد التاريخ
     """, re.VERBOSE❸)

# TODO: المرور ضمن حلقة على الملفات الموجودة في مجلد العمل
```

```
    # TODO: تخطي الملفات التي تكون بدون تاريخ

    # TODO: الحصول على الأجزاء المختلفة من اسم الملف

    # TODO: تشكيل اسم الملف على النمط الأوروبي

    # TODO: الحصول على مسارات الملفات الكاملة والمطلقة

    # TODO: إعادة تسمية الملفات
```

تعلّمنا في هذا الفصل أنه يمكن استخدام الدالة `shutil.move()` لإعادة تسمية الملفات، ووسطاؤها هي اسم الملف المراد إعادة تسميته واسم الملف الجديد، ويجب استيراد الوحدة `shutil` ❶ بسبب وجود هذه الدالة فيها.

يجب تحديد الملفات التي تريد إعادة تسميتها قبل إعادة تسميتها، إذ يجب إعادة تسمية أسماء الملفات التي لها تواريخ مثل `spam4-4-1984.txt` و `01-03-2014eggs.zip`، بينما يمكن تجاهل أسماء الملفات التي لا تحتوي على تواريخ مثل `littlebrother.epub`.

يمكنك استخدام تعبير نمطي لتحديد هذا النمط، لذا استدعِ التابع `re.compile()` لإنشاء كائن `Regex`❷ بعد استيراد الوحدة `re` في البداية.

سيسمح تمرير القيمة `re.VERBOSE` للوسيط الثاني ❸ بوجود المسافات البيضاء والتعليقات في السلسلة النصية للتعبير النمطي لجعلها أكثر قابلية للقراءة.

تبدأ السلسلة النصية للتعبير النمطي بالمحارف `^(.*?)` لمطابقة أيّ نصٍ موجود في بداية اسم الملف الذي قد يأتي قبل التاريخ. تطابق المجموعة `((0|1)?\d)` الشهر، حيث يمكن أن يكون الرقم الأول إما `0` أو `1`، وبالتالي يطابق التعبير النمطي القيمةَ `12` للشهر 12 ويطابق القيمة `02` للشهر الثاني، حيث يكون هذا الرقم اختياريًا أيضًا بحيث يمكن أن يكون الشهر `04` أو `4` للشهر الرابع. مجموعة اليوم هي `((0|1|2|3)?\d)` وتتبع منطقًا مشابهًا لمجموعة الشهر، حيث تُعَد القيم `3` و `03` و `31` أرقامًا صالحة للأيام. لاحظ أن هذا التعبير النمطي سيقبل بعض التواريخ غير الصالحة مثل `2022-31-4` و `2023-29-2` و `2024-15-0`، إذ تحتوي التواريخ على الكثير من الحالات الخاصة التي يمكن أن نخطئ بها بسهولة، ولكن يعمل التعبير النمطي في هذا البرنامج جيدًا بما فيه الكفاية للتبسيط.

تُعَد السنة 1885 سنةً صالحة، ولكن يمكنك فقط البحث عن السنوات في القرن العشرين أو الحادي والعشرين، مما سيؤدي إلى منع برنامجك من مطابقة أسماء الملفات التي لها تنسيق مشابه للتاريخ ولكنها لا تمثّل تواريخًا مثل `10-10-1000.txt` عن طريق الخطأ.

أخيرًا، يتطابق الجزء `(.*?)$` من التعبير النمطي مع أيّ نص يأتي بعد التاريخ.

10.4.2 الخطوة الثانية: تحديد أجزاء التاريخ من أسماء الملفات

يجب بعد ذلك أن يمر البرنامج ضمن حلقة على قائمة السلاسل النصية لأسماء الملفات التي يعيدها التابع `os.listdir()`، وأن يطابقها مع التعبير النمطي.

يجب تخطي أيّ ملفات لا تتضمن تاريخًا في اسمها، وسيُخزَّن النص المطابق في عدة متغيرات بالنسبة لأسماء الملفات التي تحتوي على تاريخ فيها.

املأ المهام `TODO` الثلاثة الأولى في برنامجك بالشيفرة البرمجية التالية:

```
#! python3
# renameDates.py - إعادة تسمية أسماء الملفات ذات تنسيق التاريخ الأمريكي MM-DD-YYYY إلى
# تنسيق التاريخ الأوروبي DD-MM-YYYY

--snip--

# المرور ضمن حلقة على الملفات الموجودة في مجلد العمل
for amerFilename in os.listdir('.'):
  mo = datePattern.search(amerFilename)

  # تخطي الملفات التي تكون بدون تاريخ
❶ if mo == None:
  ❷ continue

❸ # الحصول على الأجزاء المختلفة من اسم الملف
  beforePart = mo.group(1)
  monthPart  = mo.group(2)
  dayPart    = mo.group(4)
  yearPart   = mo.group(6)
  afterPart  = mo.group(8)
--snip--
```

إذا كانت قيمة كائن `Match` الذي يعيده التابع `search()` هي `None` ❶، فلن يتطابق اسم الملف الموجود في المتغير `amerFilename` مع التعبير النمطي، وستتخطى تعليمة `continue` ❷ بقية الحلقة وتنتقل إلى اسم الملف التالي، وإلّا فستُخزَّن السلاسل النصية المختلفة المطابِقة مع مجموعات التعبير النمطي في متغيرات بالاسم `beforePart` و `monthPart` و `dayPart` و `yearPart` و `afterPart` ❸. ستُستخدَم السلاسل النصية الموجودة في هذه المتغيرات لتشكيل اسم الملف على النمط الأوروبي في الخطوة التالية.

أبقِ أرقام المجموعة سهلة الاستخدام من خلال محاولة قراءة التعبير النمطي من البداية وحساب كل مرة يظهر فيها قوس مفتوح. اكتب مخططًا تفصيليًا للتعبير النمطي دون التفكير في الشيفرة البرمجية، حيث يمكن أن يساعدك المثال التالي في تصوّر هذه المجموعات:

```
datePattern = re.compile(r"""^(1) # كل النص قبل التاريخ
  (2 (3) )-                       # رقم أو رقمين للشهر
  (4 (5) )-                       # رقم أو رقمين لليوم
  (6 (7) )                        # أربعة أرقام للسنة
  (8)$                            # كل النص بعد التاريخ
  """, re.VERBOSE)
```

تمثل الأرقام من 1 إلى 8 في المثال السابق المجموعات في التعبير النمطي الذي كتبته. يمكن أن يمنحكَ إنشاء مخطط تفصيلي للتعبير النمطي باستخدام الأقواس وأرقام المجموعات فقط فهمًا أوضح لتعبيرك النمطي قبل الانتقال إلى بقية البرنامج.

10.4.3 الخطوة الثالثة: تشكيل اسم الملف الجديد وإعادة تسمية الملفات

جرّب سَلسَلة السلاسل النصية الموجودة في المتغيرات من الخطوة السابقة مع التاريخ ذي النمط الأوروبي، حيث يأتي اليوم قبل الشهر. املأ المهام `TODO` الثلاثة المتبقية في برنامجك بالشيفرة البرمجية التالية:

```
#! python3
# renameDates.py - إعادة تسمية أسماء الملفات ذات تنسيق التاريخ الأمريكي MM-DD-YYYY إلى
# DD-MM-YYYY تنسيق التاريخ الأوروبي


--snip--
  # تشكيل اسم الملف على النمط الأوروبي
```

```
❶ euroFilename = beforePart + dayPart + '-' + monthPart + '-' +
yearPart + afterPart
  # الحصول على مسارات الملفات الكاملة والمطلقة
  absWorkingDir = os.path.abspath('.')
  amerFilename = os.path.join(absWorkingDir, amerFilename)
  euroFilename = os.path.join(absWorkingDir, euroFilename)

  # إعادة تسمية الملفات
❷ print(f'Renaming "{amerFilename}" to "{euroFilename}"...')
❸ #shutil.move(amerFilename, euroFilename)    # ألغِ التعليق بعد الاختبار
```

خزّن السلسلة النصية المتسلسلة في متغير بالاسم `euroFilename` ❶، ثم مرّر اسم الملف الأصلي الموجود في المتغير `amerFilename` والمتغير `euroFilename` الجديد إلى الدالة `shutil.move()` لإعادة تسمية الملف ❸.

يتضمن هذا البرنامج تعليقًا على استدعاء الدالة `shutil.move()`، حيث يطبع أسماء الملفات التي ستُعاد تسميتها ❷.

يمكن أن يتيح لك تشغيل البرنامج بهذه الطريقة أولًا التحقق من إعادة تسمية الملفات بطريقة صحيحة، ثم يمكنك إلغاء تعليق استدعاء الدالة `shutil.move()` وتشغيل البرنامج مرة أخرى لإعادة تسمية الملفات فعليًا.

10.4.4 أفكار لبرامج مماثلة

هناك العديد من الأسباب الأخرى التي قد تجعلك ترغب في إعادة تسمية عدد كبير من الملفات مثل:

- إضافة بادئة إلى بداية اسم الملف مثل إضافة `spam_` لإعادة تسمية الملف `eggs.txt` إلى الاسم `spam_eggs.txt`.
- تغيير أسماء الملفات التي تحتوي على تواريخ ذات نمط أوروبي إلى تواريخ ذات نمط الأمريكي.
- حذف الأصفار من أسماء الملفات مثل `spam0042.txt`.

10.5 تطبيق عملي: إنشاء نسخة احتياطية لمجلد في ملف مضغوط ZIP

لنفترض أنك تعمل على مشروع تحتفظ بملفاته في مجلد بالاسم `C:\AlsPythonBook`، ولا بد أنك قلق بشأن فقدان عملك، لذا سترغب في إنشاء "لقطات" من ملفات ZIP للمجلد بأكمله، إذ قد ترغب في الاحتفاظ

بنسخ مختلفة، لذلك يجب أن يزيد اسم ملف ZIP في كل مرة تنشئ فيها نسخة مثل AlsPythonBook_1.zip و AlsPythonBook_2.zip و AlsPythonBook_3.zip وإلخ.

يمكنك إنجاز ذلك يدويًا، ولكنه أمر مزعج إلى حدٍ ما، وقد تخطئ في ترقيم أسماء ملفات ZIP، فمن الأسهل تشغيل برنامج ينجز هذه المهمة المملة نيابةً عنك.

افتح نافذة جديدة في محرّرك لإنشاء ملف جديد لهذا المشروع واحفظه بالاسم backupToZip.py.

10.5.1 الخطوة الأولى: اكتشاف اسم الملف المضغوط ZIP

سنضع الشيفرة البرمجية الخاصة بهذا البرنامج في دالة اسمها backupToZip()، حيث سيؤدي ذلك إلى تسهيل نسخ الدالة ولصقها في برامج بايثون الأخرى التي تحتاج إليها.

ستُستدعَى هذه الدالة لإجراء النسخ الاحتياطي في نهاية البرنامج، لذا اجعل برنامجك يبدو كما يلي:

```
#! python3
# backupToZip.py - نسخ مجلد كامل ومحتوياته في ملف ZIP يزيد اسمه بمقدار واحد في كل مرة يُنسَخ فيها

❶ import zipfile, os
def backupToZip(folder):
    # إنشاء نسخة احتياطية من محتويات المجلد بالكامل في ملف ZIP
    folder = os.path.abspath(folder)    # التأكد من أن المجلد مسار مطلق

    # اكتشاف اسم الملف الذي يجب أن تستخدمه هذه الشيفرة البرمجية بناءً على الملفات الموجودة مسبقًا
    ❷ number = 1
    ❸ while True:
        zipFilename = os.path.basename(folder) + '_' + str(number) +
'.zip'
        if not os.path.exists(zipFilename):
            break
        number = number + 1

    ❹ # TODO: إنشاء ملف مضغوط ZIP
```

```
    # TODO: المرور على شجرة المجلدات بأكملها وضغط الملفات الموجودة في كل مجلد
    print('Done.')
backupToZip('C:\\delicious')
```

أضِف أولًا سطر Shebang (الذي يبدأ بالسلسلة النصية `#!`) مع وصف ما يفعله البرنامج، ثم استورد وحدات `zipfile` و `os` ❶. عرّف بعد ذلك دالة بالاسم `backupToZip()`، حيث تأخذ هذه الدالة معاملًا واحدًا فقط هو `folder`، والذي هو سلسلة نصية تمثّل مسارًا إلى المجلد الذي يجب نسخ محتوياته احتياطيًا. ستحدّد هذه الدالة اسم الملف المُستخدَم لملف ZIP الذي ستنشئه، ثم تنشئ هذه الدالة الملف، وتمر على المجلد `folder`، وتضيف كلًا من المجلدات الفرعية والملفات إلى ملف ZIP. اكتب تعليقات `TODO` لهذه الخطوات في الشيفرة البرمجية لتذكير نفسك بإنجازها لاحقًا ❹.

يستخدم الجزء الأول -ويمثّل تسمية الملف ZIP- الاسم الأساسي للمسار المطلق للمجلد `folder`. إذا كان المجلد الذي ننسخه احتياطيًا هو `C:\delicious`، فيجب أن يكون اسم الملف ZIP هو `delicious_N.zip`، حيث أن `N = 1` هي المرة الأولى التي نشغّل فيها البرنامج، أما `N = 2` هي المرة الثانية وإلخ.

يمكنك تحديد ما يجب أن تكون عليه قيمة `N` من خلال التحقق مما إذا كان الملف `delicious_1.zip` موجودًا مسبقًا، ثم التحقق مما إذا كان الملف `delicious_2.zip` موجودًا مسبقًا وإلخ. استخدم متغيرًا اسمه `number` لتمثيل `N` ❷، واستمر في زيادته ضمن الحلقة التي تستدعي التابع `os.path.exists()` للتحقق من وجود الملف ❸. سيؤدي العثور على أول اسم ملف غير موجود إلى كسر الحلقة باستخدام التعليمة `break`، لأنها عثرت على اسم الملف المضغوط الجديد.

10.5.2 الخطوة الثانية: إنشاء ملف مضغوط ZIP جديد

لننشئ الآن ملف ZIP، لذا اجعل برنامجك يبدو كما يلي:

```
#! python3
# backupToZip.py - نسخ مجلد كامل ومحتوياته في ملف ZIP يزيد اسمه بمقدار واحد في كل مرة يُنسَخ فيها
--snip--
  while True:
    zipFilename = os.path.basename(folder) + '_' + str(number) + '.zip'
    if not os.path.exists(zipFilename):
      break
```

```
    number = number + 1

# إنشاء ملف مضغوط ZIP
print(f'Creating {zipFilename}...')
❶ backupZip = zipfile.ZipFile(zipFilename, 'w')

# TODO: المرور على شجرة المجلدات بأكملها وضغط الملفات الموجودة في كل مجلد
print('Done.')
backupToZip('C:\\delicious')
```

خزّنا اسم ملف ZIP الجديد في المتغير `zipFilename`، ويمكننا الآن استدعاء الدالة `zipfile.ZipFile()` لإنشاء ملف ZIP فعليًا ❶. تأكد من تمرير `'w'` كوسيطٍ ثانٍ لهذه الدالة لفتح الملف ZIP في وضع الكتابة.

10.5.3 الخطوة الثالثة: المرور على شجرة المجلدات والإضافة إلى ملف ZIP

يجب الآن أن تستخدم الدالة `os.walk()` لسرد كل ملف موجود في المجلد ومجلداته الفرعية، لذا اجعل برنامجك يبدو كما يلي:

```
#! python3
# backupToZip.py - نسخ مجلد كامل ومحتوياته في ملف ZIP يزيد اسمه بمقدار واحد في كل مرة يُنسَخ فيها
--snip--
    # المرور على شجرة المجلدات بأكملها وضغط الملفات الموجودة في كل مجلد
❶ for foldername, subfolders, filenames in os.walk(folder):
        print(f'Adding files in {foldername}...')
        # إضافة المجلد الحالي إلى ملف ZIP
❷      backupZip.write(foldername)

        # إضافة كافة الملفات الموجودة في هذا المجلد إلى ملف ZIP
❸      for filename in filenames:
            newBase = os.path.basename(folder) + '_'
```

```
        if filename.startswith(newBase) and filename.endswith('.zip'):
            continue    # لا تنشئ نسخة احتياطية من ملفات ZIP الاحتياطية
        backupZip.write(os.path.join(foldername, filename))
    backupZip.close()
    print('Done.')

backupToZip('C:\\delicious')
```

يمكنك استخدام الدالة `os.walk()` في حلقة `for` ❶، حيث ستعيد في كل تكرار اسم المجلد الحالي لهذا التكرار والمجلدات الفرعية الموجودة في هذا المجلد وأسماء الملفات الموجودة في هذا المجلد. يُضاف المجلد إلى ملف ZIP ❷ في حلقة `for`، ويمكن لحلقة `for` المتداخلة المرور على كل اسم ملف في القائمة `filenames` ❸، ويُضاف كل منها إلى ملف ZIP باستثناء ملفات ZIP الاحتياطية التي أنشأناها مسبقًا.

سينتج ما يلي عند تشغيل هذا البرنامج:

```
Creating delicious_1.zip...
Adding files in C:\delicious...
Adding files in C:\delicious\cats...
Adding files in C:\delicious\waffles...
Adding files in C:\delicious\walnut...
Adding files in C:\delicious\walnut\waffles...
Done.
```

سيضع هذا البرنامج جميع الملفات الموجودة في المجلد `C:\delicious` في ملف ZIP بالاسم `delicious_2.zip` في المرة الثانية لتشغيله وهكذا.

10.5.4 أفكار لبرامج مماثلة

يمكنك الاستفادة من فكرة المرور على شجرة المجلدات وإضافة الملفات إلى ملفات الأرشفة ZIP المضغوطة في العديد من البرامج الأخرى، فمثلًا يمكنك كتابة البرامج التي تنجز المهام التالية:

- المرور على شجرة المجلدات وأرشفة الملفات التي لها امتدادات محددة فقط مثل `.txt` أو `.py`.
- المرور على شجرة المجلدات وأرشفة جميع الملفات باستثناء ملفات `.txt` و `.py`.
- البحث عن المجلد في شجرة المجلدات الذي يحتوي على أكبر عدد من الملفات أو المجلد الذي يستخدم أكبر مساحة على القرص الصلب.

10.6 أسئلة للتدريب

1. ما الفرق بين الدالتين `shutil.copy()` و `shutil.copytree()`؟
2. ما هي الدالة المُستخدَمة لإعادة تسمية الملفات؟
3. ما هو الفرق بين دوال الحذف في وحدتي `send2trash` و `shutil`؟
4. تحتوي كائنات `ZipFile` على التابع `close()` مثل التابع `close()` الخاص بكائنات `File`، فما هو التابع الخاص بكائنات `ZipFile` المكافئة للتابع `open()` الخاص بكائنات `File`؟

10.7 مشاريع للتدريب

حاول كتابة البرامج التي تؤدي المهام التي سنوضّحها فيما يلي لكسب خبرة عملية أكبر.

10.7.1 برنامج لإنشاء نسخة انتقائية للملفات من شجرة المجلدات

اكتب برنامجًا يمر على شجرة المجلدات ويبحث عن الملفات التي لها امتداد ملف محدّد (مثل `.pdf` أو `.jpg`)، وانسخ هذه الملفات من أيّ موقع توجد فيه في مجلد جديد.

10.7.2 برنامج لحذف الملفات غير الضرورية

يمكن أن تَشغَل بعض الملفات أو المجلدات غير الضرورية والضخمة الجزء الأكبر من المساحة على قرص حاسوبك الصلب، فإذا أدرتَ تحرير مساحةٍ على حاسوبك، فستحصل على أقصى استفادة من خلال حذف أكبر عدد ممكن من الملفات غير المرغوب فيها، ولكن يجب أولًا العثور عليها.

اكتب برنامجًا يمر على شجرة المجلدات ويبحث عن الملفات أو المجلدات الكبيرة استثنائيًا مثل الملفات أو المجلدات التي يزيد حجم ملفها عن 100 ميجابايت، حيث يمكنك استخدام الدالة `os.path.getsize()` من وحدة `os` للحصول على حجم الملف. اطبع هذه الملفات مع مسارها المطلق على الشاشة.

10.7.3 ملء الفجوات في ترقيم أسماء الملفات

اكتب برنامجًا يبحث عن جميع الملفات ذات البادئة المُحدَّدة -مثل `spam001.txt` و `spam002.txt` وإلخ- في مجلدٍ واحد، ويحدّد هذا البرنامج موقع أيّ فجوات في الترقيم مثل وجود الملفين `spam001.txt` و`spam003.txt` دون وجود الملف `spam002.txt`، لذا اجعل البرنامج يعيد تسمية جميع الملفات اللاحقة من أجل سد هذه الفجوة. اكتب أيضًا برنامجًا آخر يمكنه إدراج فجوات في الملفات المرقّمة بحيث يمكن إضافة ملف جديد.

10.8 الخلاصة

يُحتمَل أنك تتعامل مع الملفات يدويًا باستخدام الفأرة ولوحة المفاتيح حتى لو كنت من مستخدمي الحاسوب ذوي الخبرة. تسهّل مستكشفات الملفات الحديثة العمل مع عددٍ من الملفات، ولكن ستحتاج في بعض الأحيان إلى تنفيذ مهمة قد تستغرق ساعات باستخدام مستكشف الملفات الخاص بحاسوبك.

توفّر وحدتا `os` و `shutil` دوالًا لنسخ الملفات ونقلها وإعادة تسميتها وحذفها، ولكن قد ترغب في استخدام وحدة `send2trash` عند حذف الملفات لنقلها إلى سلة المحذوفات أو سلة المهملات بدلًا من حذفها نهائيًا. يُفضَّل تعليق الشيفرة البرمجية التي تنسخ أو تنقل أو تعيد التسمية أو تحذف الملفات فعليًا عند كتابة البرامج التي تتعامل مع الملفات، ويجب إضافة استدعاء الدالة `print()` حتى تتمكّن من تشغيل البرنامج والتحقق مما سيفعله بالضبط.

يجب في أغلب الأحيان تنفيذ هذه العمليات على الملفات الموجودة في أحد المجلدات وعلى كل مجلد موجود في هذا المجلد وعلى كل مجلد موجود في تلك المجلدات وإلخ. تتولى الدالة `os.walk()` هذه الرحلة عبر المجلدات نيابةً عنك حتى تتمكّن من التركيز على ما يحتاج برنامجك إلى فعله مع الملفات الموجودة في هذه المجلدات.

تمنحك وحدة `zipfile` طريقةً لضغط وفك ضغط الملفات في أرشيفات `.ZIP` باستخدام لغة بايثون. تسهّل الوحدة `zipfile` -مع دوال معالجة الملفات الخاصة بوحدتي `os` و `shutil`- تجميعَ العديد من الملفات من أيّ مكان على قرص حاسوبك الصلب. يُعَد رفع هذه الملفات المضغوطة ZIP على مواقع الويب أو إرسالها بوصفها مرفقات في البريد الإلكتروني أسهلَ بكثير من العديد من الملفات المنفصلة.

وفرنا في هذه السلسلة من الفصول شيفرة برمجية يمكنك نسخها ولصقها في برنامجك، ولكن يُحتمَل ألّا تظهر بمظهرٍ مثالي في البداية. يركز الفصل التالي على بعض وحدات بايثون التي ستساعدك على تحليل برامجك وتنقيح أخطائها لتتمكّن من تشغيلها بصورة صحيحة بسرعة.

دورة تطوير التطبيقات
باستخدام لغة بايثون

مميزات الدورة

- بناء معرض أعمال قوي بمشاريع حقيقية
- وصول مدى الحياة لمحتويات الدورة
- تحديثات مستمرة على الدورة مجانًا
- شهادة معتمدة من أكاديمية حسوب
- إرشادات من المدربين على مدار الساعة
- من الصفر دون الحاجة لخبرة مسبقة

اشترك الآن

11. تنقيح الأخطاء Debugging عبر بايثون

تعرّفتَ في الفصول السابقة على معلومات كافية لكتابة برامج أكثر تعقيدًا، ولكنك ستعثر على أخطاء كثيرة فيها، لذا سنوضّح في هذا الفصل بعض الأدوات والتقنيات لإيجاد السبب الجذري للأخطاء في برنامجك لمساعدتك في إصلاح الأخطاء بسرعةٍ أكبر وبجهد أقل.

يمكن القول أن كتابة الشيفرة البرمجية تمثّل 90 % من عملية البرمجة، ولكن يمثّل تنقيح أخطاء هذه الشيفرة البرمجية نسبة 90 % أخرى من العمل، حيث سينفّذ حاسوبك ما تطلب منه فقط، إذ لن يقرأ أفكارك أو يطبّق ما تنوي فعله. يتسبّب المبرمجون وحتى المحترفون منهم بالأخطاء طوال الوقت، لذلك لا تشعر بالإحباط إذا واجه برنامجك مشكلةً ما.

يوجد عددٌ من الأدوات والتقنيات لتحديد ما تفعله شيفرتك البرمجية بالضبط ومكان حدوث الخطأ لحسن الحظ، حيث سنوضّح في هذا الفصل أولًا التسجيل Logging والتأكيدات Assertions، وهما ميزتان مساعدتان في اكتشاف الأخطاء في وقت مبكر، إذ يصبح إصلاح الأخطاء أسهل كلما اكتشفتَها مبكرًا. سنوضّح بعد ذلك كيفية استخدام منقّح الأخطاء Debugger الذي يمثّل ميزةً من ميزات المحرّر Mu، حيث ينفّذ منقّح الأخطاء البرنامج من خلال تنفيذ تعليمةٍ واحدة في كل مرة، مما يمنحك فرصة لفحص القيم في المتغيرات أثناء تشغيل شيفرتك البرمجية وتعقّب كيفية تغير هذه القيم عبر برنامجك بأكمله. يُعَد ذلك أبطأ بكثير من تشغيل البرنامج بأقصى سرعة، ولكنه مفيد لرؤية القيم الفعلية في البرنامج أثناء تشغيله بدلًا من استنتاج ما ستكون عليه القيم من الشيفرة المصدرية.

11.1 رفع الاستثناءات Raising Exceptions

ترفع لغة بايثون Python استثناءً عندما تحاول تنفيذ شيفرة برمجية غير صالحة، حيث تعرّفنا في الفصل الثالث على كيفية التعامل مع استثناءات بايثون باستخدام تعليمتي `try` و `except` حتى يتمكّن برنامجك من

التعافي من الاستثناءات المُتوقَّعة. يمكنك أيضًا رفع استثناءاتك الخاصة في شيفرتك البرمجية، إذ يمثّل رفع الاستثناء طريقةً لإيقاف تشغيل الشيفرة البرمجية في هذه الدالة ونقل تنفيذ البرنامج إلى التعليمة `except`.

تُرفَع الاستثناءات باستخدام التعليمة `raise` التي تتكون ممّا يلي:

- الكلمة المفتاحية `raise`.
- استدعاء الدالة `Exception()`.

سلسلة نصية تحتوي على رسالة خطأ مفيدة نمرِّرها إلى الدالة `Exception()`.

لندخِل مثلًا ما يلي في الصدفة التفاعلية Interactive Shell:

```
>>> raise Exception('This is the error message.')
Traceback (most recent call last):
 File "<pyshell#191>", line 1, in <module>
  raise Exception('This is the error message.')
Exception: This is the error message.
```

إن لم تُوجَد تعليمات `try` و `except` المغلِّفة للتعليمة `except` التي ترفع الاستثناء، فسيتعطل البرنامج ويعرض رسالة خطأ الاستثناء ببساطة.

تعرِف الشيفرة البرمجية التي تستدعي الدالة -وليس الدالة نفسها- كيفية التعامل مع الاستثناء، وهذا يعني أنك سترى التعليمة `raise` ضمن الدالة وتعليمتي `try` و `except` في الشيفرة البرمجية التي تستدعي هذه الدالة. افتح تبويبًا جديدًا في محرّرك لإنشاء ملف جديد، وأدخِل مثلًا الشيفرة البرمجية التالية واحفظ البرنامج بالاسم `boxPrint.py`:

```
def boxPrint(symbol, width, height):
  if len(symbol) != 1:
   ❶ raise Exception('Symbol must be a single character string.')
  if width <= 2:
   ❷ raise Exception('Width must be greater than 2.')
  if height <= 2:
   ❸ raise Exception('Height must be greater than 2.')
```

```
    print(symbol * width)
    for i in range(height - 2):
        print(symbol + (' ' * (width - 2)) + symbol)
    print(symbol * width)

for sym, w, h in (('*', 4, 4), ('O', 20, 5), ('x', 1, 3), ('ZZ', 3,
3)):
    try:
        boxPrint(sym, w, h)
    ❹ except Exception as err:
        ❺ print('An exception happened: ' + str(err))
```

اطّلع على تنفيذ هذا البرنامج، حيث عرّفنا الدالة `boxPrint()` التي تأخذ محرفًا وعرضًا وارتفاعًا، وتستخدم هذا المحرف لإنشاء صورة صغيرة لمربع له هذا العرض والارتفاع، وتطبع شكل هذا المربع على الشاشة.

لنفترض أننا نريد أن يكون هذا المحرف مفردًا، وأن يكون العرض والارتفاع أكبر من 2، فسنضيف تعليمات `if` لرفع الاستثناءات عند عدم استيفاء هذه المتطلبات. ستتعامل لاحقًا تعليمات `try`/`except` مع الوسطاء غير الصالحة عندما نستدعي الدالة `boxPrint()` مع وسطاء مختلفة.

يستخدم هذا البرنامج الصيغة `except Exception as err` للتعليمة `except` ❹. إذا أُعيد الكائن `Exception` من الدالة `boxPrint()` ❶ ❷ ❸، فستخزِّنه التعليمة `except` في متغير بالاسم `err`، ثم يمكننا تحويل الكائن `Exception` إلى سلسلة نصية من خلال تمريره إلى الدالة `str()` لإعطاء رسالة خطأ مألوفة للمستخدم ❺، وسيبدو الخرج كما يلي عند تشغيل البرنامج `boxPrint.py`:

```
****
*  *
*  *
****
OOOOOOOOOOOOOOOOOOOO
O                  O
O                  O
O                  O
OOOOOOOOOOOOOOOOOOOO
```

```
An exception happened: Width must be greater than 2.
An exception happened: Symbol must be a single character string.
```

يمكنك التعامل مع الأخطاء بأمان أكبر عبر تعليمات `try` و `except` بدلًا من ترك البرنامج يتعطل بأكمله.

11.2 الحصول على التعقب العكسي Traceback كسلسلة نصية

في حال واجهت شيفرة بايثون خطأً ما، فستعطي كنزًا من المعلومات عن الخطأ، حيث تسمّى هذه المعلومات بالتعقّب العكسي Traceback الذي يتضمّن رسالة الخطأ ورقم السطر الذي تسبّب في الخطأ وسلسلة استدعاءات الدوال التي أدّت إلى الخطأ، وتسمّى هذه السلسلة من الاستدعاءات بمكدس الاستدعاءات Call Stack.

افتح تبويبًا جديدًا في محرّر Mu لإنشاء ملف جديد، وأدخِل البرنامج التالي واحفظه بالاسم `errorExample.py`:

```
def spam():
  beef()
def beef():
  raise Exception('This is the error message.')
spam()
```

سيبدو الخرج كما يلي عند تشغيل البرنامج `errorExample.py`:

```
Traceback (most recent call last):
 File "errorExample.py", line 7, in <module>
  spam()
 File "errorExample.py", line 2, in spam
  beef()
 File "errorExample.py", line 5, in beef
  raise Exception('This is the error message.')
Exception: This is the error message.
```

يمكنك أن ترى من التعقّب العكسي السابق أن الخطأ حدث في السطر رقم 5 في الدالة `beef()`، وأتى هذا الاستدعاء للدالة `beef()` من السطر رقم 2 في الدالة `spam()` التي أُستدعيت بدورها في السطر رقم 7.

يمكن أن يساعدك مكدس الاستدعاءات في تحديد الاستدعاء الذي أدى إلى الخطأ في البرامج التي يمكن فيها استدعاء الدوال من أماكن متعددة.

تعرض لغة بايثون التعقّب العكسي عند عدم التعامل مع الاستثناء المرفوع، ولكن يمكنك أيضًا الحصول على التعقب العكسي بوصفه سلسلة نصية من خلال استدعاء الدالة `traceback.format_exc()`، حيث تكون هذه الدالة مفيدة إذا أردتَ الحصول على المعلومات من التعقب العكسي الخاص بالاستثناء ولكنك تريد أيضًا التعليمة `except` للتعامل مع الاستثناء بأمان. يجب استيراد الوحدة `traceback` الخاصة بلغة بايثون قبل استدعاء هذه الدالة.

يمكنك مثلًا كتابة معلومات التعقّب العكسي في ملف نصي والحفاظ على تشغيل البرنامج بدلًا من تعطّل برنامجك عند حدوث الاستثناء، حيث يمكنك إلقاء نظرة على الملف النصي لاحقًا عندما تكون مستعدًا لتنقيح أخطاء برنامجك.

أدخِل الآن ما يلي في الصدفة التفاعلية:

```
>>> import traceback
>>> try:
...         raise Exception('This is the error message.')
except:
...         errorFile = open('errorInfo.txt', 'w')
...         errorFile.write(traceback.format_exc())
...         errorFile.close()
...         print('The traceback info was written to errorInfo.txt.')
111
The traceback info was written to errorInfo.txt.
```

تُعَد القيمة `111` هي القيمة التي يعيدها التابع `write()`، حيث كُتِبت المحارف `111` في الملف، وكُتِب نص التعقّب العكسي في الملف `errorInfo.txt`:

```
Traceback (most recent call last):
 File "<pyshell#28>", line 2, in <module>
Exception: This is the error message.
```

سنتعلّم لاحقًا كيفية استخدام الوحدة `logging` التي تُعَد أكثر فعالية من مجرد كتابة معلومات الخطأ في ملفات نصية.

11.3 التأكيدات Assertions

يُعَد التأكيد Assertion فحص سلامةٍ للتأكد من أن شيفرتك البرمجية لا تفعل شيئًا خاطئًا، حيث يمكن إجراء عمليات التحقق من السلامة من خلال استخدام التعليمة `assert`، وإذا فشل التحقق من السلامة، فسيُرفَع الاستثناء `AssertionError`. تتكون التعليمة `assert` مما يلي في الشيفرة البرمجية:

- الكلمة المفتاحية `assert`.
- شرط (أيّ تعبير يمكن تقييمه بالقيمة `True` أو `False`).
- فاصلة.
- سلسلة نصية تُعرَض عندما تكون قيمة الشرط `False`.

تمثّل التعليمة `assert` التأكيد على أن الشرط صحيح، وإذا لم يكن الأمر كذلك، فلا بد من وجود خطأٍ في مكانٍ ما، لذا يجب إيقاف البرنامج مباشرةً. أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> ages = [26, 57, 92, 54, 22, 15, 17, 80, 47, 73]
>>> ages.sort()
>>> ages
[15, 17, 22, 26, 47, 54, 57, 73, 80, 92]
>>> assert
ages[0] <= ages[-1] # تأكيد أن العمر الأول <= العمر الأخير
```

تؤكّد التعليمة `assert` في المثال السابق على أن العنصر الأول في القائمة `ages` يجب أن يكون أصغر من العنصر الأخير أو يساويه، ويمثّل ذلك التحقق من السلامة، حيث إذا كانت الشيفرة البرمجية الموجودة في التابع `sort()` خالية من الأخطاء وأدت عملها، فسيكون التأكيد صحيحًا.

يُقيَّم التعبير `ages[0] <= ages[-1]` على أنه `True`، وبالتالي لن تفعل التعليمة `assert` شيئًا، ولكن لنتظاهر بوجود خطأ في شيفرتنا البرمجية، ولنفترض أننا استدعينا عن طريق الخطأ تابعَ القائمة `reverse()` بدلًا من تابع القائمة `sort()`، حيث سترفع التعليمة `assert` خطأ `AssertionError` مثلًا عندما ندخِل ما يلي في الصدفة التفاعلية:

```
>>> ages = [26, 57, 92, 54, 22, 15, 17, 80, 47, 73]
>>> ages.reverse()
>>> ages
[73, 47, 80, 17, 15, 22, 54, 92, 57, 26]
>>> assert ages[0] <= ages[-1] # تأكيد أن العمر الأول <= العمر الأخير
```

```
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
AssertionError
```

يجب ألّا تتعامل شيفرتك البرمجية مع التعليمة `assert` باستخدام تعليمات `try` و `except` على عكس الاستثناءات، حيث إذا فشلت التعليمة `assert`، فيُفترَض أن يتعطل برنامجك، وبالتالي ستختصر الوقت المنقضي من حدوث السبب الأصلي للخطأ إلى وقت ملاحظة الخطأ لأول مرة من خلال هذا الفشل السريع، مما سيؤدي إلى تقليل الشيفرة البرمجية التي يجب أن تتحقق منها قبل العثور على سبب الخطأ.

تُعَد التأكيدات مُخصَّصة لأخطاء المبرمج وليست خاصة بأخطاء المستخدم، إذ يجب أن تفشل التأكيدات فقط عندما يكون البرنامج قيد التطوير، ويجب ألّا يرى المستخدم أبدًا خطأ تأكيد في البرنامج النهائي، لذا يجب أن ترفع استثناءً بدلًا من اكتشافه باستخدام التعليمة `assert` بالنسبة للأخطاء التي يمكن أن يتعرض لها برنامجك كجزءٍ عادي من عمله مثل عدم العثور على ملف أو إدخال المستخدم بيانات غير صالحة، ويجب ألّا تستخدم تعليمات `assert` بدلًا من رفع الاستثناءات، لأنه يمكن للمستخدمين اختيار إيقاف التأكيدات. إذا شغّلتَ سكربت بايثون باستخدام الأمر `python -O myscript.py` بدلًا من الأمر `python myscript.py`، فستتخطّى شيفرة بايثون تعليمات `assert`، وقد يعطّل المستخدمون التأكيدات عندما يطورون برنامجًا ويحتاجون إلى تشغيله في بيئة الإنتاج التي تتطلب أداءً أعلى، بالرغم من أنهم في كثير من الحالات سيتركون التأكيدات مفعّلة حتى في ذلك الوقت.

لا تُعَد التأكيدات بديلًا عن الاختبار الشامل، فإذا ضبطنا القائمة `ages` في المثال السابق على القيمة `[20, 1, 2, 3]`، فلن تلاحظ تعليمة التأكيد `assert ages[0] <= ages[-1]` أن القائمة غير مرتبة، لأنها ترى أن العمر الأول أصغر من أو يساوي العمر الأخير، وهو الشيء الوحيد الذي تتحقق منه تعليمة التأكيد.

11.3.1 استخدام التأكيد في برنامج لمحاكاة إشارات المرور

لنفترض أنك تنشئ برنامجًا لمحاكاة إشارات المرور، حيث يكون هيكل البيانات الذي يمثّل إشارات التوقف عند التقاطع هو قاموس له المفاتيح `'ns'` و `'ew'` لإشارات التوقف التي تمثّل جهة الشمال-الجنوب وجهة الشرق-الغرب على التوالي.

ستكون القيم الموجودة في هذه المفاتيح إحدى السلاسل النصية `'green'` أو `'yellow'` أو `'red'`، حيث ستبدو الشيفرة البرمجية كما يلي:

```
market_2nd = {'ns': 'green', 'ew': 'red'}
mission_16th = {'ns': 'red', 'ew': 'green'}
```

يمثّل المتغيران السابقان تقاطعات شارع السوق Market Street والشارع الثاني 2nd Street، وشارع ميشن Mission Street والشارع السادس عشر 16th Street. نبدأ المشروع من خلال كتابة الدالة `switchLights()` التي تأخذ قاموسًا يمثّل التقاطع بوصفه وسيطًا وتبّدل بين الأضواء.

نعتقد في البداية أن الدالة `switchLights()` يجب أن تحوّل ببساطة كل ضوء إلى اللون التالي في السلسلة، حيث يجب أن تتغيّر جميع القيم `'green'` إلى القيمة `'yellow'`، ويجب أن تتغير قيم `'yellow'` إلى القيم `'red'`، ويجب أن تتغير القيم `'red'` إلى القيم `'green'`، إذ قد تبدو الشيفرة البرمجية لتطبيق هذه الفكرة كما يلي:

```
def switchLights(stoplight):
  for key in stoplight.keys():
    if stoplight[key] == 'green':
      stoplight[key] = 'yellow'
    elif stoplight[key] == 'yellow':
      stoplight[key] = 'red'
    elif stoplight[key] == 'red':
      stoplight[key] = 'green'

switchLights(market_2nd)
```

لا بد أنك رأيتَ مشكلة هذه الشيفرة البرمجية، ولكن لنفترض أنك كتبتَ بقية شيفرة المحاكاة التي يبلغ طولها آلاف الأسطر دون أن تلاحظ ذلك، حيث لن يتعطل البرنامج عندما تشغّل المحاكاة في النهاية، ولكن ستتعطّل سياراتك الافتراضية في البرنامج. لن يكون لديك أيّ فكرة عن مكان وجود الخطأ بما أنك كتبتَ بقية البرنامج فعليًا، إذ قد يكون الخطأ في الشيفرة البرمجية التي تحاكي السيارات أو في الشيفرة البرمجية التي تحاكي السائقين الافتراضيين، وبالتالي قد يستغرق الأمر ساعات لتعقّب الخطأ العكسي إلى الدالة `switchLights()`.

إذا أضفتَ تأكيدًا أثناء كتابة الدالة `switchLights()` للتحقّق من أن أحد الأضواء يكون دائمًا باللون الأحمر على الأقل، فيمكن تضمين ما يلي في نهاية الدالة:

```
assert 'red' in stoplight.values(), 'Neither light is red! ' +
str(stoplight)
```

سيتعطّل برنامجك مع ظهور رسالة الخطأ التالية باستخدام التأكيد السابق:

```
Traceback (most recent call last):
 File "carSim.py", line 14, in <module>
  switchLights(market_2nd)
 File "carSim.py", line 13, in switchLights
  assert 'red' in stoplight.values(), 'Neither light is red! ' +
str(stoplight)
❶ AssertionError: Neither light is red! {'ns': 'yellow', 'ew': 'green'}
```

السطر المهم في المثال السابق هو `AssertionError` ❶، حيث لا يُعَد تعطّل برنامجك أمرًا مثاليًا، ولكنه يشير مباشرةً إلى فشل التحقق من السلامة، إذ لا يحتوي أي من اتجاهي حركة المرور على ضوء أحمر، مما يعني أن حركة المرور يمكن أن تسير في كلا الاتجاهين من التقاطع. يمكنك توفير الكثير من جهد تنقيح الأخطاء مستقبلًا من خلال الفشل السريع في وقتٍ مبكر من تنفيذ البرنامج.

11.4 التسجيل Logging

إذا سبق لك أن وضعتَ التعليمة `print()` في شيفرتك البرمجية لإنتاج قيمة بعض المتغيرات أثناء تشغيل البرنامج، فلا بد أنك استخدمتَ صيغة تسجيلٍ لتنقيح أخطاء شيفرتك البرمجية، حيث يُعَد التسجيل طريقةً رائعة لفهم ما يحدث في برنامجك وترتيب حدوثه. تسهّل الوحدة `logging` في بايثون إنشاء سجلٍ للرسائل المُخصَّصة التي تكتبها، إذ توضّح هذه الرسائل وقت وصول تنفيذ البرنامج إلى استدعاء دالة التسجيل وتسرد المتغيرات التي حدّدتها في ذلك الوقت. بينما تشير رسالة السجل الناقصة إلى تخطي جزء من الشيفرة البرمجية وعدم تنفيذه مطلقًا.

11.4.1 استخدام الوحدة logging

يمكن تفعيل الوحدة `logging` لعرض رسائل السجل على شاشتك أثناء تشغيل البرنامج من خلال نسخ ما يلي إلى بداية برنامجك، ولكن ضمّن السطر Shebang الذي هو `#! python`:

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -  %
(levelname)
s -  %(message)s')
```

لا داعي للقلق كثيرًا بشأن كيفية عمل السطر السابق، ولكنه ينشئ الكائن `LogRecord` الذي يحتوي على معلومات حول حدثٍ ما عندما تسجّل بايثون هذا الحدث. تتيح لك الدالة `basicConfig()` الخاصة بالوحدة `logging` تحديدَ التفاصيل المتعلقة بكائن `LogRecord` الذي تريد رؤيته وكيفية عرض هذه التفاصيل.

لنفترض أنك كتبتَ دالةً لحساب عاملي Factorial عددٍ ما، حيث يكون عاملي العدد 4 في الرياضيات هو 4 × 3 × 2 × 1 أو القيمة 24 وعاملي العدد 7 هو 7 × 6 × 5 × 4 × 3 × 2 × 1 أو القيمة 5040.

افتح تبويبًا جديدًا في محرّرك لإنشاء ملف جديد وأدخِل الشيفرة البرمجية التالية التي تحتوي على خطأ، ولكنك ستدخِل أيضًا عددًا من رسائل السجل لمساعدة نفسك في اكتشاف الخطأ الذي يحدث، واحفظ البرنامج بالاسم `factorialLog.py`:

```
import logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s -  %
(levelname)s
-  %(message)s')
logging.debug('Start of program')

def factorial(n):
  logging.debug('Start of factorial(%s%%)'  % (n))
  total = 1
  for i in range(n + 1):
    total *= i
    logging.debug('i is ' + str(i) + ', total is ' + str(total))
  logging.debug('End of factorial(%s%%)'  % (n))
  return total

print(factorial(5))
logging.debug('End of program')
```

نستخدم الدالة `logging.debug()` عندما نريد طباعة معلومات السجل، وتستدعي الدالةُ `debug()` الدالةَ `basicConfig()` وستطبع سطرًا من المعلومات، حيث ستكون هذه المعلومات بالتنسيق الذي حددناه في الدالة `basicConfig()` وستتضمّن الرسائل التي مرّرناها إلى الدالة `debug()`.

يُعَد استدعاء الدالة `print(factorial(5))` جزءًا من البرنامج الأصلي، لذا ستُعرَض النتيجة حتى لو تعطّلت رسائل التسجيل.

يبدو خرج هذا البرنامج كما يلي:

```
2024-05-23 16:20:12,664 - DEBUG - Start of program
2024-05-23 16:20:12,664 - DEBUG - Start of factorial(5)
2024-05-23 16:20:12,665 - DEBUG - i is 0, total is 0
2024-05-23 16:20:12,668 - DEBUG - i is 1, total is 0
2024-05-23 16:20:12,670 - DEBUG - i is 2, total is 0
2024-05-23 16:20:12,673 - DEBUG - i is 3, total is 0
2024-05-23 16:20:12,675 - DEBUG - i is 4, total is 0
2024-05-23 16:20:12,678 - DEBUG - i is 5, total is 0
2024-05-23 16:20:12,680 - DEBUG - End of factorial(5)
0
2024-05-23 16:20:12,684 - DEBUG - End of program
```

تعيد الدالة `factorial()` القيمة `0` بوصفها ناتج عاملي العدد 5، وهذا ليس صحيحًا، حيث يجب أن تضرب حلقة `for` القيمة الموجودة في المتغير `total` بالأعداد من 1 إلى 5، ولكن توضّح رسائل السجل التي تعرضها الدالة `logging.debug()` أن المتغير `i` يبدأ من القيمة `0` بدلًا من القيمة `1`، وبالتالي تكون قيمة بقية التكرارات خاطئة للمتغير `total` أيضًا، لأن ضرب الصفر بأيّ شيء يساوي صفرًا.

توفّر رسائل التسجيل سلسلةً من مسارات التنقل التي قد تساعدك في معرفة متى بدأت الأمور تسوء.

غيّر السطر `for i in range(n + 1):` إلى `for i in range(1, n + 1):`، وشغّل البرنامج مرة أخرى، وسيبدو الخرج كما يلي:

```
2024-05-23 17:13:40,650 - DEBUG - Start of program
2024-05-23 17:13:40,651 - DEBUG - Start of factorial(5)
2024-05-23 17:13:40,651 - DEBUG - i is 1, total is 1
2024-05-23 17:13:40,654 - DEBUG - i is 2, total is 2
2024-05-23 17:13:40,656 - DEBUG - i is 3, total is 6
2024-05-23 17:13:40,659 - DEBUG - i is 4, total is 24
2024-05-23 17:13:40,661 - DEBUG - i is 5, total is 120
2024-05-23 17:13:40,661 - DEBUG - End of factorial(5)
120
2024-05-23 17:13:40,666 - DEBUG - End of program
```

يؤدي استدعاء الدالة `factorial(5)` إلى إعادة القيمة `120` الصحيحة، وتظهِر رسائل السجل ما يحدث ضمن الحلقة، مما يقودك إلى الخطأ مباشرةً.

يمكنك أن ترى أن استدعاءات الدالة `logging.debug()` لا تطبع السلاسل النصية المُمـرَّرة إليهـا فقـط، بل تطبع أيضًا العلامة الزمنية Timestamp والكلمة DEBUG.

11.4.2 لا تنقّح الأخطاء باستخدام الدالة ()print

تُعَد كتابة التعليمـتين `import logging` و `logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')` أمرًا صـعبًا بعض الشـيء، لـذا سترغب في استخدام استدعاءات الدالة `print()` بدلًا من ذلك، ولكن لا تنخدع بسهولة استخدام هـذه الدالـة، لأنك ستقضي كثيرًا من الوقت في إزالـة اسـتدعاءات الدالـة `print()` من شـيفرتك البرمجيـة لجميـع رسـائل السجل بعد الانتهاء من تنقيح الأخطاء، وقـد تزيـل أيضًـا عن طريـق الخطـأ بعض اسـتدعاءات الدالـة `print()` المُستخدَمة للرسائل التي ليس لها علاقة بالسجل.

يمكنك ملء برنامجك بالعدد الذي تريـده من رسـائل السـجل، ويمكنـك دائمًـا تعطيلهـا لاحقًـا عـبر إضـافة استدعاء واحد للدالة `logging.disable(logging.CRITICAL)`، إذ تسهّل الوحدة `logging` التبديل بين إظهار وإخفاء رسائل السجل على عكس الدالة `print()`.

تُعَد رسائل السجل خاصةً بالمبرمج وليست خاصة بالمستخدم، إذ لن يهتم المستخدم بمحتويات بعض قيم القاموس التي تحتاج إلى رؤيتها للمساعدة في تنقيح الأخطاء، لذا اسـتخدم رسـالة سـجل لـذلك، ولكن يجب أن تستخدم استدعاء الدالة `print()` بالنسبة للرسائل التي يـرغب المسـتخدم في رؤيتهـا مثـل "عـدم العثـور على ملف File not found" أو "قيمة إدخال غير صالحة لذا أدخِل قيمة عددية من فضلك Invalid input, please enter a number"، فلن ترغب في حرمان المستخدم من المعلومات المفيدة له بعد تعطيل رسائل السجل.

11.4.3 مستويات التسجيل

توفر مستويات التسـجيل طريقـة لتصـنيف رسـائل السـجل حسـب الأهميـة، إذ توجـد خمسـة مسـتويات للتسجيل في لغة بايثون التي سنوضـحها في الجـدول الآتي من الأقـل إلى الأكـثر أهميـة، حيث يمكن تسـجيل الرسائل على كل مستوى باستخدام دالة تسجيل مختلفة:

مستوى التسجيل	دالة التسجيل	وصفها
المستوى DEBUG	الدالة `logging.debug()`	المستوى الأدنى، ويُستخدَم للتفاصيل الصغيرة، حيث تهتم بهذه الرسائل عند تشخيص المشاكل فقط.
المستوى INFO	الدالة `logging.info()`	يُستخدَم لتسجيل معلومات عن الأحداث العامة في برنامجك أو للتأكد من أن الأمور تسير جيدًا في البرنامج.
المستوى WARNING	الدالة `logging.warning()`	يُستخدم للإشارة إلى مشكلة محتملة لا تمنع البرنامج من العمل لكنها قد تفعل مستقبلًا.

المستوى ERROR	الدالة `logging.error()`	يُستخدم لتسجيل خطأٍ تسبّب في فشل البرنامج بعمل شيءٍ ما.
المستوى CRITICAL	الدالة `logging.critical()`	المستوى الأعلى، ويُستخدَم للإشارة إلى خطأ كبير تسبّب أو أنه على وشك التسبّب في توقف البرنامج عن العمل بالكامل.

الجدول 11: مستويات التسجيل المختلفة

تُمرَّر رسالة التسجيل بوصفها سلسلة نصية إلى هذه الدوال. وتُعَد مستويات التسجيل مجرد اقتراحات، فالأمر متروك لك من أجل تحديد الفئة التي تندرج ضمنها رسالة السجل الخاصة بك. لنـدخِل الآن ما يلي في الصدفة التفاعلية:

```
>>> import logging
>>> logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -
%(levelname)s -  %(message)s')
>>> logging.debug('Some debugging details.')
2024-05-18 19:04:26,901 - DEBUG - Some debugging details.
>>> logging.info('The logging module is working.')
2024-05-18 19:04:35,569 - INFO - The logging module is working.
>>> logging.warning('An error message is about to be logged.')
2024-05-18 19:04:56,843- WARNING - An error message is about to be
logged.
>>> logging.error('An error has occurred.')
2024-05-18 19:05:07,737 - ERROR - An error has occurred.
>>> logging.critical('The program is unable to recover!')
2024-05-18 19:05:45,794 - CRITICAL - The program is unable to recover!
```

تتمثّل فائدة مستويات التسجيل في أنه يمكنك تغيير أولوية رسالة التسجيل التي تريد رؤيتها، حيث سيؤدي تمرير المستوى `logging.DEBUG` إلى وسيط الكلمة المفتاحية Keyword Argument الذي هو `level` الخاص بالدالة `basicConfig()` إلى إظهار الرسائل من جميع مستويات التسجيل، حيث يُعَد المستوى DEBUG هو المستوى الأدنى. قد تكون مهتمًا بالأخطاء فقط بعد تطوير برنامجك، وبالتالي يمكنك ضبط الوسيط `level` الخاص بالدالة `basicConfig()` على المستوى `logging.ERROR` في هذه الحالة، إذ سيعرض هذا المستوى رسائل المستوى ERROR ورسائل المستوى CRITICAL فقط ويتخطى رسائل المستويات DEBUG و INFO و WARNING.

11.4.4 تعطيل التسجيل

لا بد أنك تفضّل عدم إظهار جميع رسائل السجل التي تؤدي إلى جعل الشاشة مزدحمة بعد تنقيح أخطاء برنامجك، لذا توجد الدالة `logging.disable()` التي تعمل على تعطيل رسائل السجل حتى لا تضطر إلى الدخول إلى برنامجك وإزالة جميع استدعاءات التسجيل يدويًا.

يمكنك تمرير هذه الدالة إلى مستوى التسجيل فقط، وستمنع هذه الدالة جميع رسائل السجل عند هذا المستوى أو المستويات الأقل، لذا إذا أردتَ تعطيل التسجيل بالكامل، فما عليك سوى إضافة الاستدعاء `logging.disable(logging.CRITICAL)` إلى برنامجك.

أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import logging
>>> logging.basicConfig(level=logging.INFO, format=' %(asctime)s -
%(levelname)s -  %(message)s')
>>> logging.critical('Critical error! Critical error!')
2024-05-22 11:10:48,054 - CRITICAL - Critical error! Critical error!
>>> logging.disable(logging.CRITICAL)
>>> logging.critical('Critical error! Critical error!')
>>> logging.error('Error! Error!')
```

ستعطّل الدالة `logging.disable()` جميع الرسائل بعدها، لذا يُحتمَل أنك تريد إضافتها بالقرب من سطر الاستيراد `import logging` في برنامجك، وبالتالي يمكنك بسهولة العثور عليه لتعليق هذا الاستدعاء أو إلغاء تعليقه لتفعيل رسائل التسجيل أو تعطيلها حسب الحاجة.

11.4.5 التسجيل في ملف

يمكنك كتابة رسائل السجل في ملف نصي بدلًا من عرضها على الشاشة، حيث تأخذ الدالة `logging.basicConfig()` وسيط الكلمة المفتاحية `filename` كما يلي:

```
import logging
logging.basicConfig(filename='myProgramLog.txt', level=logging.DEBUG,
format='
%(asctime)s -  %(levelname)s -  %(message)s')
```

ستُحفَظ رسائل السجل في الملف `myProgramLog.txt`.يمكن أن تؤدي رسائل التسجيل إلى جعل شاشتك مزدحمة بالعناصر وتجعل من الصعب قراءة خرج البرنامج بالرغم من فوائدها التي تحدّثنا عنها سابقًا، لذا كتبنا رسائل التسجيل في ملف لإبقاء شاشتك واضحة ولتخزين الرسائل حتى تتمكّن من قراءتها بعد تشغيل البرنامج، حيث يمكنك فتح هذا الملف النصي في أي محرّر نصوص مثل المفكرة Notepad أو TextEdit.

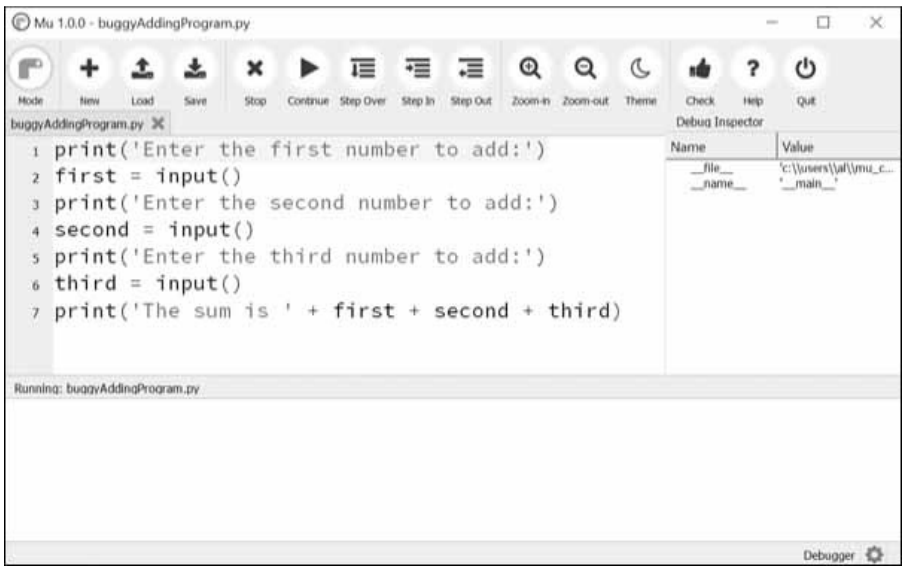
11.5 منقح أخطاء المحرّر Mu

يُعَد منقّح الأخطاء من ميزات المحرّر Mu و IDLE وبرامج تحرير النصوص الأخرى التي تسمح لك بتنفيذ برنامجك من خلال تنفيذ سطر واحد في كل مرة، حيث يشغّل منقّح الأخطاء سطرًا واحد من الشيفرة البرمجية ثم ينتظر حتى تخبره بالمتابعة.

يمكنك أن تأخذ الوقت الذي تريده لفحص القيم الموجودة في المتغيرات في أي مرحلة معينة من عمر البرنامج من خلال تشغيل برنامجك مع منقّح الأخطاء باستخدام هذه الطريقة، إذ يُعَد منقّح الأخطاء أداةً مفيدة لتعقّب الأخطاء.

يمكنك تشغيل برنامج مع منقّح أخطاء المحرّر Mu من خلال النقر على زر تنقيح الأخطاء "Debug" في الصف العلوي من الأزرار بجانب زر التشغيل "Run".

ستفتح نافذة فحص تنقيح الأخطاء "Debug Inspector" على طول الجانب الأيمن من النافذة بالإضافة إلى نافذة الخرج المعتاد في الأسفل، حيث تسرد نافذة فحص تنقيح الأخطاء قيم المتغيرات الحالية في برنامجك. يوقِف منقّح الأخطاء في الشكل التالي تنفيذ البرنامج قبل تشغيل السطر الأول من الشيفرة البرمجية، حيث يمكنك رؤية هذا السطر مميزًا في محرّر الملفات:



الشكل 45: تشغيل المحرّر Mu لبرنامجٍ ما مع منقّح الأخطاء

يضيف وضع تنقيح الأخطاء أيضًا أزرارًا جديدة إلى أعلى المحرّر وهي: زر المتابعة "Continue" وزر "Step Over" وزر "Step In" وزر "Step Out"، ويوجد زر التوقف "Stop" المعتاد أيضًا.

11.5.1 زر المتابعة Continue

يؤدي النقر على زر المتابعة "Continue" إلى تنفيذ البرنامج بطريقة طبيعية حتى ينتهي البرنامج أو يصل إلى نقطة توقف Breakpoint (سنوضح نقاط التوقف لاحقًا في هذا الفصل). إذا انتهيتَ من تنقيح الأخطاء وأردتَ أن يتابع البرنامج عمله بطريقة طبيعية، فانقر على زر المتابعة "Continue".

11.5.2 زر Step In

يؤدي النقر على زر "Step In" إلى أن ينفّذ منقّح الأخطاء السطر التالي من الشيفرة البرمجية ثم يوقفه مرة أخرى. إذا كان السطر التالي من الشيفرة البرمجية هو استدعاء دالة، فسيدخل منقّح الأخطاء إلى تلك الدالة وينتقل إلى السطر الأول من الشيفرة البرمجية فيها.

11.5.3 زر Step Over

يؤدي النقر على زر "Step Over" إلى تنفيذ السطر التالي من الشيفرة البرمجية مثل زر "Step In"، ولكن إذا كان السطر التالي من الشيفرة البرمجية هو استدعاء دالة، فسيتجاوز زر "Step Over" الشيفرة البرمجية الموجودة في هذه الدالة، حيث ستُنفَّذ الشيفرة البرمجية الخاصة بالدالة بالسرعة القصوى، وسيتوقّف منقّح الأخطاء بعد العودة من استدعاء هذه الدالة.

إذا استدعى السطر التالي من الشيفرة البرمجية الدالة `spam()` مثلًا، ولكنك لا تهتم بالشيفرة البرمجية الموجودة ضمن هذه الدالة، فيمكنك النقر على زر "Step Over" لتنفيذ الشيفرة البرمجية الموجودة في الدالة بالسرعة العادية ثم التوقف عندما تعود الدالة، لذلك يُعَد استخدام زر "Step Over" أكثر شيوعًا من استخدام زر "Step In".

11.5.4 زر Step Out

يؤدي النقر على زر "Step Out" إلى أن ينفّذ منقّح الأخطاء سطورًا من الشيفرة البرمجية بالسرعة القصوى حتى العودة من الدالة الحالية. إذا دخلتَ في استدعاء دالة باستخدام زر "Step In" وتريد الاستمرار في تنفيذ التعليمات حتى الخروج منها، فانقر على زر "Step Out" للخروج من استدعاء الدالة الحالي.

11.5.5 زر التوقف Stop

إذا أردتَ إيقاف تنقيح الأخطاء وعدم إزعاجك بمتابعة تنفيذ بقية البرنامج، فانقر فوق زر التوقّف "Stop"، حيث سيؤدي الزر "Stop" إلى إنهاء البرنامج مباشرةً.

11.5.6 تنقيح أخطاء برنامج لجمع الأعداد

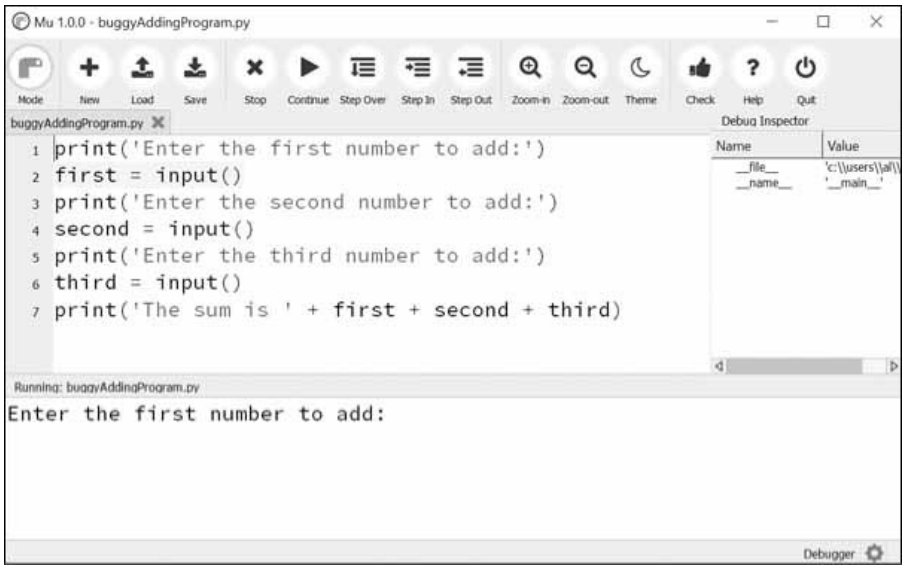
افتح تبويبًا جديدًا في محرّرك لإنشاء ملف جديد وأدخِل الشيفرة البرمجية التالية:

```
print('Enter the first number to add:')
first = input()
print('Enter the second number to add:')
second = input()
print('Enter the third number to add:')
third = input()
print('The sum is ' + first + second + third)
```

احفظ البرنامج بالاسم `buggyAddingProgram.py` وشغّله أولًا دون تفعيل منقّح الأخطاء، وسيكون خرج البرنامج كما يلي:

```
Enter the first number to add:
5
Enter the second number to add:
3
Enter the third number to add:
42
The sum is 5342
```

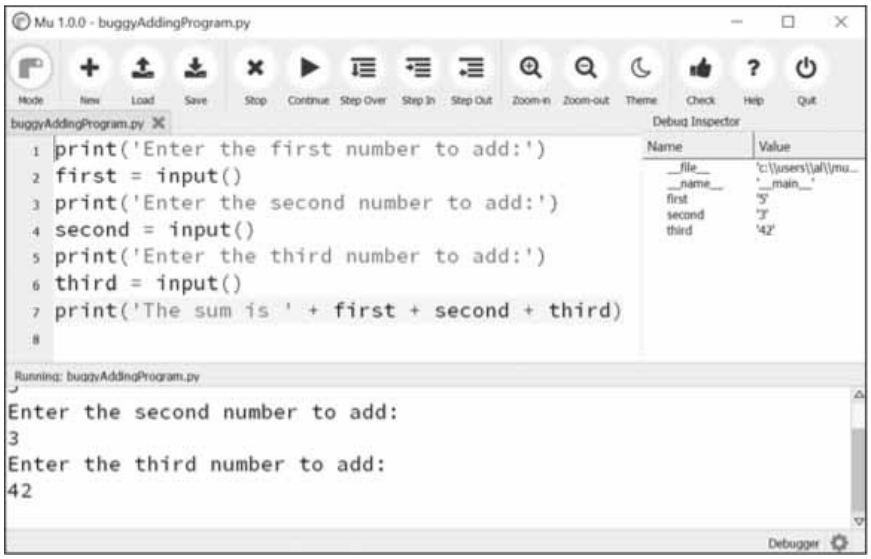
لم يتعطل البرنامج، ولكن من الواضح أن ناتج الجمع خاطئ. شغّل البرنامج مرة أخرى ولكن مع منقّح الأخطاء هذه المرة، حيث إذا نقرتَ على زر تنقيح الأخطاء "Debug"، فسيتوقف البرنامج مؤقتًا عند السطر 1، وهو سطر الشيفرة البرمجية الذي نوشك على تنفيذه، إذ يجب أن يبدو المحرّر Mu كما في الشكل السابق. انقر على زر "Step Over" مرة واحدة لتنفيذ الاستدعاء الأول `print()`، إذ يجب أن تستخدم زر "Step Over" بدلًا من زر "Step In" هنا، لأنك لا تريد الدخول إلى الشيفرة البرمجية الخاصة بالدالة `print()`، بالرغم من أن المحرّر Mu يجب أن يمنع منقّح الأخطاء من الدخول إلى دوال بايثون المُدمَجة. ينتقل منقّح الأخطاء إلى السطر 2، ويميّز السطر 2 في محرر الملفات كما هو موضح في الشكل التالي، مما يوضّح لك مكان تنفيذ البرنامج حاليًا:



الشكل 46: نافذة المحرّر Mu بعد النقر على زر "Step Over"

انقر على زر "Step Over" مرة أخرى لتنفيذ استدعاء الدالة `input()`، حيث سيختفي التمييز عن الشيفرة البرمجية أثناء انتظار المحرّر Mu أن تكتب شيئًا ما لاستدعاء الدالة `input()` في نافذة الخرج. أدخِـل القيمـة 5 واضغط على مفتاح `ENTER`، ثم سيعود التمييز إلى الشيفرة البرمجية.

استمر في النقر على زر "Step Over"، وأدخِل القيمتين 3 و 42 بوصفهما العددين التاليين. يجب أن تبـدو نافذة المحرّر Mu كما في الشكل التالي عنـدما يصـل منقّح الأخطـاء إلى السـطر 7 الـذي يمثّـل اسـتدعاء الدالـة `print()` النهائي في البرنامج:



الشكل 47: نافذة فحص تنقيح الأخطاء Debug Inspector

توضّح نافذة فحص تنقيح الأخطاء Inspector Debug الموجودة على الجانب الأيمن أن المتغيرات مضبوطة بوصفها سلاسلًا نصية وليست أعدادًا صحيحة، مما تسبّب في حدوث الخطأ

يجب أن ترى في نافذة فحص تنقيح الأخطاء Inspector Debug أن المتغيرات `first` و `second` و`third` مضبوطة بوصفها سلاسلًا نصية `'5'` و `'3'` و `'42'` بدلًا من الأعداد الصحيحة 5 و 3 و 42. تضم لغة بايثون هذه السلاسل النصية مع بعضها بعضًا عند تنفيذ السطر الأخير بدلًا من جمع هذه الأعداد، مما يتسبّب في حدوث الخطأ.

يُعَد التنقل عبر البرنامج باستخدام منقّح الأخطاء بطيئًا، لذا نريد في أغلب الأحيان أن يعمل البرنامج بصورة طبيعية حتى يصل إلى سطر معين من الشيفرة البرمجية، حيث يمكنك ضبط منقّح الأخطاء لتطبيق ذلك باستخدام نقاط التوقف.

11.5.7 نقاط التوقف Breakpoints

يمكن ضبط نقطة توقف على سطر معين من الشيفرة البرمجية وإجبار منقح الأخطاء على التوقف مؤقتًا عندما يصل تنفيذ البرنامج إلى هذا السطر.

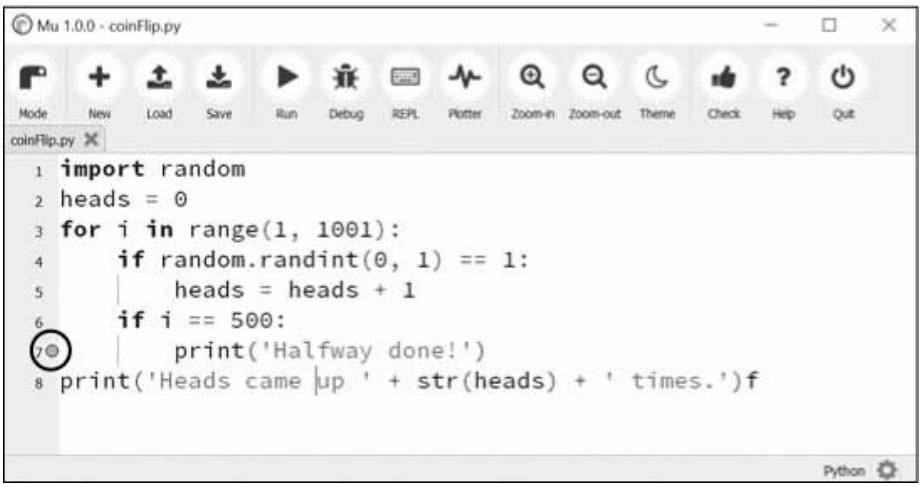
افتح تبويبًا جديدًا في محرّر الملفات وأدخِل البرنامج التالي الذي يحاكي رمي عملة معدنية 1000 مرة، واحفظ البرنامج بالاسم `coinFlip.py`:

```
import random
heads = 0
for i in range(1, 1001):
 ❶ if random.randint(0, 1) == 1:
    heads = heads + 1
  if i == 500:
   ❷ print('Halfway done!')
print('Heads came up ' + str(heads) + ' times.')
```

سيعيد الاستدعاء `random.randint(0, 1)` ❶ القيمة 0 في نصف الوقت والقيمة 1 في النصف الآخر من الوقت، حيث يمكن استخدام هذا الاستدعاء لمحاكاة رمي قطعة نقود وفق الاحتمال 50/50 وتمثل القيمة 1 الصورة من العملة المعدنية. يكون خرج هذا البرنامج كما يلي عند تشغيله بدون منقّح الأخطاء:

```
Halfway done!
Heads came up 490 times.
```

إذا شغّلتَ هذا البرنامج مع منقّح الأخطاء، فيجب أن تنقر على زر "Step Over" آلاف المرات قبل إنهاء البرنامج. إذا كنت مهتمًا بالقيمة `heads` التي تمثّل صورة العملة المعدنية عند منتصف تنفيذ البرنامج أو عند اكتمال 500 مرة من 1000 مرة لرمي قطعة نقود، فيمكنك ضبط نقطة توقف على السطر `print('Halfway done!')` ❷، حيث يمكنك ضبط نقطة توقف من خلال النقر على رقم السطر في محرّر الملفات بحيث تظهر نقطة حمراء كما في الشكل التالي:



الشكل 48: يؤدي ضبط نقطة التوقف إلى ظهور نقطة حمراء (محاطة بدائرة) بجانب رقم السطر

لا نريد ضبط نقطة توقف عند سطر التعليمة `if` لأنها تُنفَّذ في كل تكرار للحلقة، حيث إذا ضبطنا نقطة التوقف على الشيفرة البرمجية الموجودة ضمن التعليمة `if`، فسيتوقّف منقّح الأخطاء فقط عندما يدخل التنفيذ إلى هذه التعليمة.

سيكون للسطر الذي يحتوي على نقطة التوقف نقطة حمراء بجانبه. إذا شغّلنا البرنامج مع منقّح الأخطاء، فسيبدأ في حالة التوقّف المؤقت عند السطر الأول كالمعتاد، ولكن إذا نقرت على زر المتابعة "Continue"، فسيُشغَّل البرنامج بالسرعة القصوى حتى يصل إلى السطر الذي ضبطنا نقطة التوقف عنده. يمكنك بعد ذلك النقر على أزرار "Continue" أو "Step Over" أو "Step In" أو "Step Out" للمتابعة كالمعتاد.

إذا أردتَ إزالة نقطة توقف، فانقر على رقم السطر مرة أخرى، وستختفي النقطة الحمراء، ولن يتوقف منقّح الأخطاء عند هذا السطر لاحقًا.

11.6 أسئلة للتدريب

1. اكتب تعليمة `assert` التي تؤدي إلى ظهور الخطأ `AssertionError` إذا كان المتغير `spam` عددًا صحيحًا أصغر من 10.

2. اكتب تعليمة `assert` التي تؤدي لظهور الخطأ `AssertionError` في حال ما إذا احتوى المتغيران `eggs` و`beef` على سلاسل نصية متماثلة حتى ولو كانت حالات حروفها مختلفة (مثل السلسلتين النصيتين `'hello'` و`'hello'` المتماثلتين، والسلسلتين النصيتين `'goodbye'` و`'GOODbye'` المتماثلتين أيضًا).
3. اكتب تعليمة `assert` التي تؤدي دائمًا إلى ظهور الخطأ `AssertionError`.
4. ما هما السطران اللذان يجب أن يتوفرا في برنامجك حتى يتمكن من استدعاء الدالة `logging.debug()`؟
5. ما هما السطران اللذان يجب أن يتوفرا في برنامجك حتى تتمكّن الدالة `logging.debug()` من إرسال رسالة تسجيل إلى ملفٍ اسمه `programLog.txt`؟
6. ما هي مستويات التسجيل الخمسة؟
7. ما هو سطر الشيفرة البرمجية الذي يمكنك إضافته لتعطيل جميع رسائل التسجيل في برنامجك؟
8. لماذا يُعَد استخدام رسائل التسجيل أفضل من استخدام الدالة `print()` لعرض الرسالة نفسها؟
9. ما هي الاختلافات بين أزرار "Step Over" و "Step In" و "Step Out" في منقّح الأخطاء؟
10. متى سيتوقف منقّح الأخطاء بعد النقر على زر المتابعة "Continue"؟
11. ما هي نقطة التوقف؟
12. كيف يمكنك ضبط نقطة توقف على سطر من شيفرة برمجية في المحرّر Mu؟

11.7 مشاريع للتدريب

حاول كتابة برنامج يطبّق ما يلي لكسب خبرة عملية أكبر.

11.7.1 تنقيح أخطاء برنامج لرمي عملة معدنية

يهدف هذا البرنامج إلى إنشاء لعبة تخمين بسيطة لرمي عملة معدنية مع حصول اللاعب على تخمينين، ولكنه يحتوي على العديد من الأخطاء بالرغم من سهولة هذه اللعبة. شغّل البرنامج عدة مرات للعثور على الأخطاء التي تمنع البرنامج من العمل بصورة صحيحة.

```
import random
guess = ''
while guess not in ('heads', 'tails'):
  print('Guess the coin toss! Enter heads or tails:')
```

```
  guess = input()

toss = random.randint(0, 1) # تمثّل القيمة 0 الكتابة، وتمثل القيمة 1 الصورة في العملة المعدنية

if toss == guess:
  print('You got it!')
else:
  print('Nope! Guess again!')
  guesss = input()
  if toss == guess:
    print('You got it!')
  else:
    print('Nope. You are really bad at this game.')
```

11.8 الخلاصة

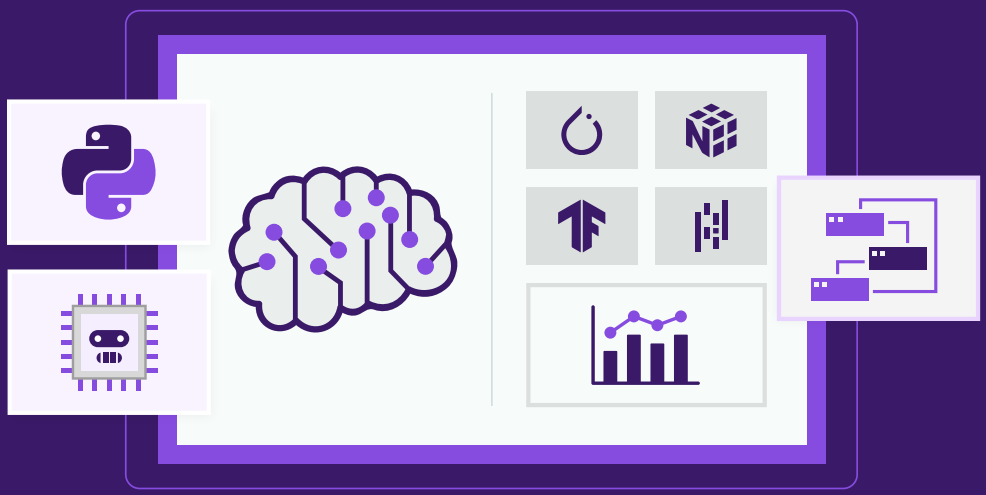
تُعَـد التأكيـدات والاسـتثناءات والتسـجيل ومنقح الأخطـاء أدوات قيّمـة للعثـور على الأخطـاء ومنعهـا في برنامجك، حيث تُعَد التأكيدات باستخدام التعليمة `assert` في لغة بايثون طريقة جيـدة لتنفيـذ فحص السـلامة الذي يمنحك تحذيرًا مبكرًا عندما لا يكون الشرط الضروري صحيحًا، وتكـون التأكيـدات مخصَّصـة فقـط للأخطـاء التي يجب ألّا يحاول البرنامج التعافي منها ويجب أن يفشل بسرعة، وإلّا فيجب أن ترفع استثناء.

يمكن اكتشاف الاستثناء ومعالجته باستخدام تعليمات `try` و `except`. تُعَد الوحدة `logging` طريقة جيدة لتفحّص شيفرتك البرمجيـة أثنـاء تشـغيلها، وهي أكـثر ملاءمـة للاسـتخدام من الدالـة `print()` لاحتوائهـا على مستويات تسجيل متعددة ولقدرتها على التسجيل في ملف نصي.

يتيح لك منقّح الأخطاء التنقل عبر برنامجك سطرًا تلـو الآخـر، ويمكنـك تشـغيل برنامجـك بالسـرعة العاديـة وجعل منقّح الأخطاء يوقف التنفيذ مؤقتًا عندما يصل إلى سطرٍ ضبطنا عنده نقطة توقـف، ويمكنـك رؤيـة حالـة قيمة أيّ متغير في أي وقت من عمر البرنامج باستخدام منقّح الأخطاء.

تساعدك هذه الأدوات والتقنيات لتنقيح الأخطاء على كتابة الـبرامج الناجحـة، حيث يُعَـد إدخـال أخطـاء في شيفرتك البرمجية عن طريق الخطأ حقيقة يجب التسليم بها بغض النظر عن عدد سنوات خبرتك في البرمجة.

دورة الذكاء الاصطناعي



تعلم الذكاء الاصطناعي وتعلم الآلة والتعلم العميق
وتحليل البيانات، وأضفها إلى تطبيقاتك

التحق بالدورة الآن

12. استخراج بيانات الويب عبر لغة بايثون

كانت لحظة مرعبة حينما جلست على حاسوبي بعد أن «عضّ القرش» كبل الإنترنت وانقطع، وأدركت حينها كم أقضي وقتًا على الإنترنت حينما أستعمل حاسوبي؛ فلدي عادة أن أتحقق من بريدي يدويًا (مع أن التنبيهات تصلني أولًا بأول!) وأفتح تويتر (إكس، سمةٌ ما شئت) وأنظر ما آخر المستجدات.

كثيرٌ من عملنا على الحاسوب يتطلب وصولًا إلى الانترنت، ومصطلح «استخراج البيانات من الويب Web scraping» يستعمل مع البرامج التي تنزل المحتوى من الإنترنت وتعالجه. فمثلًا لدى غوغل عدد من البوتات لتنزيل محتوى صفحات الويب وفهرستها وأرشفتها لتستعملها في محرك البحث.

سنتعلم في هذا الفصل عن عددٍ من الوحدات في بايثون التي تسهل علينا استخراج البيانات من الويب:

- `webbrowser`: حزمة تأتي مع بايثون وتفتح متصفحًا على صفحة ويب معينة.
- `requests`: تنزل الملفات وصفحات الويب من الإنترنت.
- `bs4`: تفسّر شيفرات HTML التي تكتب فيها صفحات الويب.
- `selenium`: تشغل وتتحمل في متصفح ويب، والوحدة `selenium` قادرة على ملء الاستمارات ومحاكاة ضغطات الفأرة في المتصفح.

12.1 مشروع: برنامج maplt.py مع وحدة webbrowser

الدالة `open()` في الوحدة `webbrowser` تفتح صفحة ويب معينة في نافذة متصفح جديدة.

جرب ما يلي في الصدفة التفاعلية:

```
>>> import webbrowser
>>> webbrowser.open('https://academy.hsoub.com/')
```

ستجد أكاديمية حسوب مفتوحةً في لسانٍ جديد في المتصفح. هذا كل ما تستطيع الوحـدة webbrowser فعله، لكن مع ذلك يمكننا أن نجري بعض الأمور اللطيفة مع الدالة open()، فمثلًا قد تكـون مهمـة فتح خرائـط غوغل والبحث عن عنوان معين أمرًا مملًا، ونستطيع التخلص من بضـع خطـوات لـو كتبنـا سـكربتًا يفتح خرائـط غوغل ويوجهها إلى عنوان الشارع المنسوخ في الحافظة لـديك، وبالتـالي سـيكون عليـك أن تنسـخ العنـوان إلى الحافظة وتشغل السكربت، وستفتح الخريطة لديك.

هذا ما يفعله البرنـامج: يحصـل على عنـوان الشـارع من وسـائط سـطر الأوامـر أو من الحافظـة يفتح نافـذة متصفح ويوجهها إلى صفحة خرائط غوغل المرتبطة بعنوان الشارع

هذا يعني أن على الشيفرة البرمجية أن تفعل ما يلي:

- تقرأ وسائط سطر الأوامر
- تقرأ محتويات الحافظة
- تستدعي الدالة webbrowser.open() لتفتح صفحة الويب

احفظ ملفًا جديدًا باسم mapIt.py، ولنبدأ برمجته.

12.1.1 الخطوة 1: معرفة الرابط الصحيح

اعتمادًا على التعليمات الموجودة في الملحق 2، اضبط برنامج mapIt.py ليعمل من سطر الأوامر كما في المثال الآتي:

```
C:\> mapit 870 Valencia St, San Francisco, CA 94110
```

سيستخدم البرنامج وسائط سـطر الأوامـر بـدلًا من الحافظـة، وإذا لم نمـرر إليـه أيّـة وسـائط فحينهـا سـيقرأ محتويات الحافظة.

علينا بدايةً أن نحصّل ما هو عنوان URL الذي يجب فتحه للعثور على شارع معين. إذا فتحت خرائط غوغـل في المتصفح وبحثت عن عنوان فسيكون الرابط في الشريط العلوي يشبه:

```
https://www.google.com/maps/place/870+Valencia+St/@37.7590311,-
122.4215096,17z/data=!3m1!4b1!4m2!3m1!
1s0x808f7e3dadc07a37:0xc86b0b2bb93b73d8
```

نعم العنوان في الرابط، لكن هنالك نص كثير إضافي غيره.

تضيف المواقع عادةً بيانات إضافية لتتبع الزوار أو تخصيص المواقع. لكن إن حاولت الذهاب إلى الرابط https://www.google.com/maps/place/870+Valencia+St+San+Francisco+CA/ مثلًا، فسوف ترى العنوان مفتوحًا أمامك. وبهذا كل ما سنحتاج إليه هو فقط فتح صفحة ويب ذات العنوان https://www.google.com/maps/place/your_address_string فـ your_address_string هو العنوان الذي تريد عرضه في الخريطة.

12.1.2 الخطوة 2: التعامل مع وسائط سطر الأوامر

يجب أن تبدو الشيفرة لديك كما يلي:

```
#! python3
# mapIt.py - تشغيل خريطة في المتصفح باستخدام عنوان

# من سطر الأوامر أو الحافظة

import webbrowser, sys
if len(sys.argv) > 1:
  # الحصول على العنوان من سطر الأوامر

  address = ' '.join(sys.argv[1:])

# TODO: الحصول على العنوان من الحافظة.
```

من بعد سطر shebang `#!` سنستورد الوحدة `webbrowser` لتشغيل المتصفح والوحدة `sys` لمحاولة قراءة وسائط سطر الأوامر.

يخزن المتغير `sys.argv` اسم الملف ووسائط سطر الأوامر، وإذا احتوت هذه القائمة على أكثر من عنصر واحد (الذي هو اسم الملف) فيجب أن تكون قيمة استدعاء `len(sys.argv)` أكبر من 1، وهذا يعني أن هنالك وسائط في سطر الأوامر.

عادةً ما يُفصَل بين وسائط سطر الأوامر بفراغات، لكن في هذه الحالة نريد أن نفسر جميع الوسائط على أنها سلسلة نصية واحدة، ولما كانت قيمة `sys.argv` هي قائمة تحتوي على سلاسل نصية، فيمكننا استخدام التابع `join()` معها، مما يعيد سلسلة نصية واحدة؛ لكن انتبه أننا لا نريد اسم الملف ضمن تلك السلسلة النصية، فعلينا استخدام `sys.arv[1:]` لإزالة أول عنصر في القائمة، ثم سنتخزن تلك القيمة في المتغير `address`.

يمكنك تشغيل البرنامج بكتابة ما يلي في سطر الأوامر:

```
mapit 870 Valencia St, San Francisco, CA 94110
```

وستكون قيمة المتغير `sys.argv` هي القائمة:

```
['mapIt.py', '870', 'Valencia', 'St, ', 'San', 'Francisco, ', 'CA',
'94110']
```

وبالتالي تكون قيمة المتغير `address` هي السلسلة النصية التالية:

```
'870 Valencia St, San Francisco, CA 94110'
```

12.1.3 الخطوة 3: التعامل مع محتويات الحافظة وتشغيل المتصفح

تأكد أن الشيفرة الخاصة بك تشبه الشيفرة الآتية:

```
#! python3
# mapIt.py - تشغيل خريطة في المتصفح باستخدام عنوان

# من سطر الأوامر أو الحافظة.
import webbrowser, sys, pyperclip
if len(sys.argv) > 1:
  # الحصول على العنوان من سطر الأوامر.
  address = ' '.join(sys.argv[1:])
else:
  # الحصول على العنوان من الحافظة.
  address = pyperclip.paste()

webbrowser.open('https://www.google.com/maps/place/' + address)
```

إذا لم تكن هنالك وسائط ممررة عبر سطر الأوامر، فسيفترض البرنامج أن العنوان منسوخ إلى الحافظة، ويمكننا الحصول على محتوى الحافظة باستخدام `pyperclip.paste()` وتخزينها في المتغير `address`. آخر خطوة هي تشغيل المتصفح مع توفير رابط URL صحيح لخرائط غوغل عبر `webbrowser.open()`.

ستوفر عليك بعض البرامج التي ستطورها ساعات من العمل، لكن بعض قد يكون بسيطًا ويوفر عليك بضع ثوانٍ في كل مرة تجري فيها مهمة تكرارية مثل فتح عنوان ما على الخريطة.

يقارن الجدول الآتي بين الخطوات اللازمة لعرض الخريطة:

فتح الخريطة يدويًا	استخدام سكربت mapIt.py
تحديد العنوان / نسخ العنوان/ فتح متصفح الويب/ الذهاب إلى خرائط غوغل / الضغط على حقل الإدخال/ لصق العنوان الضغط على enter	تحديد العنوان/ نسخ العنوان / تشغيل `mapIt.py`

الجدول 12: الفرق بين الفتح اليدوي والمؤتمت للخريطة

12.1.4 أفكار لبرامج مشابهة

تساعدك الوحدة `webbrowser` إذا كان لديك عنوان URL لصفحة معينة تريد اختصار عملية فتح المتصفح والتوجه إليها، يمكنك أن تستفيد منها لإنشاء:

- برنامج يفتح جميع الروابط المذكورة في مستند نصي في ألسنة جديدة
- برنامج يفتح المتصفح على صفحة الطقس لمدينتك
- برنامج يفتح مواقع التواصل الاجتماعي التي تزورها عادة

12.2 تنزيل الملفات من الويب باستخدام الوحدة requests

تسمح لك الوحدة `requests` بتنزيل الملفات من الويب دون أن تفكر في مشاكل الشبكة أو الاتصال أو ضغط البيانات. لا تأتي الوحدة `requests` مضمنةً في بايثون، وإنما عليك تثبيتها أولًا من سطر الأوامر بتشغيل الأمر `pip install --user requests` (راجع الملحق 1 لمزيد من المعلومات حول تثبيت الوحدات الخارجية).

أتت الوحدة `requests` لتقدم حلًا بديلًا لوحدة `urllib2` في بايثون لأنها معقدة زيادة عن اللزوم، وأنصحك أن تزيل الوحدة `urllib2` من ذهنك تمامًا، لأنها وحدة صعبة دون داعٍ، وعليك استخدام الوحدة `requests` دومًا.

لنجرب الآن أن الحزمة `requests` مثبتة صحيحًا بإدخال ما يلي في الصدفة التفاعلية:

```
import requests>>> import requests
```

إذا لم تظهر أي رسالة خطأ فهذا يعني أن الحزمة `requests` مثبتة عندك.

12.2.1 تنزيل صفحة ويب باستخدام الدالة requests.get()

الدالة `requests.get()` تقبل سلسلةً نصيةً فيها رابط URL لتنزيلها. إذا استعملت الدالة `type()` على القيمة المعادة من الدالة `requests.get()` فسترى أن الناتج هو كائن `Response`، الذي يحتوي على الرد الذي تلقاه برنامجك بعد إتمام الطلبية إلى خادم الويب. سنستكشف سويةً الكائن `Response` بالتفصيل لاحقًا، لكن لنكتب الآن الأسطر الآتية في الطرفية التفاعلية على حاسوب متصل بالإنترنت:

```
>>> import requests

❶ >>> res =
requests.get('https://automatetheboringstuff.com/files/rj.txt')
>>> type(res)
<class 'requests.models.Response'>
❷ >>> res.status_code == requests.codes.ok
True
>>> len(res.text)
178981
>>> print(res.text[:250])
The Project Gutenberg EBook of Romeo and Juliet, by William
Shakespeare

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Proje
```

الرابط الذي طلبناه هو مستند نصي لمسرحية روميو وجولييت على موقع الكتاب الأصلي ❶، يمكنك أن ترى نجاح الطلبية إلى صفحة الويب بالنظر إلى السمة `status_code` من الكائن `Response`، إذا كانت القيمة الخاصة بها تساوي `requests.codes.ok` فهذا يعني أن كل شيء على ما يرام ❷.

بالمناسبة، رمز الاستجابة الذي يدل على أن الأمور على ما يرام OK في HTTP هو 200، ومن المرجح أنك تعرف الحالة 404 التي تشير إلى رابط غير موجود. يمكنك العثور على قائمة برموز الاستجابة في HTTP ومعانيها في الصفحة صفحة ويكيبيديا حول قائمة رموز الاستجابة في HTTP.

إذا نجحت الطلبية، فستخزن صفحة الويب التي نزلناها كسلسلة نصية في المتغير `text` في كائن `Response`. يحتوي هذا المتغير على سلسلة نصية طويلة فيها المسرحية كاملةً. وإذا استدعيت `len(res.text)` فسوف ترى أنها أطول من 178,000 محرف. لهذا استدعينا في النهاية `print(res.text[:250])` لعرض أول 250 محرف.

إذا فشل الطلب وظهرت رسالة خطأ مثل "Failed to establish a new connection" أو "Max retries exceeded" فتأكد من اتصالك بالإنترنت. لا نستطيع نقاش جميع أسباب عدم القدرة على الاتصال بالخوادم لتعقيد الموضوع، لكن أنصحك بالبحث في الويب عن المشكلة التي تواجهك لترى حلها.

12.2.2 التأكد من عدم وجود مشاكل

كما رأينا سويةً، يملك الكائن `Response` السمة `status_code` التي تأكدنا أنها تساوي `requests.codes.ok` (وهو متغير فيه القيمة الرقمية 200) للتحقق من نجاح عملية التنزيل.

هنالك طريقة أخرى سهلة للتحقق من نجاح التنفيذ هو استدعاء التابع `raise_for_status()` على الكائن `Response`، التي ستؤدي إلى إطلاق استثناء إذا حدث خطأ حين تنزيل الملف، ولن تفعل شيئًا إن سارت الأمور على ما يرام. جرب ما يلي في الصدفة التفاعلية:

```
>>> res =
requests.get('https://inventwithpython.com/page_that_does_not_exist')
>>> res.raise_for_status()
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>


 File "C:\Users\Al\AppData\Local\Programs\Python\Python37\lib\site-
packages\requests\models
.py", line 940, in raise_for_status
  raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 404 Client Error: Not Found for url:
https://inventwithpython
.com/page_that_does_not_exist.html
```

يضمن لنا التابع `raise_for_status()` أن البرنامج سيتوقف إذا حدثت مشكلة في التنزيل، وهذا مناسب جدًا إذا أردنا إيقاف البرنامج حين حصول مشكلة في التنزيل. أما لو كنا نريد استمرار البرنامج حتى لو فشل تنزيل الملف فيمكننا حينئذٍ أن نحيط التابع `raise_for_status()` بالتعبيرات `try` و `except` لمعالجة هذا الاستثناء:

```
import requests
res =
requests.get('https://inventwithpython.com/page_that_does_not_exist')
try:
  res.raise_for_status()
```

```
except Exception as exc:
  print('There was a problem: %s' % (exc))
```

سيؤدي التابع raise_for_status()raise_for_status() في الشيفرة السابقة لطباعة ما يلي:

```
There was a problem: 404 Client Error: Not Found for url: https://
inventwithpython.com/page_that_does_not_exist.html
```

احرص دومًا على استدعاء التابع raise_for_status() بعد استدعاء requests.get() لتضـمن أن برنامجك قد نزَّل الملف دون مشاكل قبل إكمال التنفيذ.

12.2.3 حفظ الملفات المنزلة إلى نظام الملفات

يمكنك الآن حفظ صفحة الـويب إلى نظـام الملفـات لـديك باسـتخدام الدالـة open() والتـابع write()، هنالك بعض الاختلافات البسيطة عمّا فعلناه سابقًا: علينا أن نفتح الملف في وضع الكتابة بالنظام الثنائي write binary بتمرير السلسلة النصية 'wb' كوسيط ثانٍ إلى الدالة open() حتى لو كان الملـف المـنزل نصـيًا (مثـل مسرحية روميو وجولييت في المثال أعلاه)، لأننا نريد كتابة البيانات الثنائيـة بـدلًا من البيانـات النصـية للحفـاظ على ترميز النص encoding.

لنستعمل حلقة for مع التابع iter_content() للكائن Response لكتابة صفحة الويب إلى ملف:

```
>>> import requests
>>> res =
requests.get('https://automatetheboringstuff.com/files/rj.txt')
>>> res.raise_for_status()
>>> playFile = open('RomeoAndJuliet.txt', 'wb')
>>> for chunk in res.iter_content(100000):
   playFile.write(chunk)


100000
78981
>>> playFile.close()
```

يعيد التابع iter_content() قطعًا من النص في كل تكرار لحلقـة التكـرار، وكـل قطعـة يكـون لهـا نـوع البيانات bytes وتحدد لها كم بايتًا يجب أن يكون طول كل قطعة، وأرى أن مئة ألف بايت هو حجم مناسب، لـذا لنمرر 100000 إلى التابع iter_content().

أنشأنا الآن الملف `RomeoAndJuliet.txt` في مجلد العمل الحالي، لاحظ أن اسم الملف في موقع الويب هو rj.txt بينما اسم الملف المحفوظ لدينا مختلف. تذكر أن الوحدة `requests` تنزل محتويات صفحات الويب لديك، وبعد تنزيلها يكون على عاتقك التعامل معها وحفظها إن شئت أينما تشاء.

> **ترميز يونيكود Unicode encoding**: ترميز يونيكود خارج عن نطاق هذا الكتاب، لكن يمكنك قراءة المزيد عنه في هذين المقالين Unicode.html وunipain.html.
>
> يعيد التابع `write()` عدد البايتات المكتوبة إلى ملف، وكتبنا في المثال السابق 100,000 بايت إلى الملف في أول «قطعة chunk» وبقي له 78,981 بايت للكتابة للمرة الثانية.

للمراجعة، هذه هي الخطوات الكاملة لتنزيل وحفظ ملف:

1. استدعاء التابع `requests.get()` لتنزيل ملف.
2. استدعاء `open()` مع الخيار `'wb'` لإنشاء ملف جديد وفتحه للكتابة في الوضع الثنائي.
3. المرور على التابع `iter_content()` للكائن `Response`.
4. استدعاء التابع `write()` في كل تكرار لكتابة المحتوى إلى الملف.
5. إغلاق الملف `close()`.

هذا كل ما يتعلق بالوحدة `requests`! قد تبدو لك حلقة `for` مع `iter_content()` معقدةً مقارنةً باستخدام `open()` و `write()` و `close()` التي استخدمناها لكتابة الملفات النصية؛ لكننا فعلنا ذلك لنضمن أن برنامجنا لن يستهلك ذاكرة كثيرة إذا نزلنا ملفات ضخمة. يمكنك قراءة المزيد حول ميزات الوحدة `requests` الأخرى من readthedocs.org.

12.3 لغة HTML

قبل أن نستخلص المعلومات من صفحات الويب، لنتعلم بعض أساسيات HTML أولًا، ولنرى كيف نصل إلى أدوات المطور في متصفح الويب، التي ستجعل عملية استخراج البيانات من الويب أمرًا سهلًا جدًا.

12.3.1 مصادر لتعلم HTML

لغة HTML هي الصيغة التي تكتب فيها صفحات الويب، ويفترض هذا الفصل أن لديك بعض الأساسيات حول HTML، لكن إن كنت تريد البدء من الصفر فأنصحك أن تراجع:

- توثيق HTML في موسوعة حسوب
- قسم HTML في أكاديمية حسوب

12.3.2 تذكرة سريعة

في حال لم تلمس HTML من مدة، سأخبرك بملخص بسيط عنها.

ملفات HTML هي ملفات نصية لها اللاحقة html، وتكون النصوص فيها محادثة بالوسوم tags، وكل وسم يكون ضمن قوسي زاوية <>، وتخبر هذه الوسوم المتصفحات كيف يجب أن تعرض الصفحة.

يمكن أن يكون هنالك نص بين وسم البداية ووسم النهاية، وهذا ما يؤلف عنصرًا element.

فمثلًا، الشيفرة الآتية تعرض لك `Hello, world!` في المتصفح، وتكون كلمة `Hello` بخط عريض:

```
<strong>Hello</strong>, world!
```

وستبدو في المتصفح كما في الشكل التالي:

الشكل 49: مثال Hello, world! معروضة في متصفح

وسم البداية `<strong>` يخبر المتصفح أن النص سيكون بخط عريض، ووسم النهاية `</strong>` يخبر المتصفح أين هي نهاية النص العريض. هنالك وسوم متنوعة في HTML، ولبعض تلك الوسوم خاصيات تُذكَر ضمن قوسَي الزاوية <>، مثلًا الوسم `<a>` يعني أن النص هو رابط، وتكون قيمة هذا الرابط محددة بالخاصية `href`. مثال:

```
Al's free <a href="https://inventwithpython.com">Python books</a>.
```

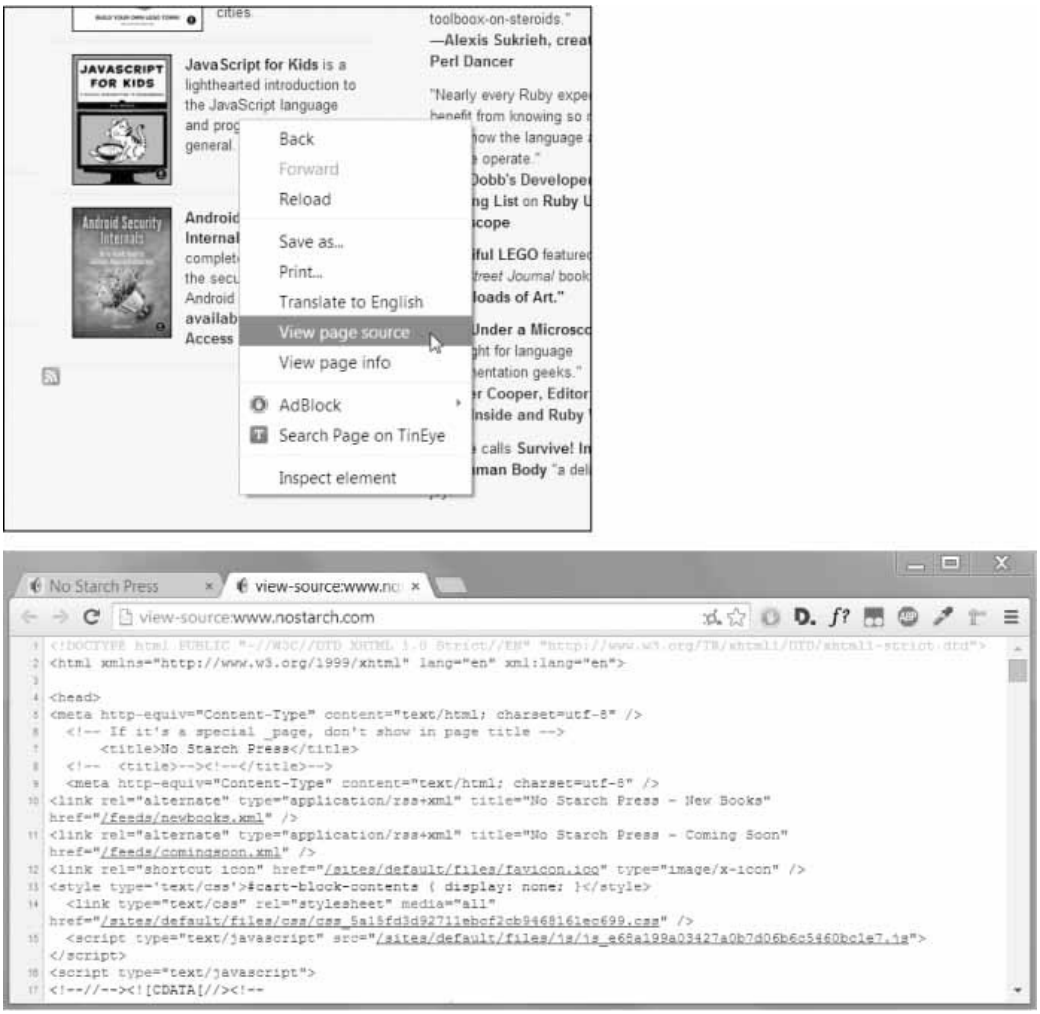
ستبدو صفحة الويب كما في الشكل التالي:

الشكل 50: رابط معروض في متصفح

تمتلك بعض العناصر الخاصية id التي تستخدم لتعريف عناصر الصفحة بشكل فريد، ويمكنك أن تطلب من برامجك البحث عن عنصر ما باستخدام معرفه id، لهذا تكون معرفة قيمة id لأحد العناصر من أهم الأمور التي سنستعمل فيها أدوات المطور أثناء كتابة برامج استخلاص البيانات من صفحات الويب.

12.3.3 عرض مصدر صفحة HTML

إذا أردت إلقاء نظرة على مصدر صفحة HTML لإحدى الصفحات التي تزورها، اضغط على الزر الأيمن للفأرة واختر View page source كما في الشكل التالي. هذا النص هو ما يحصل عليه متصفحك وهو يعرف كيف يعرض الصفحة اعتمادًا على ذاك النص.



الشكل 51: عرض مصدر صفحة ويب

أنصح -وبشدة- أن تجرب عرض مصدر صفحات HTML لبعض مواقعك المفضلة، حتى لو لم تفهم تمامًا كل ما تراها أمامك، فلست بحاجة إلى احتراف HTML لتعرف كيف تكتب برامج لاستخراج البيانات؛ فليس المطلوب منك برمجة موقعك الشخصي وإنما أن يكون لديك ما يكفي لتحصل البيانات المطلوبة.

12.3.4 فتح أدوات المطور

إضـافةً إلى عـرض المصـدر، يمكنـك أن تلقي نظـرة على صـفحات HTML باسـتخدام أدوات المطـور في متصفحك. يمكنك أن تضغط الزر F12 في كروم لإظهارها، أو الضغط على F12 مرة أخرى لإخفائها. يمكنك أيضًـا فتحها من القائمة الجانبية ثم Developer Tools، أو الضغط على ⌘-⌥-I في ماك.

الشكل 52: أدوات المطور في متصفح كروم

أمـا في فـيرفكس، فبامكنـك فتح أدوات المطـور من خلال الضـغط على Ctrl+Shift+C في ويـنـدوز ولينكس، أو ⌘-⌥-C في ماك، وأدوات المطور هنا تشبه كروم كثيرًا.

بعد تفعيل أدوات المطور في متصفحك، يمكنك الضغط بالزر الأيمن للفأرة على أي عنصر واختيار Inspect Element في القائمة المنبثقة لترى شيفرة HTML المسؤولة عن ذاك الجزء من الصـفحة. ستسـتفيد من ذلـك كثيرًا عندما تبدأ تفسير صفحات HTML لاستخراج البيانات منها.

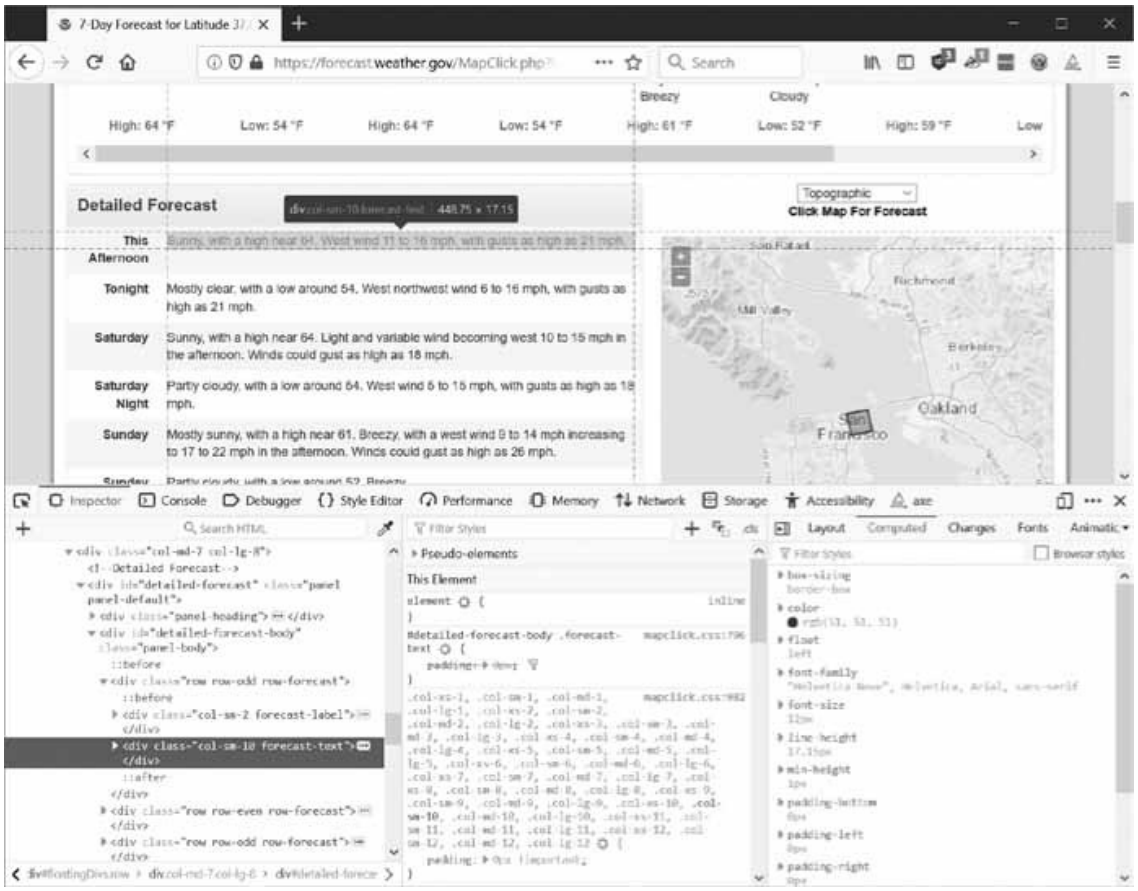
> لا تستعمل التعابير النمطية لتفسير HTML: قد تظن أن التعابير النمطية هي الأداة الأمثل العثور على سلسلة نصية معينة في HTML، لكنني أنصحك بعدم فعل ذلك، لأن هنالك طرائق كثيرة يمكننا كتابة شيفرات HTML سليمة، ومحاولة تغطية جميع حالات الاستخدام يدويًا باستخدام التعابير النمطية هو أمر مرهق ومعرض للخطأ كثيرًا. هنالك وحدات مطورة خصيصًا لتفسير شيفرات HTML مثل bs4، التي ستقلل كثيرًا من العلل والأخطاء وتسهل عليك العمل.

12.3.5 استخدام أدوات المطور للعثور على عناصر HTML

بعد أن نزل برنامج صفحة الويب باستخدام الوحدة `requests` فسيكون لدينا صفحة الويب كاملةً كسلسلة نصية واحدة، وعلينا أن نجد طريقة لمعرفة ما هي عناصر HTML التي تحتوي على المعلومات التي نريد استخراجها من الصفحة.

ستساعدنا هنا أدوات المطور في ذلك. لنقل مثلًا أننا نريد كتابة برنامج لجلب بيانات الطقس من موقع weather.gov. لكن قبل أن نبدأ بكتابة الشيفرات فعلينا القيام ببعض التحريات أولًا. إذا زرنا الموقع وبحثنا عن الرمز البريدي 94105 فستحصل على صفحة تظهر لك الطقس في تلك المنطقة.

ماذا إن كانت مهتمًا باستخراج بيانات الطقس لهذا العنوان البريدي؟ اضغط بالزر الأيمن على مكان معلومات الطقس في تلك الصفحة واختر Inspect Element من القائمة، وستظهر لك نافذة أدوات المطور، التي تريك شيفرات HTML المسؤولة عن ذاك الجزء من الصفحة كما يظهر في الشكل التالي. انتبه إلى أن الموقع قد يتغير دوريًا وقد تظهر لك عناصر مختلفة وعليك اتباع نفس الخطوات لتفحص العناصر الجديدة.



الشكل 53: تفحص عناصر صفحة الطقس باستخدام أدوات المطور

يمكننا أن نرى ما هي شيفرة HTML المسؤولة عن عرض الطقس في الصفحة السابقة، وهي:

```
<div class="col-sm-10 forecast-text">Sunny, with a high near 64. West
wind 11 to 16 mph, with gusts as high as 21 mph.</div>
```

هذا ما نبحث عنه تمامًا، يبدو أن معلومات الطقس موجودة داخل عنصر `<div>` له صنف CSS باسم `forecast-text`. اضغط بالزر الأيمن على العنصر في أدوات المطور واختر من القائمة CSS ◂ Copy Selector، وستُنسَخ لك سلسلة نصية مثل `'div.row-odd:nth-child(1) > div:nth-child(2)'` إلى الحافظة، ويمكنك أن تستخدم تلك السلسلة النصية مع التابع `select()` في BS4 أو `find_element_by_css_selector()` في Selenium كما هو مشروح في هذا الفصل.

الآن وبعد تعرفك على ما تبحث عنه تحديدًا، يمكن لوحدة `Beautiful Soup` مساعدتك في العثور عليه.

12.4 تفسير HTML مع وحدة BS4

تستخدم الوحدة `Beautiful Soup` في استخراج المعلومات من صفحة HTML، واسم الوحدة في بايثون هو `bs4` للإشارة إلى الإصدار الرابع منها، ويمكننا تثبيتها عبر الأمر `pip install --user beautifulsoup4` (راجع الملحق 1 لتعليمات حول تثبيت الوحدات). صحيح أننا استخدمنا `beautifulsoup4` لتثبيت الوحدة، لكن حين استيرادها في بايثون سنستعمل `import bs4`.

ستكون أمثلتنا عن BS4 عن تفسير ملف HTML (أي تحليلها والتعرف على أجزائها) مخزن محليًا على حاسوبنا. افتح لسانًا جديدًا في محرر Mu وأدخل ما يلي فيه واحفظه باسم `example.html`:

```
<!-- This is the example.html example file. -->

<html>
<head><title>The Website Title</title></head>
<body>
<p>Download my <strong>Python</strong> book from <a href="https://
inventwithpython.com">my website</a>.</p>
<p class="slogan">Learn Python the easy way!</p>
<p>By <span id="author">Al Sweigart</span></p>
</body>
</html>
```

نعم، حتى ملفات HTML البسيطة فيها مختلف العناصر والخاصيات، وستكون الأمور أعقد بكثير في المواقع الكبيرة، لكن مكتبة BS4 هنا لمساعدتنا وتسهيل الأمر علينا.

12.4.1 إنشاء كائن BeautifulSoup من سلسلة HTML نصية

يمكننا استدعاء الدالة `bs4.BeautifulSoup()` مع تمرير سلسلة نصية تحتوي على شيفرة HTML الـتي ستفسر. تعيد الدالة `bs4.BeautifulSoup()` كائن `BeautifulSoup`. جرب ما يلي في الصـدفة التفاعليـة على حاسوبك مع وجود إنترنت:

```
>>> import requests, bs4
>>> res = requests.get('https://academy.hsoub.com')
>>> res.raise_for_status()
>>> academySoup = bs4.BeautifulSoup(res.text, 'html.parser')
>>> type(academySoup)
<class 'bs4.BeautifulSoup'>
```

هذه الشيفرة تستعمل `requests.get()` لتنزيل صفحة الويب من موقع أكاديمية حسوب ثم تمرر قيمـة السمة text للرد إلى الدالة `bs4.BeautifulSoup()` التي تعيد كـائن `BeautifulSoup` نخزنـه في المتغـير `academySoup`.

يمكننـــا أيضًـــا تحميـــل ملـــف HTML من القـــرص لـــدينا بتمريـــر كـــائن `File` إلى الدالـــة `bs4.BeautifulSoup()` مع وسيط ثانٍ يخبر وحدة `BS4` ما المفسر الذي عليها استعماله لتفسير الملف.

أدخـل مـا يلي في الصـدفة التفاعليـة مـع التأكـد أنـك في نفس المجلـد الـذي يحتـوي على ملـف `example.html`:

```
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile, 'html.parser')
>>> type(exampleSoup)
<class 'bs4.BeautifulSoup'>
```

المفسر `'html.parser'` المستخدم هنا يأتي مع بايثون، لكن يمكنك استخدام المفسر `'lxml'` الأسرع إذا ثبتت الوحدة الخارجية `lxml`. اتبع التعليمات الموجودة في الملحق 1 لتثبيت الوحـدة باسـتخدام الأمـر `pip install --user lxml`. إذا لم نضـمِّن الوسـيط الثـاني الـذي يحـدد المفسـر فسـيظهر التحـذير UserWarning: No parser was explicitly specified.

بعد أن يكون لدينا كائن `BeautifulSoup` سنتمكن من استخدام توابعه من أجل تحديـد أجـزاء معينـة من مستند HTML.

12.4.2 العثور على عنصر باستخدام التابع select()

يمكنك الحصول على عنصر من عناصر صفحة الويب في الكائن BeatuifulSoup باستدعاء التابع select() وتمرير محدد CSS (أي CSS Selector) للعنصر الذي تبحث عنه. المحددات تشبه التعابير النمطية في وظيفتها: تحدد نمطًا يمكن البحث عنه في صفحات الويب.

نقاش محددات CSS خارج سياق هذا الكتاب، لكن هذه مقدمة مختصرة عنها في الجدول الآتي، وأنصحك بالاطلاع على توثيق المحددات في موسوعة حسوب.

المحدد الذي يمرر إلى التابع select()	سيطابق
`soup.select('div')`	جميع عناصر `<div>`
`soup.select('#author')`	جميع العناصر التي لها الخاصية `id` وقيمتها `author`
`soup.select('.notice')`	جميع العناصر التي لها خاصية `class` ولها صنف CSS باسم `notice`
`soup.select('div span')`	جميع عناصر `<span>` الموجود داخل عناصر `<div>`
`soup.select('div > span')`	جميع العناصر `<span>` الموجودة مباشرةً داخل عناصر `<div>` بدون وجود أي عنصر بينهما
`soup.select('input[name]')`	جميع عناصر `<input>` التي لها الخاصية `name` بغض النظر عن قيمتها
`soup.select('input[type="button"]')`	جميع عناصر `<input>` التي لها الخاصية `type` وتكون مساوية إلى `button`

الجدول 13: أمثلة عن محددات CSS

يمكن دمج مختلف أنماط المحددات مع بعضها لمطابقة أنماط أكثر تعقيدًا، فمثلًا `soup.select('p #author')` ستطابق أي عنصر له خاصية `id` قيمتها `author` لكن يجب أن يكون هذا العنصر موجودًا داخل عنصر `<p>`.

بدلًا من محاولة كتابة المحددات بنفسك، يمكنك الضغط بالزر الأيمن على أي عنصر في صفحة الويب واختيار Inspect Element، وحينما تفتح أدوات المطور اضغط بالزر الأيمن على عنصر HTML واختر Copy ‣ CSS Selector لنسخ محدد CSS إلى الحافظة لتستطيع لصقه في الشيفرة لديك.

التابع select() سيعيد قائمةً بعناصر Tag، وهذا ما تفعله وحدة BS4 لتمثيل عنصر HTML. هذه القائمة ستحتوي على كائن Tag لكل مطابقة للمحدد في كائن BeautifulSoup. يمكن تمرير كائن Tag إلى الدالة str() لعرض وسوم HTML التي تمثلها.

تمتلك قيم Tag أيضًا السمة attrs التي تظهر جميع خاصيات HTML للعنصر ممثلةً كقاموس. لنجرب على ملف example.html بإدخال ما يلي في الصدفة التفاعلية:

```
>>> import bs4
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile.read(), 'html.parser')
>>> elems = exampleSoup.select('#author')
>>> type(elems) # elems هي قائمة من كائنات Tag
<class 'list'>
>>> len(elems)
1
>>> type(elems[0])
<class 'bs4.element.Tag'>
>>> str(elems[0]) # تحويل الكائن إلى سلسلة نصية.
'<span id="author">Al Sweigart</span>'
>>> elems[0].getText()
'Al Sweigart'
>>> elems[0].attrs
{'id': 'author'}
```

ستستخرج الشيفرة السابقة العنصر الذي له id="author" في مستند HTML. سنستخدم select('#author') للحصول على قائمة بجميع العناصر التي لها id="author"، وسنخزن قائمة كائنات Tag في المتغير elems، وسيخبرنا التعبير len(elems) أن لدينا كائن Tag وحيد في القائمة، أي جرت عملية المطابقة لعنصر HTML وحيد.

استدعاء التابع getText() على العناصر سيعيد النص الموجود في العنصر، أي السلسلة النصية الموجودة بين وسم البدء والإغلاق.أما تمرير الكائن إلى الدالة str() سيعيد سلسلة نصيةً فيها وسمَي البداية والنهاية مع نص العنصر؛ أما السمة attrs فستعطينا قاموسًا فيه خاصيات العنصر كاملةً.

يمكنك أيضًا استخراج جميع عناصر <p> من كائن BeautifulSoup كما في المثال الآتي:

```
>>> pElems = exampleSoup.select('p')
>>> str(pElems[0])
'<p>Download my <strong>Python</strong> book from <a href="https://
inventwithpython.com">my website</a>.</p>'
>>> pElems[0].getText()
```

```
'Download my Python book from my website.'
>>> str(pElems[1])
'<p class="slogan">Learn Python the easy way!</p>'
>>> pElems[1].getText()
'Learn Python the easy way!'
>>> str(pElems[2])
'<p>By <span id="author">Al Sweigart</span></p>'
>>> pElems[2].getText()
'By Al Sweigart'
```

ستعطينا `select()` ثلاث مطابقات، والتي ستخزن في المتغير `pElems`. سنجرب استخدام `str()` على الكائنات `pElems[0]` و `pElems[1]` و `pElems[2]` لعرض العناصر كسلاسل نصية، وسنجرب أيضًا التابع `getText()` لإظهار نص تلك العناصر فقط.

12.4.3 الحصول على معلومات من خاصيات العنصر

يسهل علينا التابع `get()` الخاص بكائنات `Tag` الوصول إلى خاصيات العناصر، ونمرر لهذا التابع سلسلةً نصيةً باسم الخاصية `attribute` وسيعيد لنا قيمتها. لنجرب المثال الآتي على الملف `example.html`:

```
>>> import bs4
>>> soup = bs4.BeautifulSoup(open('example.html'), 'html.parser')
>>> spanElem = soup.select('span')[0]
>>> str(spanElem)
'<span id="author">Al Sweigart</span>'
>>> spanElem.get('id')
'author'
>>> spanElem.get('some_nonexistent_addr') == None
True
>>> spanElem.attrs
{'id': 'author'}
```

يمكننا استخدام `select()` للعثور على أي عناصر `<span>` ثم تخزين أول عنصر مطابق في المتغير `spanElem`، الذي سنمرر للتابع `get()` القيمة `'id'` للحصول على قيمة المعرف الخاصة به، وهي `'author'` في مثالنا.

12.5 مشروع: فتح جميع نتائج البحث

في كل مرة أبحث فيها في غوغل، لا أريد أن أرى نتيجة بحث واحدة فقط ثم أنتقل إلى النتيجة التي بعدها، بل أضغط بسرعة على الزر الأوسط للفأرة على كل الروابط (أو الضغط عليها مع زر Ctrl) لفتحها في لسان جديد وأقرؤها معًا لاحقًا. فعلت هذا الأمر كثيرًا في غوغل لدرجة أنني مللت من البحث ثم الضغط على كل الروابط واحدًا واحدًا. ماذا لو كتبت برنامجًا أستطيع عبره كتابة عبارة البحث وسيفتح لي متصفحًا فيه كل نتائج البحث الأولى مفتوحةً كل واحد منها في لسان في المتصفح؟

لنكتب سكربتًا يفعل ذلك مع صفحة نتائج بحث فهرس حزم بايثون في pypi.org. يمكن تعديل هذا البرنامج ليعمل على الكثير من المواقع الأخرى، لكن انتبه إلى أن محركات البحث غوغل و DockDockGo عادةً ما تصعب عملية استخراج نتائج البحث من مواقعها.

هذا ما يفعله البرنامج:

1. الحصول على عبارة البحث من سطر الأوامر
2. الحصول على صفحة نتائج البحث
3. فتح لسان متصفح جديد لكل نتيجة

هذا يعني أن على برنامجك أن يفعل ما يلي:

1. قراءة وسائط سطر الأوامر من `sys.argv`.
2. الحصول على صفحة نتائج البحث عبر الوحدة `requests`.
3. العثور على جميع روابط نتائج البحث.
4. استدعاء الدالة `webbrowser.open()` لفتحها في المتصفح.

لنبدأ البرمجة بفتح ملف جديد باسم `searchpypi.py` في محررنا.

12.5.1 الخطوة 1: الحصول على وسائط سطر الأوامر وطلب صفحة نتائج البحث

قبل أن نبدأ بالبرمجة، علينا أن نعرف ما هو رابط URL لصفحة نتائج البحث. انظر إلى شريط العنوان في المتصفح بعد إتمامك لعملية البحث، وسترى أن الرابط يشبه https://pypi.org/search/?q=SEARCH_TERM_HERE. يمكننا استخدام الوحدة `requests` لتنزيل هذه الصفحة، ثم استخدام BS4 للبحث عن الروابط في مستند HTML.

يمكننا في النهاية أن نستعمل وحدة `webbrowser` لفتح تلك الروابط في ألسنة جديدة.

يجب أن تكون الشيفرة كما يلي:

```
#! python3
# searchpypi.py  - فتح عدة نتائج بحث
import requests, sys, webbrowser, bs4
print('Searching...')    # عرض النص ريثما تحمل الصفحة
res = requests.get('https://pypi.org/search/?q='
+ ' '.join(sys.argv[1:]))
res.raise_for_status()
# TODO: .إعادة روابط أفضل نتائج البحث
# TODO: فتح لسان لكل نتيجة
```

سيحدد المستخدم كلمات البحث عبر تمريرها إلى سطر الأوامر أثناء تشغيل البرنامج، وستخزن في `sys.argv` كما رأينا في الفصول السابقة لهذا الكتاب.

12.5.2 الخطوة 2: العثور على كل النتائج

ستحتاج الآن إلى وحدة BS4 لاستخراج نتائج البحث من مستند HTML المنزل. لكن كيف يمكننا معرفة المحدد المناسب لحالتنا؟ ليس من المنطقي مثلًا البحث عن جميع عناصر `<a>` لوجود روابط كثيرة لا تهمنا؛ والحل هنا أن نفتح أدوات المطور في المتصفح وننظر إلى المحدد المناسب الذي سينتقي لنا الروابط التي نريدها فقط. بعد أن نبحث فسنجد بعض العناصر التي تشبه الآتي:

```
<a class="package-snippet" href="/project/pyautogui/">
```

ولا يهمنا إن كانت تلك العناصر معقدة، وإنما نحتاج إلى النمط الذي علينا استخدامه للبحث عن الروابط.

لنعدل شيفرتنا إلى:

```
#! python3
# searchpypi.py - .فتح عدة نتائج من جوجل
import requests, sys, webbrowser, bs4
--snip--
# .إعادة روابط أفضل نتائج البحث
soup = bs4.BeautifulSoup(res.text, 'html.parser')
# .فتح لسان لكل نتيجة
linkElems = soup.select('.package-snippet')
```

إذا ألقيت نظـرةً إلى عناصـر <a> فسـتجد أن جميـع نتـائج البحث لهـا الخاصـية class="package-snippet" وإذا بحثنا في كامل مصدر الصفحة فسنتأكد أن الصنف package-snippet ليس مسـتخدمًا إلا لروابط نتائج البحث. لا يهمنا ما هو الصنف package-snippet ولا ما يفعل، وإنما يهمنا كيف سنستفيد منه لتحديد عناصر <a> التي نبحث عنها.

لننشِـئ كـائن BeautifulSoup من صـفحة HTML الـتي نزلناهـا ونسـتخدم المحـدد '.package-snippet' لتحديد جميع عناصر <a> التي لها صنف CSS المحدد؛ ضع في ذهنك أن موقـع PyPI قـد يحـدث واجهته الرسومية وقد تحتاج إلى استخدام محدد CSS مختلف مستقبلًا، إن حدث ذلك فمرر المحدد الجديـد إلى التابع soup.select() وسيعمل البرنامج على ما يرام.

12.5.3 الخطوة 3: فتح نتائج البحث في المتصفح

لنخبر برنامجنا الآن أن يفتح لنا النتائج الأولى في متصفح الويب. أضف ما يلي إلى برنامجك:

```
#! python3
# searchpypi.py - فتح عدة نتائج بحث
import requests, sys, webbrowser, bs4
--snip--
# فتح لسان لكل نتيجة
linkElems = soup.select('.package-snippet')
numOpen = min(5, len(linkElems))
for i in range(numOpen):
  urlToOpen = 'https://pypi.org' + linkElems[i].get('href')
  print('Opening', urlToOpen)
  webbrowser.open(urlToOpen)
```

سيفتح البرنامج افتراضيًا أول خمسة روابط في المتصفح باستخدام الوحـدة webbrowser، لكن قـد يكـون ناتج بحث المسـتخدم أقـل من خمس نتـائج، فحينهـا ننظـر أيهمـا أقـل، 5 أم عـدد عناصـر القائمـة المعـادة من استدعاء التابع soup.select().

الدالة المضمنة في بايثون min() تعيد العدد الأصـغر من الوسـائط الممـررة إليهـا (والدالـة max() تفعـل العكس). يمكنك أن تستخدم الدالة min() لمعرفة إذا كان عدد الروابـط أقـل من 5 وتخـزين النـاتج في المتغـير numOpen، والذي ستستفيد منه في حلقة for عبر range(numOpen).

سنستخدم الدالة `webbrowser.open()` في كل تكرار لحلقة `for` لفتح الرابط في المتصفح. لاحظ أن قيمة الخاصية `href` في عناصر `<a>` لا تحتوي على `https://pypi.org` لذا علينا إضافتها بأنفسنا إلى قيمة الخاصية `href` قبل فتح الرابط.

يمكنك الآن فتح أول 5 نتائج بحث عن `«boring stuff»` في محرك بحث PyPI بكتابة الأمر `searchpypi boring stuff` في سطر الأوامر (راجع الملحق 2 لشرح تشغيل برامجك ببساطة في نظام تشغيلك).

12.5.4 أفكار لبرامج مشابهة

من الجميل أن يكون لدينا برنامج يفتح لنا عدة ألسنة في المتصفح لأي عملية روتينية مكررة، مثل:

- فتح جميع صفحات المنتجات في متجر أمازون بعد البحث عن منتج معين.
- فتح كل روابط المراجعات لمنتج ما.
- فتح الصور الناتجة بعد إجراء عملية بحث سريعة على أحد مواقع الصور مثل Flickr.

12.6 مشروع: تنزيل كل رسمات XKCD

عادةً ما تحدث المواقع والمدونات الصفحة الرئيسية لها بعرض آخر منشور فيها، مع وجود زر «السابق» الذي يأخذك إلى المنشور السابق، ثم تجد في المنشور السابق زرًا يأخذك للمنشور الذي قبله، وهلم جرًا، مما ينشِئ سلسلةً من الصفحات التي تأخذك من الصفحة الرئيسية إلى صفحة أول منشور في الموقع.

إذا أردت نسخةً من محتوى موقعٍ ما لتقرأها وأنت غير متصل بالإنترنت، فيمكنك أن تفتح بنفسك كل صفحة وتحفظها، لكن هذا الأمر ممل جدًا ومن المناسب كتابة برنامج لأتمتة هذه المهمة.

موقع XKCD هو موقع كوميكس له نفس البنية المذكورة. وصفحته الرئيسية xkcd.com فيها زر Prev للعودة إلى الكوكميس السابقة، لكن عملية تنزيل كل الكوميكس واحدةً واحدةً تأخذ وقتًا كثيرًا، لكننا نستطيع كتابة سكربت لأتمتة ذلك في ثوانٍ معدودة.

هذا ما سيفعله برنامجنا:

1. فتح صفحة XKCD الرئيسية.
2. حفظ صورة الكوميكس في تلك الصفحة.
3. الانتقال إلى صفحة الكوميكس السابق.
4. تكرار العملية حتى الوصول إلى أول صورة كوميكس.

هذا يعني أن الشيفرة البرمجية ستفعل ما يلي:

1. تنزيل الصفحات باستخدام الوحدة `requests`.
2. العثور على رابط URL لصورة الكوميكس باستخدام وحدة `BS4`.
3. تنزيل وحفظ صورة الكوميكس إلى نظام الملفات باستخدام `iter_content()`.
4. العثور على رابط صورة الكوميكس السابقة، وتكرار العملية كلها.

افتح ملفًا جديدًا في محررك وسمِّه باسم `downloadXkcd.py`.

12.6.1 الخطوة 1: تصميم البرنامج

إذا فتحت أدوات المطور في المتصفح ونظرت إلى عناصر الصفحة، فسترى ما يلي:

- رابط URL لملف صورة الكوميكس في الخاصية `href` في العنصر `<img>`.
- العنصر `<img>` موجود داخل العنصر `<div id="comic">`.
- الزر `Perv` له الخاصية `rel` وقيمتها `prev`.
- صفحة أول صورة كوميكس يشير فيها الزر `Prev` إلى الرابط `https://xkcd.com/#` مما يشير إلى عدم وجود صفحات سابقة. عدّل الشيفرة لتبدو كما يلي:

```
#! python3
# downloadXkcd.py - تنزيل كل صورة كوميكس في XKCD
import requests, os, bs4
url = 'https://xkcd.com'                    # فتح رابط الصفحة
os.makedirs('xkcd', exist_ok=True)     # حفظ صور الكوميكس في صفحة /xkcd

while not url.endswith('#'):
  # TODO: تحميل الصفحة
  # TODO: العثور على رابط URL لملف صورة الكوميكس
  # TODO: تحميل الصورة
  # TODO: حفظ الصورة إلى ./xkcd
  # TODO: الحصول على رابط الصفحة السابقة
print('Done.')
```

سيكون لدينا متغير اسمه `url` يبدأ بالقيمة `'https://xkcd.com'` وتحدث قيمتـه دوريًـا ضـمن حلقـة `for` لتصبح الرابط الموجود في الزر `Prev` للصفحة الحالية. وسننزل صورة الكوميكس في كل تكرار من تكرارات حلقة `for` الموجودة في الرابط url، وسننتهي من حلقة التكرار حينما تنتهي قيمة url بالعلامة `'#'`.

سننزل ملفات الصور إلى مجلد موجود في مجلد العمل الحالي باسم `xkcd`، وسنتأكد من أن المجلد موجود باستدعاء `os.makedirs()` مع تمرير وسيط الكلمات المفتاحية `exist_ok=True` الذي يمنع الدالة من رمي استثناء إن كان المجلد موجودًا مسبقًا. بقية الشيفرة هو تعليقات سنملؤها في الأقسام القادمة.

12.6.2 الخطوة 2: تنزيل صفحة الويب

لنكتب الشيفرة الآتية التي ستنزل الصفحة:

```
#! python3
# downloadXkcd.py - تنزيل كل صورة كوميكس في XKCD
import requests, os, bs4
url = 'https://xkcd.com'                    # فتح رابط الصفحة
os.makedirs('xkcd', exist_ok=True)     # حفظ صور الكوميكس في صفحة /xkcd

while not url.endswith('#'):
  # تحميل الصفحة
  print('Downloading page %s...' % url)
  res = requests.get(url)
  res.raise_for_status()
  soup = bs4.BeautifulSoup(res.text, 'html.parser')
  # TODO: العثور على رابط URL لملف صورة الكوميكس
  # TODO: تحميل الصورة
  # TODO: حفظ الصورة إلى ./xkcd
  # TODO: الحصول على رابط الصفحة السابقة
print('Done.')
```

لنطبع بدايةً قيمة url ليعرف المستخدم ما هو رابط URL الذي يحاول البرنامج تنزيله، ثم سنسـتخدم الدالـة `requests.get()` في الوحدة `requests` لتنزيل الصفحة. ثم سنستدعي التـابع `raise_for_status()`

كالعادة لرمي استثناء إن حدثت مشكلة ما في التنزيل؛ ثم إن سارت الأمور على ما يرام فسننشِئ كائن `BeautifulSoup` من نص الصفحة المنزلة.

12.6.3 الخطوة 3: البحث عن صورة الكوميكس وتنزيلها

لنعدل شيفرة برنامجنا:

```
#! python3
# downloadXkcd.py - تنزيل كل صورة كوميكس في XKCD
import requests, os, bs4

--snip--

  # العثور على رابط URL لملف صورة الكوميكس
  comicElem = soup.select('#comic img')
  if comicElem == []:
    print('Could not find comic image.')
  else:
    comicUrl = 'https:' + comicElem[0].get('src')
    # تحميل الصفحة
    print('Downloading image %s...' % (comicUrl))
    res = requests.get(comicUrl)
    res.raise_for_status()
  # TODO: حفظ الصورة إلى ./xkcd
  # TODO: الحصول على رابط الصفحة السابقة

print('Done.')
```

بعد تفحص صفحة XKCD عبر أدوات المطور، سنعرف أن عنصر `<img>` لصورة الكوميكس موجود داخل عنصر `<div>` له الخاصية `id` التي قيمتها هي `comic`، وبالتالي سيكون المحدد `'#comic img'` صحيحًا لتحديد العنصر `<img>` عبر كائن `BeautifulSoup`.

تمتلك بعض صفحات موقع XKCD على محتوى خاص لا يمثل صورة بسيطة، لكن لا بأس فيمكننـا تخطي تلك الصفحات، فإن لم يطـابِق المحـدد الخـاص بنـا أي عنصـر فسـيعيد اسـتدعاء `soup.select('#comic img')` قائمة فارغة، وحينها سيظهر برنامجنا رسالة خطأ ويتجاوز الصفحة دون تنزيل الصورة.

عدا ذلك، فسيعيد المحدد السابق قائمةً فيها عنصر `<img>` وحيد، ويمكننـا الحصـول على قيمـة الخاصـية `src` لعنصر `<img>`ونمررها إلى `requests.get()` لتنزيل صورة الكوميكس.

12.6.4 الخطوة 4: حفظ الصورة والعثور على الصفحة السابقة

لنعدل الشيفرة لتصبح كما يلي:

```
#! python3
# downloadXkcd.py - تنزيل كل صورة كوميكس في XKCD
import requests, os, bs4

--snip--


    # حفظ الصورة إلى ./xkcd
    imageFile = open(os.path.join('xkcd', os.path.basename(comicUrl)),
'wb')
    for chunk in res.iter_content(100000):
      imageFile.write(chunk)
    imageFile.close()


  # الحصول على رابط الصفحة السابقة
  prevLink = soup.select('a[rel="prev"]')[0]
  url = 'https://xkcd.com' + prevLink.get('href')
print('Done.')
```

أصبح لدينا ملف صورة الكوميكس مخزنًا في المتغير `res`، وعلينا كتابة بيانات هـذه الصـورة إلى ملـف في نظام الملفات المحلي لدينا. سنحدد اسم الملف المحلي للصورة ونمـرره إلى الدالـة `open()`، لاحـظ أن المتغـير `comicUrl` يحتوي على قيمة تشبه القيمة الآتية:

```
'https://imgs.xkcd.com/comics/heartbleed_explanation.png'
```

ويبدو أنك لاحظت وجود مسار الملف فيها.

يمكنك استخدام الدالة `os.path.basename()` مع `comicUrl` لإعادة آخر جزء من رابـط URL السـابق، أي `'heartbleed_explanation.png'`، ويمكنك حينها استخدام هذا الاسم حين حفظ الصـورة إلى نظـام الملفات المحلي.

يمكنك إضافة هذا الاسم لاسم مجلد `xkcd` عبر الدالة `os.path.join()` كما تعلمنا سابقًا لكي يسـتخدم برنامجك الفاصل الصحيح (الخط المائل الخلفي `\` في ويندوز، والخط المائل `/` في لينكس ومـاك). أصـبح لـدينا الآن اسم ملف صحيح، ويمكننا استدعاء الدالة `open()` لفتح ملف جديد بوضع الكتابة الثنائية `'wb'`.

إذا كنت تذكر حينما حفظنا الملفات التي نزلناها عبر الوحدة `requests` في بدايـة الفصـل كنـا نمـر بحلقـة تكرار على القيمة المعادة من التابع `iter_content()`، وستكتب الشيفرة الموجودة في حلقة `for` قطعًـا من بيانات الصورة (كحد أقصى 100,000 بايت كل مرة) إلى الملـف، ثم تغلـق الملـف. أصـبحت الصـورة محفوظـة لديك محليًا الآن!

علينا بعدئذٍ استخدام المحدد `'a[rel="prev"]'` لتحديد العنصر `<a>` الذي له العلاقة `rel` مضبوطةً إلى `prev`. يمكنـك اسـتخدام قيمـة الخاصـية `href` لعنصـر `<a>` المحـدد للحصـول على رابـط صـورة الكـوميكس السابقة، والتي ستخزن في المتغير `url`، ثم ستدور حلقة `while` مرة أخرى وتعيد تنفيذ عملية التنزيل من جديد على الصفحة الجديدة. سيبدو ناتج تنفيذ البرنامج كما يلي:

```
Downloading page https://xkcd.com...
Downloading image https://imgs.xkcd.com/comics/phone_alarm.png...
Downloading page https://xkcd.com/1358/...
Downloading image https://imgs.xkcd.com/comics/nro.png...
Downloading page https://xkcd.com/1357/...
Downloading image https://imgs.xkcd.com/comics/free_speech.png...
Downloading page https://xkcd.com/1356/...
Downloading image
https://imgs.xkcd.com/comics/orbital_mechanics.png...
Downloading page https://xkcd.com/1355/...
Downloading image https://imgs.xkcd.com/comics/airplane_message.png...
Downloading page https://xkcd.com/1354/...
Downloading image
https://imgs.xkcd.com/comics/heartbleed_explanation.png...
--snip--
```

هذا المشروع هو مثال ممتاز لبرامج تتبع الروابط تلقائًا لاستخراج كمية معلومات كبيرة من الـويب. يمكنـك تعلم المزيد من المعلومات حول ميزات `Beautiful Soup` الأخرى من توثيقها الرسمي.

12.6.5 أفكار لبرامج مشابهة

تنزيل صفحات الويب وتتبع الروابط فيها هـو أسـاس أي برنـامج زحـف crawler. يمكنـك أن تـبرمج بـرامج تفعل ما يلي:

- تنسخ موقعًا كاملًا احتياطيًا.
- تنسخ جميع التعليقات من أحد المنتديات.
- تعرض قائمة بجميع المنتجات التي عليها تخفيضات في أحد المتاجر.

الوحدتان `requests` و `bs4` رائعتين جدًا، لكن عليـك أن تعـرف مـا هـو رابـط URL المناسـب لتمريـره إلى `requests.get()`، لكن في بعض الأحيان قد لا يكون ذلـك سـهلًا؛ أو ربمـا عليـك تسـجيل الـدخول إلى أحـد المواقع قبل بدء عملية الاستخراج، لهذا السبب سنستكشف سويةً الوحدة `selenium` لأداء مهام معقدة.

12.7 التحكم في المتصفح عبر الوحدة selenium

تسمح الوحدة `selenium` لبايثون بالتحكم برمجيًا بالمتصـفح بضـغط الروابـط وتعبئـة الاسـتمارات، مثلهـا كمثل أي تفاعل من مستخدم بشري. يمكننـا أن نتفاعـل مـع صـفحات الـويب بطرائـق أكـثر تقـدمًا في الوحـدة `selenium` مقارنة مـع `requests` و `bs4`، لكنهـا قـد تكـون أبطـأ وأصـعب بالتشـغيل في الخلفيـة لأننـا نفتح متصفح ويب كامل مقارنة بتنزيل بعض الصفحات من الويب.

لكن إن كان علينا التعامـل مـع صـفحات الـويب بطريقـة تعتمـد على شـيفرات JavaScript الـتي تحـدث الصفحة، فعلينا استخدام `selenium` بدلًا من `requests`. أضف إلى ذلـك أن المواقـع الشـهيرة مثـل أمـازون تستخدم برمجيات للتعرف على الزيارات الآتيـة من سـكربتات الـتي تحـاول اسـتخراج البيانـات من صـفحاتها أو إنشـاء عـدّة حسـابات مجانيـة، ومن المـرجح أن تحجب هـذه المواقـع برامجـك بعـد فـترة؛ وهنـا تـأتي الوحـدة `selenium` التي تعمل مثل متصفحات الويب العادية.

أحد أشهر العلامات التي تتعرف فيها المواقـع أن الزيـارات آتيـة من برنـامج (أو سـكربت script) هي عبـارة `user-agent`، التي تُعـرِّف متصـفح الـويب المسـتخدم وتضـمَّن في كـل طلبيـات HTTP. فمثلًا تكـون عبـارة `user-agent` للوحدة `requests` شيء يشبه `'python-requests/2.21.0'`.

يمكنـك زيـارة موقـع مثـل whatsmyua.info لتعـرف مـا هي `user-agent` الخاصـة بـك. أمـا الوحـدة `selenium` فمن المـرجح أن تظن المواقــع أنهــا مســتخدم بشــري، لأنهــا تســتخدم `user-agent` شــبيه بالمتصـفحات العاديـة مثـل `'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:124.0)`

'Gecko/20100101 Firefox/124.0' ولأن نمط التصفح فيها يشبه المتصفحات الأخرى، فستُنزَّل الصور والإعلانات وملفات تعريف الارتباط والمتتبعات كما في المتصفحات العادية. لكن يمكن للمواقع كشف selenium وتحاول شركات شراء بطاقات الدخول والمتاجر الإلكترونية حجب أي سكربتات تستعمل متصفح selenium.

12.7.1 تشغيل متصفح تتحكم فيه selenium

سنرى في الأمثلة القادمة كيفية التحكم في متصفح فيرفكس من selenium، إذا لم يكن مثبتًا لديك فيمكنك تنزيله من getfirefox.com، ويمكنك تثبيت selenium،من سطر الأوامر بتشغيل pip install --user selenium، وأذكرك بالذهاب إلى الملحق 1 لمزيد من المعلومات.

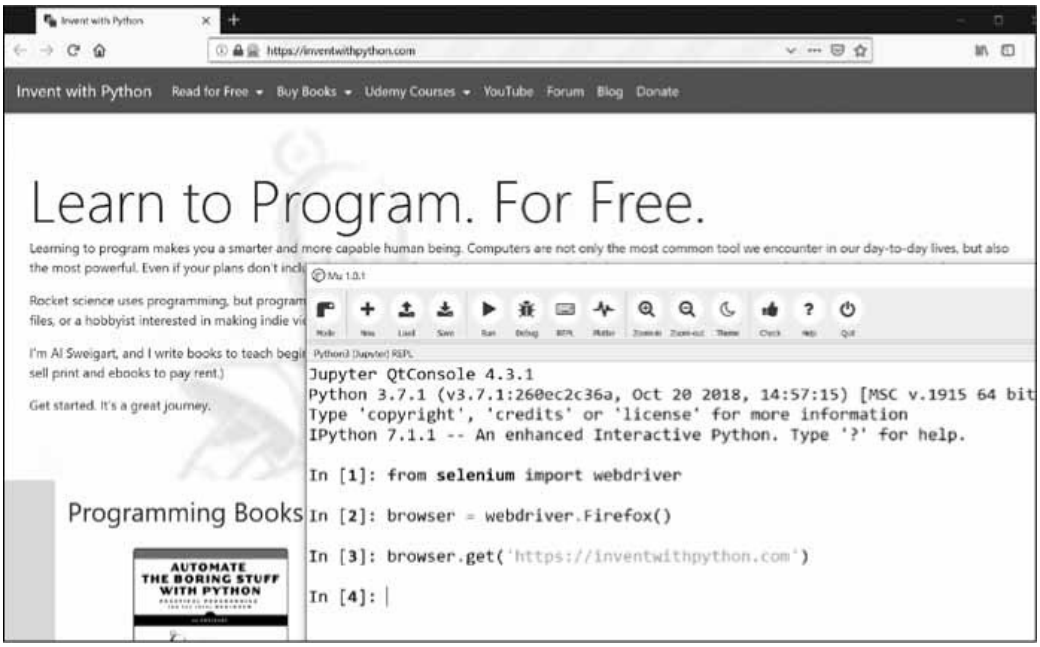
قد يكون استيراد مكونات selenium مختلفًا عليك، فبدلًا من from selenium import سنستعمل selenium import webdriver (سبب فعل selenium لذلك خارج نطاق هذا الكتاب لا تقلق حوله).

يمكنك بعد ذلك تشغيل فايرفوكس مع selenium بكتابة ما يلي في الصدفة التفاعلية:

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> type(browser)
<class 'selenium.webdriver.firefox.webdriver.WebDriver'>
>>> browser.get('https://inventwithpython.com')
```

ستلاحظ أن متصفح فايرفوكس قد بدأ بالعمل حين استدعاء webdriver.Firefox().

استدعاء type() على webdriver.Firefox() سيظهر أنها من نوع البيانات WebDrive،أما استدعاء browser.get('https://inventwithpython.com') فسوف يوجه المتصفح إلى https://inventwithpython.com/.



الشكل 54: بعد استدعاء webdriver.Firefox() و get() ستفتح الصفحة في متصفح فايرفوكس

إذا واجهت مشكلة من قبيل `'geckodriver' executable needs to be in PATH` فهذا يعني أن عليك تنزيل محرك فايرفوكس يدويًا قبل استخدام `selenium` للتحكم به. يمكنك التحكم بمتصفحات أخرى غير فيرفكس إذا ثبتت محرك الويب لهم `webdriver`.

لمتصفح فايرفوكس، اذهب إلى github.com وثبت محرك geckodriver لنظام تشغيلك. (كلمة «Gecko» هي اسم محرك المتصفح engine في فيرفكس). سيحتوي الملف المضغوط المنزل على `geckodriver.exe` في ويندوز أو `geckodriver` في ماك ولينكس، وعليك وضعه في مسار PATH الخاص في نظامك، راجع الإجابة الآتية stackoverflow.com.

أما لمتصفح كروم فاذهب إلى chromium.org ونزل الملف المضغوط المناسب لنظامك، وضع الملف `chromedriver.exe` أو `chromedriver` في مسار PATH كما ذكرنا أعلاه.

يمكن فعل نفس الأمر لبقية المتصفحات الرئيسية، ابحث سريعًا في الويب عن اسم المتصفح متبوعًا بالكلمة `webdriver` وستجدها.

قد تواجه مشاكل في تشغيل المتصفحات الجديدة في `selenium`، بسبب عدم التوافقية بينهما، لكن يمكنك إجراء حل التفافي بتثبيت نسخة قديمة من المتصفح أو من وحدة `selenium`. يمكنك معرفة تاريخ الإصدارات من pypi.org. للأسف قد تحدث مشاكل توافقية بين `selenium` وبعض إصدارات المتصفحات، وأنصحك حينها بالبحث في الويب عن الحلول المقترحة، وراجع الملحق 1 لمعلومات حول تثبيت إصدار محدد من `selenium` (مثل تثبيت `pip install --user -U selenium==3.14.1`).

12.7.2 العثور على عناصر في الصفحة

توفر كائنات WebDriver عددًا من التوابع للعثـور على عناصـر صـفحة، ويمكن تقسـيمها إلى قسـمين من التوابع `find_elements_*` و `find_element_*`. توابـع `find_element_*` تعيـد كـائن `WebElement` وحيد يمثل أول عنصر في الصفحة يطابق الطلبية التي حددتها، بينما تعيـد توابـع `find_elements_*` قائمـة من كائنات `WebElement_*` لكل عنصر مطابق في الصفحة.

يظهر الجدول الآتي أمثلةً على التوابع `find_elements_*` و `find_elements_*` المستدعاة على كـائن `WebDriver` مخزن في المتغير `browser`.

اسم التابع	كائن/ قائمة كائنات WebElement
`browser.find_element_by_class_name(name)` `browser.find_elements_by_class_name(name)`	العناصر التي تستخدم صنف CSS باسم `name`
`browser.find_element_by_css_selector(selector)` `browser.find_elements_by_css_selector(selector)`	العناصر التي تطابق محدد CSS معين
`browser.find_element_by_id(id)` `browser.find_elements_by_id(id)`	العناصر التي تطابق id معين
`browser.find_element_by_link_text(text)` `browser.find_elements_by_link_text(text)`	عناصر `<a>` التي يطابق محتواها القيمة `text` كاملةً
`browser.find_element_by_partial_link_text(text)` `browser.find_elements_by_partial_link_text(text)`	عناصر `<a>` التي يطابق محتواها القيمة `text` جزئيًا
`browser.find_element_by_name(name)` `browser.find_elements_by_name(name)`	العناصر التي لها الخاصية `name` وتطابق قيمة معينة
`browser.find_element_by_tag_name(name)` `browser.find_elements_by_tag_name(name)`	العناصر التي تطابق وسمًا معينًا، وهي غير حساسة لحالة الأحرف (أي `<a>` يمكن أن يطابق عبر `'a'` أو `'A'`)

الجدول 14: توابع WebDriver للعثور على العناصر في selenium

باستثناء التوابع `*_by_tag_name()` تكون جميع الوسائط الممررة إلى الدوال حساسةً لحالة الأحرف، وإذا لم تكن العناصر موجـودة في الصـفحة فسـتطلق `selenium` الاسـتثناء `NoSuchElement`، وعليـك اسـتخدام عبارات `try` و `except` إذا لم ترغب بتوقف برنامج عن العمل عند ذلك.

بعد أن تحصل على كائن WebElement فيمكنك أن تتعرف على المزيد حوله بقراءة سماته أو استدعاء توابع المذكورة في الجدول الآتي.

السمة أو التابع	الوصف
tag_name	اسم الوسم، مثل 'a' لعنصر <a>
get_attribute(name)	قيمة خاصية name للعنصر
text	النص الموجود داخل العنصر، مثل 'hello' في hello
clear()	مسح النص المكتوب في الحقول النصية
is_displayed()	إعادة True إذا كان العنصر ظاهرًا، وإلا فيعيد False
is_enabled()	إعادة True إذا كان حقل الإدخال مفعلًا، وإلا فيعيد False
is_selected()	لأزرار الانتقاء radio ومربعات التأشير checkbox ستعيد True إذا كان العنصر مختار، وإلا فتعيد False
location	قاموس فيه المفاتيح 'x' و 'y' لمكان العنصر في الصفحة

الجدول 15: سمات وتوابع الكائن WebElement

مثلًا، افتح ملفًا جديدًا واكتب البرنامج الآتي:

```
from selenium import webdriver
browser = webdriver.Firefox()
browser.get('https://inventwithpython.com')
try:
  elem = browser.find_element_by_class_name(' cover-thumb')
  print('Found <%s> element with that class name!' % (elem.tag_name))
except:
  print('Was not able to find an element with that name.')
```

فتحنا هنا متصفح فيرفكس ووجهناه إلى رابط URL معين، وحاولنا العثور على العناصر التي لها الصنف bookcover، وإذا عثرنا على أحد العناصر فسنطبع اسم الوسم عبر السمة tag_name؛ أما لو لم يعثر على أي عنصر فسنطبع رسالة مختلفة. سيعيد البرنامج الناتج الآتي:

```
Found <img> element with that class name!
```

أي عثرنا على عنصر `<img>` له اسم الصنف `'bookcover'`.

12.7.3 الضغط على عناصر الصفحة

تمتلك كائنات `WebElement` المعادة من التوابع `find_element_*` و `find_elements_*` التابع `click()` الذي يحاكي ضغطات الفأرة على العنصر، ويمكن استخدام هذا التابع للضغط على رابط أو تحديد زر انتقاء أو الضغط على زر الإرسال أو أي حدث آخر يُطلَق عند الضغط على أحد العناصر. جرب المثال الآتي في الطرفية التفاعلية:

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('https://inventwithpython.com')
>>> linkElem = browser.find_element_by_link_text('Read Online for
Free')
>>> type(linkElem)
<class 'selenium.webdriver.remote.webelement.FirefoxWebElement'>
>>> linkElem.click() # محاكاة الضغط على رابط النص "Read Online for Free"
```

سيفتح متصفح فيرفكس الصفحة ويحصل على العنصر `<a>` الذي يحتوي على النص Read it Online ويحاكي الضغط على الرابط كما لو ضغطته بنفسك.

12.7.4 تعبئة وإرسال الاستمارات

يمكن تعبئة الحقول النصية مثل `<input>` أو `<textarea>` بالبحث عنها ثم استدعاء التابع `send_keys()`. جرب ما يلي في الطرفية التفاعلية:

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('https://login.metafilter.com')
>>> userElem = browser.find_element_by_id('user_name)
>>> userElem.send_keys('your_real_username_here')
>>> passwordElem = browser.find_element_by_id('user_pass')
>>> passwordElem.send_keys('your_real_password_here')
>>> passwordElem.submit()
```

لطالما لم تغيّر صفحة MetaFilter المعرف `id` لحقول اسم المستخدم وكلمة المرور بعد نشر هذا الكتاب، فيمكنك استخدام الشيفرة السابقة لملء تلك الحقول النصية (تذكر أنك تستطيع استخدام أدوات المطور للتأكد من المعرف `id` للحقول في أي وقت).

استدعاء التابع `submit()` على أي عنصر في الاستمارة له نفس تأثير الضغط على زر الإرسال.

> **تحذير**: ابتعد عن تخزين كلمات المرور الخاصة بك في الشيفرة المصدرية لبرامجك قدر الإمكان، فمن السهل تسريب كلمات المرور إذا تركناها دون تشفير. يمكن لبرنامجك أن يطلب كلمة المرور من المستخدم أثناء التشغيل كما ناقشنا في الفصل الثامن.

12.7.5 إرسال المفاتيح الخاصة

تمتلك `selenuim` وحدةً للمفاتيح الخاصة التي لا يمكن كتابتها كسلسلة نصية، مثل زر Tab أو الأسهم. تلك القيم مخزنة كسمات في الوحدة `selenium.webdriver.common.keys`، ولأن اسم الوحدة طويل جدًا فمن الأسهل كتابة الآتي في بداية البرنامج:

```
from selenium.webdriver.common.keys import Keys
```

وحينها تستطيع استخدام `Keys` في أي مكان تريد أن تكتب فيه سمات الوحدة التي أشرنا لها أعلاه: `selenium.webdriver.common.keys`.

يعرض الجدول الآتي أشهر القيم المستخدمة.

السمة	الشرح
`Keys.DOWN` و `Keys.UP` و `Keys.LEFT` و `Keys.RIGHT`	الأسهم في لوحة المفاتيح
`Keys.ENTER` و `Keys.RETURN`	زر Enter
`Keys.HOME` و `Keys.END` و `Keys.PAGE_DOWN` و `Keys.PAGE_UP`	أزرار Home و End و PageDown و PageUp على التوالي
`Keys.ESCAPE` و `Keys.BACK_SPACE` و `Keys.DELETE`	أزرار Escape و Backspace و Delete على التوالي
`Keys.F1` و `Keys.F2` و . . . و `Keys.F12`	الأزرار F1 حتى F12 في الصف العلوي من لوحة المفاتيح
`Keys.TAB`	زر Tab

الجدول 16: أشهر القيم المستخدمة في الوحدة selenium.webdriver.common.keys

فمثلًا لو لم يكن المؤشر موجودًا داخل حقل نصي، فالضغط على زر Home و End سيؤدي إلى التمرير إلى بداية ونهاية الصفحة على التوالي. جرب ما يلي على الصدفة التفاعلية ولاحظ كيف يؤدي استدعاء `send_keys()` إلى تمرير الصفحة:

```
>>> from selenium import webdriver
>>> from selenium.webdriver.common.keys import Keys
>>> browser = webdriver.Firefox()
>>> browser.get('https://nostarch.com')
>>> htmlElem = browser.find_element_by_tag_name('html')
>>> htmlElem.send_keys(Keys.END)      # التمرير إلى نهاية الصفحة
>>> htmlElem.send_keys(Keys.HOME)     # التمرير إلى بداية الصفحة
```

العنصر `<html>` موجود في كل ملفات HTML، ويكون كامل محتوى مستند HTML داخله؛ وتحديد هذا العنصر مفيد لو أردنا إرسال المفاتيح إلى صفحة الويب عمومًا، وهذا بدوره مفيد إذا كانت الصفحة تحمِّل محتوى جديد عند التمرير إلى أسفل الصفحة.

12.7.6 الضغط على أزرار المتصفح

يمكن أن تحاكي الوحدة `selenium` الضغط على أزرار المتصفح المختلفة:

- التابع `browser.back()` يضغط على زر الرجوع.
- التابع `browser.forward()` يضغط على زر إلى الأمام.
- التابع `browser.refresh()` يضغط على زر التحديث.
- التابع `browser.quit()` يضغط على زر إغلاق النافذة.

12.7.7 المزيد من المعلومات حول Selenium

يمكن للوحدة `selenium` فعل الكثير مما لم نذكره هنا، فيمكنك أن تعدل محتوى ملفات تعريف الارتباط، وتأخذ لقطة شاشة، وتشغل سكربت JavaScript مخصص ...إلخ. لمزيد من المعلومات حول تلك الخصائص فأنصحك أن تزور التوثيق الرسمي.

12.8 أسئلة للتدريب

1. اشرح باختصار الفروقات بين الوحدات `webbrowser` و `requests` و `bs4` و `selenium`.
2. ما هو نوع الكائن المعاد من `requests.get()`؟ كيف يمكنك الحصول إلى المحتوى المنزل كقيمة نصية؟
3. ما هو تابع `requests` الذي يتأكد أن عملية التنزيل ناجحة؟
4. كيف تحصل على رمز حالة HTTP في رد `requests`؟

5. كيف تحفظ ناتج `requests` إلى ملف؟
6. ما هو اختصار فتح أدوات المطور في المتصفحات؟
7. كيف يمكنك عرض شيفرة HTML لأحد عناصر الصفحة في أدوات المطور؟
8. ما هو محدد CSS الذي يستطيع العثور على عنصر له المعرف `id` بقيمة `main`؟
9. ما هو محدد CSS الذي يعثر على العناصر التي لها صنف CSS بقيمة `highlight`؟
10. ما هو محدد CSS الذي يعثر على جميع عناصر `<div>` الموجودة داخل عنصر `<div>` آخر؟
11. ما هو محدد CSS الذي يعثر على عنصر `<button>` له الخاصية `value` وقيمتها `favorite`؟
12. فلنقل أن لدينا كائن `Tag` في `bs4` مخزن في المتغير `spam`، علمًا أنه يمثل العنصر الآتي `<div>Hello world</div>`، هنا كيف نحصل على السلسلة النصية `'Hello world'` من الكائن `Tag`؟
13. كيف نخزن كل خاصيات الكائن `Tag` في `bs4` في متغير باسم `linkElem`؟
14. لن تعمل العبارة `import selenium`. كيف نستورد الوحدة `selenium` استيرادًا صحيحًا؟
15. ما الفرق بين توابع `find_element_*` و `find_elements_*`؟
16. ما هي توابع الكائنات `WebElement` التي تحاكي ضغطات الفأرة ولوحة المفاتيح؟
17. ماذا يحدث حين استدعاء `send_keys(Keys.ENTER)` على زر الإرسال لكائن `WebElement`؟ وما أسهل طريقة لإرسال الاستمارة؟
18. كيف يمكننا محاكاة أزرار العودة إلى الخلف والانتقال إلى الأمام وتحديث الصفحة في `selenium`؟

12.9 مشاريع للتدريب

لكي تتدرب، اكتب برامج لتنفيذ المهام الآتية.

12.9.1 إرسال البريد الإلكتروني من سطر الأوامر

اكتب برنامج يقبل عنوان بريد إلكتروني وسلسلة نصية في سطر الأوامر، ثم يستخدم `selenium` للدخول إلى حسابك البريدي وإرسال بريد إلكتروني إلى العنوان المحدد (أنصحك بإنشاء حساب تجريبي مختلف عن حسابك الأساسي). اكتب برامج أخرى لإضافة منشورات إلى فيسبوك أو تويتر (إكس، سمه ما شئت!).

12.9.2 منزِّل الصور

اكتب برنامج يفتح أحد مواقع مشاركة الصور مثل Flickr ويبحث عن تصنيف معين من الصور وينزل جميع نتائج البحث في الصفحة الأولى. يمكنك كتابة برنامج يعمل على أي موقع مشاركة صور ولديه خاصية البحث.

12.9.3 لعبة 2048

لعبة 2048 هي لعبة بسيطة تسمح لك بجمع مربعات بجعلها تنزلق إلى أحد الاتجاهات باستخدام أزرار الأسهم. يمكنك أن تحصل على نتيجة عالية إذا جعلت الانزلاق بهذا الترتيب مرارًا وتكرارًا: الأعلى ثم اليمين ثم الأسفل ثم اليسار. اكتب برنامج بسيط يفتح اللعبة gabrielecirulli.github.io وينفذ النمط السابق للعب اللعبة تلقائيًا.

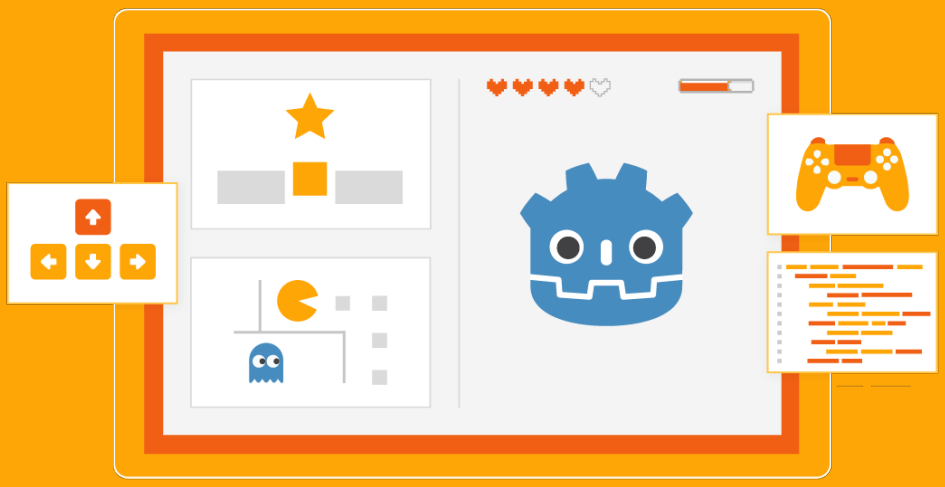
12.9.4 التحقق من الروابط

اكتب برنامج يأخذ رابط URL من صفحة ويب، ويحاول أن ينزل كل صفحة مرتبطة فيها، ويُعلِّم كل الصفحات التي تعيد 404 وتؤشر عليها أنها روابط مكسورة.

12.10 الخلاصة

المهام المملة ليست متعلقة بالملفات المحلية على حاسوبك. لذا إن كنت قادرًا على تنزيل صفحات الويب برمجيًا فستتمكن من أتمتة أي مهمة تردك! تسهل الوحدة `requests` عملية تنزيل صفحات الويب. وبعد أن تتعلم أساسيات HTML سيكون بإمكانك استخدام الوحدة `BeautifulSoup` لتفسير الصفحات التي تنزلها. لكن إن أردت أتمتة أي مهمة متعلقة بالويب أيًا كانت، فيمكن التحكم برمجيًا مباشرةً بمتصفح الويب عبر الوحدة `selenium` التي تسمح بفتح المواقع وملء الاستمارات كما لو كان برنامجك كائنًا بشريًا يتصفح المواقع.

دورة تطوير الألعاب



مميزات الدورة

- بناء معرض أعمال قوي بمشاريع حقيقية
- وصول مدى الحياة لمحتويات الدورة
- تحديثات مستمرة على الدورة مجانًا
- شهادة معتمدة من أكاديمية حسوب
- إرشادات من المدربين على مدار الساعة
- من الصفر دون الحاجة لخبرة مسبقة

اشترك الآن

13. العمل بجداول إكسل باستخدام بايثون

قد لا نفكر في جداول البيانات بوصفها أدوات برمجية، ولكن يستخدمها الجميع تقريبًا لتنظيم المعلومات في هياكل بيانية ثنائية الأبعاد، وإجراء العمليات الحسابية باستخدام الصيغ، وعرض المخرجات على شكل مخططات، لذا سندمج لغة بايثون مع تطبيقين شائعين خاصين بجداول البيانات وهما: مايكروسوفت إكسل Microsoft Excel وجداول بيانات جوجل Google Sheets.

يُعَد إكسل تطبيق جداول بيانات قوي وشائع الاستخدام لنظام تشغيل ويندوز، وتوجد وحدةٌ برمجية هي وحدة `openpyxl` التي تسمح لبرامجك المكتوبة بلغة بايثون بقراءة وتعديل ملفات جداول بيانات إكسل. يمكن أن تكون لديك مهمة مملة متمثلة في نسخ بيانات معينة من جدول بيانات ولصقها في جدول بيانات آخر مثلًا، أو قد يتعين عليك أن تمر على آلاف الصفوف واختيار عددٍ قليل منها لإجراء تعديلات بسيطة بناءً على بعض المعايير، أو قد يتعين عليك الاطلاع على مئات جداول البيانات الخاصة بميزانيات الأقسام باحثًا عن الخلايا الملونة باللون الأحمر، وهذه هي مهام جداول البيانات البسيطة والمملة التي يمكن لبايثون تطبيقها نيابةً عنك.

يُعَد برنامج إكسل برنامجًا خاصًا بشركة مايكروسوفت، ولكن توجد بدائل مجانية تعمل على أنظمة تشغيل ويندوز وماك macOS ولينكس Linux، حيث يعمل كل من ليبرأوفيس كالك LibreOffice Calc وأوبن أوفيس كالك OpenOffice Calc بنجاح مع صيغة ملفات جداول البيانات `.xlsx` الخاصة بإكسل، مما يعني أن الوحدة `openpyxl` يمكن أن تعمل مع جداول البيانات الخاصة بهذين التطبيقين أيضًا، ويمكنك تنزيلهما من موقعهما الرسمي مباشرةً. قد تجد أن هذين البرنامجين أسهل في الاستخدام من إكسل بالرغم من أن برنامج إكسل قد يكون مثبت مسبقًا على حاسوبك الشخصي، ولكن لقطات الشاشة الموجودة في هذا الفصل جميعها مأخوذة من إكسل 2010 على نظام تشغيل ويندوز 10.

13.1 مستندات إكسل

لنتعـرّف أولًا على بعض التعريفـات الأساسـية، إذ يسـمى مسـتند جـدول بيانـات إكسـل بالمصـنف Workbook، ويُحفَـظ المصـنف في ملـف امتـداده `.xlsx`، ويمكن أن يحتـوي المصـنف على أوراق Sheets متعـددة (وتسـمَّى أوراق عمـل Worksheets أيضًـا)، كمـا تُسـمَّى الورقـة الـتي يعرضـها المسـتخدم حاليًـا -أو المعروضة آخر مرة قبل إغلاق إكسل- بالورقة النشطة Active Sheet.

تحتوي كل ورقة على أعمدة Columns تتعامل معهـا باسـتخدام حـروف تبـدأ بـالحرف A، وصـفوف Rows تتعامل معها باستخدام أعـداد تبـدأ بالعـدد 1. يسـمى المربـع الموجـود في عمـود أو صـف معين بالخليـة Cell، ويمكن أن تحتوي كل خلية على قيمة عددية أو نصية، وتتشكّل الورقة من شبكةٍ من الخلايا التي تحوي البيانات.

13.2 تثبيت وحدة openpyxl

لا تحتوي لغة بايثون مسبقًا على وحدة `OpenPyXL`، إذ يجب عليك تثبيتها أولًا، لذا اتبع الإرشادات الخاصـة بتثبيت الوحدات الخارجية التي سنوضحها في الملحق 1.

سنستخدم في هذا الفصل الإصدار 2.6.2 من الوحدة `OpenPyXL`، لذا من المهم أن تثبّت هذا الإصـدار من خلال تشغيل الأمر`pip install --user -U openpyxl==2.6.2`، لأن الإصدارات الأحدث منها غير متوافقة مع المعلومات الموجودة في هذا الفصل. أدخِل الأمر التالي في الصدفة التفاعليـة Interactive Shell لاختبار ما إذا كان كانت وحدة `OpenPyXL` مُثبَّتة بصورة صحيحة:

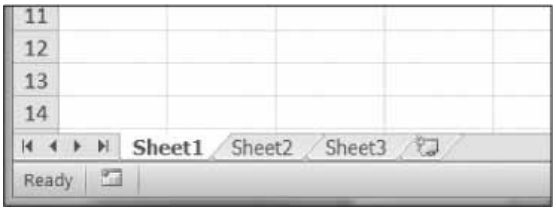
```
>>> import openpyxl
```

إذا جرى تثبيت الوحدة بصورة صحيحة، فلن ينتج عن الأمر السـابق أيّ رسـائل خطـأ. تـذكّر اسـتيراد الوحـدة `openpyxl` قبـل تشـغيل أمثلـة أوامـر الصـدفة التفاعليـة في هـذا الفصـل، وإلّا فستحصـل على الخطـأ `NameError: name 'openpyxl' is not defined`.

> **ملاحظة**: يمكنك العثور على توثيق وحدة OpenPyXL الكامل على موقعها الرسمي.

13.3 قراءة مستندات إكسل

سنستخدم في الأمثلة الواردة في هذا الفصـل جـدول بيانـات اسـمه `example.xlsx` مخـزن في المجلـد الجذر، حيث يمكنك إما إنشاء جدول البيانات بنفسك أو تنزيله. يوضّح الشكل التالي تبويبات للأوراق الافتراضية الثلاثة التي اسـمها Sheet1 و Sheet2 و Sheet3 الـتي يوفرهـا إكسـل تلقائيًـا للمصـنفات الجديـدة، ولكن قـد يختلف عدد هذه الأوراق الافتراضية بحسب نظام التشغيل وبرنامج جداول البيانات:



الشكل 55: أشهر القيم المستخدمة في الوحدة

selenium.webdriver.common.keys

توجد التبويبات الخاصة بأوراق المصنف في الزاوية السفلية اليسرى من إكسل

يجب أن تبدو الورقة 1 في مثالنا مثل الجدول الآتي، ولكن إن لم تنزّل الملف `example.xlsx` من الموقع، فيجب أن تدخِل هذه البيانات في الورقة بنفسك:

C	B	A	
73	Apples	PM 1:34:02 4/5/2015	1
85	Cherries	AM 3:41:23 4/5/2015	2
14	Pears	PM 12:46:51 4/6/2015	3
52	Oranges	AM 8:59:43 4/8/2015	4
152	Apples	AM 2:07:00 4/10/2015	5
23	Bananas	PM 6:10:37 4/10/2015	6
98	Strawberries	AM 2:40:46 4/10/2015	7

الجدول 17:البيانات لازمة الإدخال في الورقة 1

أصبح جدول البيانات جاهزًا الآن، إذًا لنتعرّف على كيفية التعامل معه باستخدام وحدة openpyxl.

13.3.1 فتح مستندات إكسل باستخدام وحدة OpenPyXL

يمكنك استخدام الدالة `openpyxl.load_workbook()` بعد استيراد وحدة `openpyxl` مباشرةً، لذا أدخِل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> type(wb)
<class 'openpyxl.workbook.workbook.Workbook'>
```

تأخذ الدالة `openpyxl.load_workbook()` اسم الملف وتعيد قيمة نوع بيانات المصنف `workbook`، حيث يمثل الكائن `Workbook` ملف إكسل، ويشبه ذلك كيفية تمثيل الكائن `File` ملفًا نصيًا مفتوحًا.

تذكّر أن الملف `example.xlsx` يجب أن يكون موجودًا في مجلد العمل الحالي لتتمكّن من التعامل معه، إذ يمكنك معرفة مجلد العمل الحالي من خلال استيراد وحدة `os` واستخدام الدالة `os.getcwd()`، ويمكنك تغيير مجلد العمل الحالي باستخدام الدالة `os.chdir()`.

13.3.2 الحصول على الأوراق من المصنف

يمكنك الحصول على قائمة بجميع أسماء الأوراق في المصنف من خلال الوصول إلى السمة Attribute التي هي `sheetnames`، لذا أدخِل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> wb.sheetnames # أسماء الأوراق الخاصة بالمصنف
['Sheet1', 'Sheet2', 'Sheet3']
>>> sheet = wb['Sheet3'] # الحصول على ورقة من المصنف
>>> sheet
<Worksheet "Sheet3">
>>> type(sheet)
<class 'openpyxl.worksheet.worksheet.Worksheet'>
>>> sheet.title # الحصول على عنوان الورقة بوصفه سلسلة نصية
'Sheet3'
>>> anotherSheet = wb.active # الحصول على الورقة النشطة
>>> anotherSheet
<Worksheet "Sheet1">
```

يمثّل الكائنُ `Worksheet` الورقة التي يمكنك الحصول عليها باستخدام الأقواس المربعة مع السلسلة النصية التي تمثّل اسم هذه الورقة، حيث يشبه ذلك استخدام مفتاح القاموس، ويمكنك استخدام السمة `active` للكائن `Workbook` للحصول على ورقة المصنف النشطة، فالورقة النشطة هي الورقة الموجودة في المقدمة عند فتح المصنف في إكسل. كما يمكنك الحصول على اسم الكائن `Worksheet` من سمة العنوان `title` بعد الحصول على هذا الكائن.

13.3.3 الحصول على الخلايا من الأوراق

يمكنك الوصول إلى الكائن `Cell` باستخدام اسمه بعد الحصول على الكائن `Worksheet`، لذا أدخِل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb['Sheet1'] # الحصول على ورقة من المصنف
>>> sheet['A1'] # الحصول على خلية من الورقة
<Cell 'Sheet1'.A1>
>>> sheet['A1'].value # الحصول على القيمة من الخلية
datetime.datetime(2015, 4, 5, 13, 34, 2)
>>> c = sheet['B1'] # الحصول على خلية أخرى من الورقة
>>> c.value
'Apples'
>>> # الحصول على الصف والعمود والقيمة من الخلية
>>> 'Row %s, Column %s is %s' % (c.row, c.column, c.value)
'Row 1, Column B is Apples'
>>> 'Cell %s is %s' % (c.coordinate, c.value)
'Cell B1 is Apples'
>>> sheet['C1'].value
73
```

يحتوي كائن الخلية `Cell` على سمة القيمة `value` التي تحتوي على القيمة المُخزَّنة في هذه الخلية، وتحتوي الكائنات `Cell` أيضًا على سمات الصف `row` والعمود `column` والإحداثيات `coordinate` التي توفّر معلومات موقع الخلية، فمثلًا يعطينا الوصول إلى السمة `value` للكائن `Cell` الخاص بالخلية B1 السلسلة النصية `'Apples'`، وتعطينا السمة `row` العدد الصحيح 1، وتعطينا السمة `column` العمود `'B'`، وتعطي السمة `coordinate` القيمة `'B1'`.

تفسّر الوحدة `OpenPyXL` تلقائيًا التواريخ الموجودة في العمود A وتعيدها بوصفها قيم `datetime` بدلًا من إعادتها كسلاسل نصية، حيث سنوضّح لاحقًا نوع البيانات `datetime`.

يمكن أن يكون تحديد عمود باستخدام حرف أمرًا صعبًا برمجيًا، وخاصةً لأن الأعمدة تبدأ باستخدام حرفين مثل AA و AB و AC بعد العمود Z، لذا يمكنك بدلًا من ذلك الحصول على خلية باستخدام التابع `cell()` الخاص بالورقة وتمرير أعداد صحيحة لوسطاء الكلمات المفتاحية Keyword Arguments التي هي `row` و`column` الخاصة بهذا التابع، ويكون العدد الصحيح للصف أو العمود الأول هو 1 وليس 0.

لندخِل ما يلي في الصدفة التفاعلية:

```
sheet.cell(row=1, column=2)
<Cell 'Sheet1'.B1>
>>> sheet.cell(row=1, column=2).value
'Apples'
>>> for i in range(1, 8, 2): # المرور على جميع الصفوف الأخرى
...     print(i, sheet.cell(row=i, column=2).value)
...
1 Apples
3 Pears
5 Apples
7 Strawberries
```

لاحظ أن استخدام التابع `cell()` الخاص بالورقة وتمرير `row=1` و `column=2` له يعطي كائن `Cell` للخلية B1 كما فعل استخدام `sheet['B1']` تمامًا. يمكنك بعد ذلك كتابة حلقة `for` لطباعة قيم سلسلة من الخلايا باستخدام التابع `cell()` ووسطاء الكلمات المفتاحية الخاصة به.

لنفترض أنك تريد الانتقال إلى العمود B وطباعة القيمة الموجودة في جميع الخلايا التي يكون رقم صفها عددًا فرديًا، حيث يمكنك الحصول على الخلايا لجميع الصفوف لها أرقام فردية من خلال تمرير القيمة 2 لمعامل "الخطوة step" الخاص بالدالة `range()`. يُمرَّر المتغير `i` الخاص بالحلقة `for` إلى وسيط الكلمة المفتاحية `row` الخاص بالتابع `cell()`، بينما تُمرّر القيمة 2 دائمًا إلى وسيط الكلمة المفتاحية `column` في هذه الحالة. لاحظ أننا مرّرنا العدد الصحيح 2 ولم نمرّر السلسلة النصية `'B'`.

يمكنك تحديد حجم الورقة باستخدام السمتين `max_row` و `max_column` للكائن `Worksheet` كما يلي:

```
import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb['Sheet1']
>>> sheet.max_row # الحصول على عدد الصفوف الأكبر
7
>>> sheet.max_column # الحصول على عدد الأعمدة الأكبر
3
```

لاحظ أن السمة `max_column` تمثل عددًا صحيحًا ولا تمثّل الحرف الذي يظهر في إكسل.

13.3.4 تحويل حروف الأعمدة إلى أعداد

يمكنك تحويل أسماء الأعمدة من حروف إلى أعداد من خلال استدعاء الدالة التالية:

```
openpyxl.utils.column_index_from_string()
```

ويمكنك التحويل من أعداد إلى حروف من خلال استدعاء الدالة الآتية:

```
openpyxl.utils.get_column_letter()
```

إذًا لندخِل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> from openpyxl.utils import get_column_letter,
column_index_from_string
>>> get_column_letter(1) # ترجمة العمود 1 إلى حرف
'A'
>>> get_column_letter(2)
'B'
>>> get_column_letter(27)
'AA'
>>> get_column_letter(900)
'AHP'
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb['Sheet1']
>>> get_column_letter(sheet.max_column)
'C'
>>> column_index_from_string('A') # الحصول على العدد المقابل للحرف A
1
>>> column_index_from_string('AA')
27
```

يمكنك استدعاء الدالة `get_column_letter()` وتمرير عدد صحيح لها مثل العدد 27 لمعرفة اسم الحرف في العمود السابع والعشرين بعد استيراد الدالتين السابقتين من الوحدة `openpyxl.utils`. بينما تطبّق الدالة `column_index_string()` العكس، حيث تمرّر لها الحرف الذي يمثل اسم العمود، وتعطيك رقم هذا العمود. لا تحتاج إلى تحميل مصنف لاستخدام هذه الدوال، ولكن إن أردت يمكنك تحميل مصنف، ثم

الحصول على الكائن `Worksheet` واستخدام سمته `max_column` مثلًا للحصول على عدد صحيح، ثم يمكنك تمرير هذا العدد الصحيح إلى الدالة `get_column_letter()`.

13.3.5 الحصول على الصفوف والأعمدة من الأوراق

يمكنك تقسيم الكائنات `Worksheet` للحصول على كافة الكائنات `Cell` الموجودة في صف أو عمود أو منطقة مستطيلة من جدول البيانات، ثم يمكنك التكرار على جميع الخلايا الموجودة في كل قسم. أدخِل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb['Sheet1']
>>> tuple(sheet['A1':'C3']) # الحصول على كل الخلايا من A1 إلى C3
((<Cell 'Sheet1'.A1>, <Cell 'Sheet1'.B1>, <Cell 'Sheet1'.C1>), (<Cell
'Sheet1'.A2>, <Cell 'Sheet1'.B2>, <Cell 'Sheet1'.C2>), (<Cell
'Sheet1'.A3>,
<Cell 'Sheet1'.B3>, <Cell 'Sheet1'.C3>))
❶ >>> for rowOfCellObjects in sheet['A1':'C3']:
❷ ...     for cellObj in rowOfCellObjects:
...         print(cellObj.coordinate, cellObj.value)
...     print('-- نهاية الصف --')

A1 2015-04-05 13:34:02
B1 Apples
C1 73
-- نهاية الصف --
A2 2015-04-05 03:41:23
B2 Cherries
C2 85
```

```
-- نهاية الصف --
A3 2015-04-06 12:46:51
B3 Pears
C3 14
-- نهاية الصف --
```

حدّدنا في المثال السابق أننا نريد كائنات `Cell` في المنطقة المستطيلة من الخلية A1 إلى الخلية C3، وحصلنا على كائن `Generator` الذي يحتوي على كائنات `Cell` في تلك المنطقة، حيث يمكننا تصوّر الكائن `Generator` من خلال استخدام الدالة `tuple()` معه لعرض كائنات `Cell` الخاصة به ضمن مجموعة Tuple.

تحتوي هذه المجموعة على ثلاث مجموعات أخرى، مجموعة لكل صف من أعلى المنطقة المطلوبة إلى أسفلها، حيث تحتوي كل مجموعة من هذه المجموعات الداخلية الثلاث على كائنات `Cell` في صف واحد من المنطقة المطلوبة من الخلية الموجودة في أقصى اليسار إلى اليمين.

يحتوي قسم الورقة الذي حدّدناه على جميع كائنات `Cell` في المنطقة المؤلفة من الخلية A1 إلى الخلية C3، بدءًا من الخلية العلوية اليسرى وانتهاءً بالخلية السفلية اليمنى. طبعنا قيم كل خلية في هذه المنطقة باستخدام حلقتي `for`، حيث تتكرر حلقة `for` الخارجية على كل صف في القسم ❶، ثم تتكرر حلقة `for` المتداخلة لكل صف على كل خلية في هذا الصف ❷.

يمكن أيضًا الوصول إلى قيم الخلايا في صف أو عمود معين من خلال استخدام سمة `rows` و `columns` الخاصة بكائن `Worksheet`، ولكن يجب تحويل هذه السمات إلى قوائم باستخدام الدالة `list()` قبل أن تتمكّن من استخدام الأقواس المربعة والفهرس معها.

إذًا لندخِل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.active
>>> list(sheet.columns)[1] # الحصول على خلايا العمود الثاني
(<Cell 'Sheet1'.B1>, <Cell 'Sheet1'.B2>, <Cell 'Sheet1'.B3>, <Cell
'Sheet1'.
B4>, <Cell 'Sheet1'.B5>, <Cell 'Sheet1'.B6>, <Cell 'Sheet1'.B7>)
>>> for cellObj in list(sheet.columns)[1]:
    print(cellObj.value)
```

```
Apples
Cherries
Pears
Oranges
Apples
Bananas
Strawberries
```

سيؤدي استخدام السمة `rows` مع الكائن `Worksheet` إلى إعطاء مجموعة من عدة مجموعات، وتمثل كل مجموعة من هذه المجموعات الداخلية صفًا، وتحتوي على كائنات `Cell` الموجودة في هذا الصف. تعطي السمة `columns` أيضًا مجموعة من عدة مجموعات، حيث تحتوي كل مجموعة من المجموعات الداخلية على كائنات `Cell` في عمود معين.

لدينا في مثالنا الملف `example.xlsx` الذي يحتوي على 7 صفوف و3 أعمدة، وبالتالي تعطي السمة `rows` مجموعة مؤلفةً من 7 مجموعات، حيث تحتوي كل منها على 3 كائنات `Cell`، وتعطي السمة `columns` مجموعة مكونة من 3 مجموعات، حيث تحتوي كل منها على 7 كائنات `Cell`.

يمكن الوصول إلى مجموعة معينة من خلال الإشارة إليها عبر استخدام فهرسها في المجموعة الكبرى، فقد نحصل على المجموعة التي تمثل العمود B من خلال استخدام `list(sheet.columns)[1]`، كما قد نحصل على المجموعة التي تحتوي على كائنات `Cell` في العمود A من خلال استخدام `list(sheet.columns)[0]`. يمكنك بعد أن تكون لديك مجموعة تمثل صفًا أو عمودًا واحدًا التكرار على كائنات `Cell` الخاصة بها وطباعة قيمها.

13.3.6 المصنفات والأوراق والخلايا

إليك ملخص بجميع الدوال والتوابع وأنواع البيانات المستخدمة في قراءة خلية من ملف جدول بيانات:

1. استيراد وحدة `openpyxl`.
2. استدعاء الدالة `openpyxl.load_workbook()`.
3. الحصول على كائن `Workbook`.
4. استخدام السمة `active` أو السمة `sheetnames`.
5. الحصول على كائن `Worksheet`.
6. استخدم طريقة الفهرسة أو تابع `cell()` الخاص بالورقة مع وسطاء الكلمات المفتاحية `row` و`column`.

7. الحصول على كائن `Cell`.
8. قراءة السمة `value` الخاصة بالكائن `Cell`.

13.4 تطبيق عملي: قراءة البيانات من جدول بيانات

لنفترض أن لديك جدول بيانات يمثّل الإحصاء السكاني للولايات المتحدة الأمريكية لعام 2010، ولديك مهمة مملة تتمثل في استعراض آلاف الصفوف لحساب إجمالي عدد السكان وعدد المناطق الإحصائية Census Tracts لكل مقاطعة County، فالمنطقة الإحصائية هي ببساطة منطقة جغرافية محددة لأغراض الإحصاء السكاني، ويمثل كل صف في جدول البيانات منطقةً إحصائية واحدة. سنسمّي ملف جدول البيانات `censuspopdata.xlsx` الذي يمكنك تنزيله، وتبدو محتوياته كما يلي:

	A	B	C	D	E
1	CensusTract	State	County	POP2010	
9841	06075010500	CA	San Francisco	2685	
9842	06075010600	CA	San Francisco	3894	
9843	06075010700	CA	San Francisco	5592	
9844	06075010800	CA	San Francisco	4578	
9845	06075010900	CA	San Francisco	4320	
9846	06075011000	CA	San Francisco	4827	
9847	06075011100	CA	San Francisco	5164	

Population by Census Tract

Ready

الشكل 56: جدول بيانات censuspopdata.xlsx

يستطيع إكسل حساب مجموع خلايا محددة متعددة، ولكن يجب عليك أيضًا تحديد الخلايا التي تمثل المقاطعات التي يزيد عدد سكانها عن 3000 نسمة. كما قد يستغرق حساب عدد سكان المقاطعة يدويًا بضع ثوانٍ فقط، ولكنه قد يستغرق ساعات لجدول البيانات بأكمله.

ستكتب في هذا التطبيق العملي سكربتًا يمكنه القراءة من ملف جدول بيانات الإحصاء السكاني وحساب إحصائيات كل مقاطعة في غضون ثوانٍ، إذ سيفعل برنامجك ما يلي:

1. يقرأ البيانات من جدول بيانات إكسل.
2. يحسب عدد المناطق الإحصائية في كل مقاطعة.
3. يحسب إجمالي عدد السكان في كل مقاطعة.
4. يطبع النتائج.

وهذا يعني أن شيفرتك البرمجية ستحتاج ما يلي:

1. فتح وقراءة خلايا مستند إكسل باستخدام وحدة `openpyxl`.

2. حساب جميع بيانات المناطق الإحصائية وعدد السكان وتخزينها في هيكل بيانات.
3. كتابة هيكل البيانات في ملف نصي له الامتداد `.py` باستخدام الوحدة `pprint`.

13.4.1 الخطوة الأولى: قراءة بيانات جدول البيانات

توجد ورقة واحدة فقط في جدول البيانات `censuspopdata.xlsx`، تسمى "عدد السكان حسب المنطقة الإحصائية" `'Population by Census Tract'`، ويحتوي كل صف على بيانات منطقة إحصائية واحدة، والأعمدة هي رقم المنطقة (A) واختصار الولاية (B) واسم المقاطعة (C) وعدد سكان المنطقة (D).

افتح تبويبًا جديدًا لإنشاء ملف جديد في محرّرك وأدخِل الشيفرة البرمجية التالية، واحفظ الملف بالاسم `readCensusExcel.py`:

```
#! python3
 # readCensusExcel.py - جدول عدد السكان وعدد المناطق الإحصائية لكل مقاطعة

❶ import openpyxl, pprint

 print('Opening workbook...')

❷ wb = openpyxl.load_workbook('censuspopdata.xlsx')

❸ sheet = wb['Population by Census Tract']

 countyData = {}


 # املأ بيانات المقاطعة countyData بعدد سكان كل مقاطعة ومناطقها الإحصائية

 print('Reading rows...')

❹ for row in range(2, sheet.max_row + 1):

    # يحتوي كل صف في جدول البيانات على بياناتٍ لمنطقة إحصائية واحدة

    state  = sheet['B' + str(row)].value

    county = sheet['C' + str(row)].value

    pop    = sheet['D' + str(row)].value

# افتح ملف نصي جديدواكتب محتويات بيانات المقاطعة كاونتيداتا فيه
```

تستورد الشيفرة البرمجية السابقة الوحدة `openpyxl` والوحدة `pprint` التي ستستخدمها لطباعة بيانات المقاطعة النهائية ❶، ثم تفتح الملف `censuspopdata.xlsx` ❷، وتحصل على الورقة التي تحتوي على البيانات الإحصائية ❸، وتبدأ بالتكرار على صفوف هذه الورقة ❹.

لاحظ أنك أنشأتَ أيضًا متغيرًا بالاسم `countyData`، والذي سيحتوي على عدد السكان وعدد المناطق التي تحسبها لكل مقاطعة، ولكن يجب عليك تحديد كيفية هيكلة البيانات بداخله قبل أن تتمكن من تخزين أيّ شيء فيه.

13.4.2 الخطوة الثانية: ملء هيكل البيانات

يُعَد هيكل البيانات المُخزَّن في المتغير `countyData` قاموسًا تكون اختصارات أسماء الولايات مفاتيحًا له، حيث سيُربَط اختصار كل ولاية مع قاموس آخر مفاتيحه هي سلاسل نصية تمثّل أسماء المقاطعات في تلك الولاية، وسيُربَط كل اسم مقاطعة بدوره مع قاموسٍ آخر يحتوي على مفتاحين فقط هما `'tracts'` و `'pop'`، ويُربَط هذان المفتاحان مع عدد المناطق الإحصائية وعدد السكان في المقاطعة، فمثلًا سيبدو القاموس مشابهًا لما يلي:

```
{'AK': {'Aleutians East': {'pop': 3141, 'tracts': 1},
    'Aleutians West': {'pop': 5561, 'tracts': 2},
    'Anchorage': {'pop': 291826, 'tracts': 55},
    'Bethel': {'pop': 17013, 'tracts': 3},
    'Bristol Bay': {'pop': 997, 'tracts': 1},
    --snip--
```

إذا خُزِّن القاموس السابق في المتغير `countyData`، فيمكن تقييم التعابير التالية كما يلي:

```
>>> countyData['AK']['Anchorage']['pop']
291826
>>> countyData['AK']['Anchorage']['tracts']
55
```

وستكون مفاتيح قاموس `countyData` كما يلي:

```
countyData[state abbrev][county]['tracts']
countyData[state abbrev][county]['pop']
```

عرفتَ كيفيـة تنظيم هيكـل بيانـات countyData، ويمكنـك الآن كتابـة الشـيفرة البرمجيـة الـتي سـتملؤه ببيانات المقاطعة، لذا أضِف الشيفرة البرمجية التالية إلى نهاية برنامجك:

```
#! python 3
# readCensusExcel.py - جدول عدد السكان وعدد المناطق الإحصائية لكل مقاطعة

--snip--

for row in range(2, sheet.max_row + 1):
    # يحتوي كل صف في جدول البيانات على بيانات لمنطقة إحصائية واحدة
    state  = sheet['B' + str(row)].value
    county = sheet['C' + str(row)].value
    pop    = sheet['D' + str(row)].value

    # تأكد من وجود مفتاح هذه الولاية state
❶   countyData.setdefault(state, {})

    # تأكد من وجود مفتاح هذه المقاطعة county في تلك الولاية
❷   countyData[state].setdefault(county, {'tracts': 0, 'pop': 0})

    # يمثل كل صف منطقة إحصائية واحدة، لذا يجب زيادة عدد المناطق بمقدار واحد
❸   countyData[state][county]['tracts'] += 1

    # زيادة عدد سكان pop المقاطعة بمقدار عدد السكان في هذه المنطقة الإحصائية
❹   countyData[state][county]['pop'] += int(pop)

# افتح ملف نصي جديد واكتب محتويات بيانات المقاطعة كاونتيداتا فيه
```

يجري السـطران الأخـيران من الشـيفرة البرمجيـة السـابقة العمليـات الحسـابية الفعليـة، حيث تزيـد قيمـة المناطق الإحصائية tracts ❸ وقيمة عدد السكان pop ❹ للمقاطعة الحالية في كل تكرار لحلقة for. بينما سببُ وجود الشيفرة البرمجية المتبقية هو أنه لا يمكنك إضافة قاموس المقاطعة بوصفه قيمةً لمفتـاح اختصـار الولايــة إلّا عنــد وجــود المفتــاح نفســه في countyData، إذ ستتســبّب التعليمــة countyData['AK']['Anchorage']['tracts'] += 1 في حدوث خطأ إن لم يكن المفتاح 'AK' موجودًا بعد. يمكنك التأكد

من وجود مفتاح اختصار الولاية في هيكل بياناتك من خلال استدعاء التابع `setdefault()` لضبط قيمة الولاية `state` ❶ إن لم تكن موجودة مسبقًا.

يحتاج قاموس `countyData` إلى قاموس آخر بوصفه قيمةً لكل مفتاح يمثّل اختصار الولاية، وبالتالي سيحتاج كلٌّ من هذه القواميس إلى قاموس خاص به بوصفه قيمة لكل مفتاح مقاطعة ❷، وسيحتاج كل من هذه القواميس بدوره إلى مفاتيح `'tracts'` و `'pop'` التي تبدأ بالقيمة الصحيحة 0. إذا شعرت بالضياع عند تتبّع بنية القاموس، فارجع إلى مثال القاموس في بداية هذه الفقرة.

لن يفعل التابع `setdefault()` شيئًا إذا كان المفتاح موجودًا مسبقًا، وبالتالي يمكنك استدعاؤه في كل تكرار للحلقة `for` بدون مشاكل.

13.4.3 الخطوة الثالثة: كتابة النتائج في ملف

سيحتوي قاموس `countyData` بعد انتهاء حلقة `for` على جميع معلومات عدد السكان والمناطق المرتبطة بمفتاح المقاطعة county والولاية state، ويمكنك عندها برمجة مزيدٍ من الشيفرة البرمجية لكتابة هذه المعلومات في ملف نصي أو جدول بيانات إكسل آخر. لنستخدم الآن الدالة `pprint.pformat()` لكتابة قيمة قاموس `countyData` بوصفها سلسلة نصية ضخمة في ملف اسمه `census2010.py`، لذا أضِف الشيفرة البرمجية التالية إلى نهاية برنامجك، وتأكد من إبقائه بدون مسافة بادئة بحيث يبقى خارج حلقة `for`:

```
#! python 3
# readCensusExcel.py - جدول عدد السكان وعدد المناطق الإحصائية لكل مقاطعة

--snip--
for row in range(2, sheet.max_row + 1):
--snip--

# افتح ملف نصي جديد واكتب محتويات بيانات المقاطعة كاونتيداتا فيه
print('Writing results...')
resultFile = open('census2010.py', 'w')
resultFile.write('allData = ' + pprint.pformat(countyData))
resultFile.close()
print('Done.')
```

ينتج عن الدالة `pprint.pformat()` سلسلةٌ نصية مُنسَّقة كشيفرة بايثون صالحة، والتي يمكنك إخراجها إلى ملف نصي اسمه `census2010.py`، وبالتالي سيتولد برنامج بايثون من برنامج بايثون الخاص بك. قد يبدو ذلك معقدًا، ولكن تتمثّل الفائدة في أنه يمكنك استيراد الملف `census2010.py` مثل أي وحدة بايثون أخرى. غيّر مجلد العمل الحالي إلى المجلد الذي يحتوي على الملف `census2010.py` ثم استورده في الصدفة التفاعلية كما يلي:

```
>>> import os
>>> import census2010
>>> census2010.allData['AK']['Anchorage']
{'pop': 291826, 'tracts': 55}
>>> anchoragePop = census2010.allData['AK']['Anchorage']['pop']
>>> print('The 2010 population of Anchorage was ' + str(anchoragePop))
The 2010 population of Anchorage was 291826
```

يُعَد برنامج `readCensusExcel.py` عديم الجدوى، فلا حاجة لتشغيله مرة أخرى بعد حفظ نتائجه في الملف `census2010.py`، ويمكنك تشغيل الأمر `import census2010` عندما تحتاج إلى بيانات المقاطعة.

سيستغرق حساب هذه البيانات يدويًا ساعات يدويًا، ولكن أنجز هذا البرنامج هذا الأمر في بضع ثوان، ولن تواجه أيّ مشكلة في استخراج المعلومات المحفوظة في جدول بيانات إكسل وإجراء العمليات الحسابية عليها باستخدام وحدة `OpenPyXL`.

> **ملاحظة**: لا تنسَ أنه يمكنك تنزيل البرنامج الكامل.

13.4.4 أفكار لبرامج مماثلة

تستخدم العديد من الشركات والمكاتب برنامج إكسل لتخزين أنواع مختلفة من البيانات، وتصبح جداول البيانات كبيرة الحجم وغير عملية بسهولة.

تمتلك البرامج التي تحلّل جدول بيانات إكسل بنية مماثلة، فهي تحمّل ملف جدول البيانات، وتجهّز بعض المتغيرات أو هياكل البيانات، ثم تتكرر على كل صف من الصفوف في جدول البيانات، حيث يمكن أن تفعل مثل هذه البرامج ما يلي:

- مقارنة البيانات في صفوف متعددة في جدول بيانات.
- فتح ملفات إكسل متعددة ومقارنة البيانات بين جداول بيانات.
- التحقق من احتواء جدول البيانات على صفوف فارغة أو بيانات غير صالحة في أيّ خلايا وتنبيه المستخدم في حالة وجود ذلك.
- قراءة البيانات من جدول البيانات واستخدامها كدخلٍ لبرامج بايثون الخاصة بك.

13.5 الكتابة في مستندات إكسل

سنتعرّف الآن على كيفية كتابة مستندات إكسل باستخدام لغة بايثون، إذ تـوفّر وحـدة OpenPyXL الخاصـة بلغة بايثون طرقًا لكتابة البيانات، مما يعني أن برامجك يمكنهـا إنشـاء ملفـات جـداول البيانـات وتعـديلها، ومن السهل إنشاء جداول بيانات تحتوي على آلاف الصفوف من البيانات باستخدام بايثون.

13.5.1 إنشاء وحفظ مستندات إكسل

استدعِ الدالة `openpyxl.Workbook()` لإنشاء كائن مصنف `Workbook` جديد وفارغ، لذا أدخِل مـا يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook() # إنشاء مصنف فارغ
>>> wb.sheetnames # يبدأ المصنف بورقة واحدة
['Sheet']
>>> sheet = wb.active
>>> sheet.title
'Sheet'
>>> sheet.title = 'Meat Eggs Sheet' # تغيير العنوان
>>> wb.sheetnames
['Meat Eggs Sheet']
```

سيبدأ المصنف بورقة واحدة تسمى "Sheet"، ويمكنك تغيير اسـم الورقـة من خلال تخـزين سلسـلة نصـية جديدة في السمة Attribute الخاصة بها وهي `title`.

لن يُحفظ ملف جدول البيانات عند تعديل كائن `Workbook` أو أوراقه وخلاياه حتى اسـتدعاء تـابع المصـنف `save()`. أدخِل ما يلي في الصدفة التفاعلية (مع الملف `example.xlsx` في مجلد العمل الحالي):

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.active
>>> sheet.title = 'Spam Spam Spam'
>>> wb.save('example_copy.xlsx') # حفظ المصنف
```

غيّرنا اسم الورقة، وحفظنا التغييرات من خلال تمرير اسم الملف كسلسلة نصية إلى التابع `save()`. يؤدي تمرير اسم ملف مختلف عن الاسم الأصلي -مثل الاسم `'example_copy.xlsx'`- إلى حفظ التغييرات في نسخة من جدول البيانات.

إذا عدّلتَ على جدول بيانات حمّلته من ملف، فيجب عليك دائمًا حفظ جدول البيانات الجديد المُعدَّل باسم ملفٍ مختلف عن اسم الملف الأصلي، وبذلك سيظل لديك ملف جدول البيانات الأصلي للعمل عليه في حالة وجود خطأ في شيفرتك البرمجية، وهو ما يؤدي إلى احتواء الملف الجديد المحفوظ على بيانات غير صحيحة أو تالفة.

13.5.2 إنشاء وحذف الأوراق

يمكن إضافة الأوراق وحذفها من المصنف باستخدام التابع `create_sheet()` والعامل `del`. إذًا لندخِل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> wb.sheetnames
['Sheet']
>>> wb.create_sheet() # إضافة ورقة جديدة
<Worksheet "Sheet1">
>>> wb.sheetnames
['Sheet', 'Sheet1']
>>> # إنشاء ورقة جديدة في الفهرس 0
>>> wb.create_sheet(index=0, title='First Sheet')
<Worksheet "First Sheet">
>>> wb.sheetnames
['First Sheet', 'Sheet', 'Sheet1']
>>> wb.create_sheet(index=2, title='Middle Sheet')
<Worksheet "Middle Sheet">
>>> wb.sheetnames
['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']
```

يعيد التابع `create_sheet()` كائن `Worksheet` جديد اسمه `SheetX`، والذي ضُبِط افتراضيًا ليكون الورقة الأخيرة في المصنف. يمكن اختياريًا تحديد فهرس واسم الورقة الجديدة باستخدام وسطاء الكلمات المفتاحية `index` و `title`.

لنتابع المثال السابق من خلال كتابة ما يلي:

```
>>> wb.sheetnames
['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']
>>> del wb['Middle Sheet']
>>> del wb['Sheet1']
>>> wb.sheetnames
['First Sheet', 'Sheet']
```

يمكنك استخدام العامل `del` من أجل حذف ورقة من مصنف، وهذا يماثل استخدامه لحذف زوج مفتاح-قيمة من القاموس.

> **ملاحظة**: تذكّر استدعاء التابع `save()` لحفظ التغييرات بعد إضافة أوراق إلى المصنف أو حذفها منه.

13.5.3 كتابة القيم في الخلايا

تشبه كتابة القيم في الخلايا إلى حدٍ كبير كتابة القيم في المفاتيح الموجودة ضمن القاموس. إذًا لندخل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb['Sheet']
>>> sheet['A1'] = 'Hello, world!' # تعديل قيمة الخلية
>>> sheet['A1'].value
'Hello, world!'
```

إذا كان لديك إحداثيات الخلية كسلسلة نصية، فيمكنك استخدامها بالطريقة نفسها لاستخدام مفتاح القاموس في الكائن `Worksheet` لتحديد الخلية التي تريد الكتابة فيها.

13.6 تطبيق عملي: تحديث جدول بيانات

ستكتب في هذا التطبيق العملي برنامجًا لتحديث الخلايا في جدول بيانات مبيعات المنتجات، حيث سيبحث برنامجك في جدول البيانات، ويعثر على أنواع معينة من المنتجات، ويحدّث أسعارها. يمكنك تنزيل جدول البيانات الخاص بهذا التطبيق العملي، ويوضح الشكل التالي كيف يبدو جدول البيانات:

	A	B	C	D	E
1	PRODUCE	COST PER KiloGram	KiloGrams SOLD	TOTAL	
2	Potatoes	0.86	21.6	18.58	
3	Okra	2.26	38.6	87.24	
4	Fava beans	2.69	32.8	88.23	
5	Watermelon	0.66	27.3	18.02	
6	Garlic	1.19	4.9	5.83	
7	Parsnips	2.27	1.1	2.5	
8	Asparagus	2.49	37.9	94.37	
9	Avocados	3.23	9.2	29.72	
10	Celery	3.07	28.9	88.72	
11	Okra	2.26	40	90.4	

الشكل 57: جدول بيانات مبيعات المنتجات

يمثّل كل صف في هذا الجدول عملية بيع واحدة، والأعمدة هي نوع المنتج المباع (A)، والتكلفة لكل كيلوجرام من هذا المنتج (B)، وعدد الكيلوجرامات المباعة (C)، وإجمالي الإيرادات من عمليات البيع (D). يُضبَط عمود "الإجمالي TOTAL" على صيغة إكسل `=ROUND(B3*C3, 2)` التي تضرب تكلفة كل كيلوجرام بعدد الكيلوجرامات المُباعة وتقريب النتيجة إلى أقرب سنت.

ستحدّث الخلايا الموجودة في العمود TOTAL نفسها تلقائيًا باستخدام هذه الصيغة في حالة وجود تغيير في العمود B أو C.

لنفترض إدخال أسعار الثوم والكرفس والليمون بصورة غير صحيحة، مما يتركك أمام مهمة مملة تتمثل في المرور على آلاف الصفوف في جدول البيانات لتحديث تكلفة الكيلوجرام الواحد لصفوف الثوم Garlic والكرفس Celery والليمون Lemon.

لا يمكنك إجراء عملية بحث واستبدال "find-and-replace" بسيطة للسعر بسبب وجود عناصر أخرى لها السعر نفسه ولا نريد تغييرها بصورة خاطئة. قد يستغرق تنفيذ ذلك يدويًا ساعات بالنسبة لآلاف الصفوف، ولكن يمكنك كتابة برنامج يمكنه إنجاز ذلك في ثوانٍ، حيث يطبّق برنامجك ما يلي:

1. يتكرر على جميع الصفوف ضمن حلقة.
2. إذا كان الصف للثوم أو الكرفس أو الليمون، فسيغيِّر السعر.

وهذا يعني أن شيفرتك البرمجية يجب أن تطبّق الخطوات التالية:

1. فتح ملف جدول البيانات.
2. التحقق مما إذا كانت القيمة الموجودة في العمود A هي الكرفس `Celery` أو الثوم `Garlic` أو الليمون `Lemon` لجميع الصفوف.

3. إذا كان الأمر كذلك، فسيتحدّث السعر في العمود B.
4. حفظ جدول البيانات في ملف جديد حتى لا تفقد جدول البيانات القديم.

13.6.1 الخطوة الأولى: إعداد هيكل البيانات باستخدام معلومات التحديث

إليك الأسعار التي يجب تحديثها:

المنتج	السعر
الكرفس Celery	1.19
الثوم Garlic	3.07
الليمون Lemon	1.27

الجدول 18: معلومات التحديث

ويمكنك كتابة الشيفرة البرمجية التالية:

```
if produceName == 'Celery':
  cellObj = 1.19
if produceName == 'Garlic':
  cellObj = 3.07
if produceName == 'Lemon':
  cellObj = 1.27
```

يُعَد الحصول على بيانات المنتج والأسعار المُحدَّثة الثابتة باستخدام الطريقة السابقة أمـرًا غـير مناسـب إلى حدٍ ما، فإذا كنت بحاجة إلى تحديث جدول البيانات مرةً أخرى بأسعار مختلفة أو بمنتجات مختلفة، فيجب عليك تغيير الكثير من الشيفرة البرمجية، وبالتالي ستخاطر بإدخال أخطاء في كل مرة تغيّر فيها شيفرتك البرمجية.

يتمثّل الحل الأفضل في تخزين معلومات السعر المُصحَّحة في قاموس وكتابة شيفرتك البرمجية لاستخدام هيكل البيانات، لذا أدخِل الشيفرة التالية في تبويب جديد لإنشاء ملف جديد في محرّرك:

```
#! python3
# updateProduce.py - تصحيح أسعار المنتجات في جدول بيانات المبيعات
import openpyxl
wb = openpyxl.load_workbook('produceSales.xlsx')
sheet = wb['Sheet']
# أنواع المنتجات وأسعارها المُحدَّثة
```

```
PRICE_UPDATES = {'Garlic': 3.07,
        'Celery': 1.19,
        'Lemon': 1.27}

# التكرار على جميع الصفوف وتحديث الأسعار
```

احفظ الملف بالاسم updateProduce.py.

إذا أردتَ تحديث جدول البيانات مرةً أخرى، فيجب تحديث القاموس PRICE_UPDATES فقط دون تغيير أي شيفرة برمجية أخرى.

13.6.2 الخطوة الثانية: التحقق من الصفوف وتحديث الأسعار غير الصحيحة

سيتكرر الجزء التالي من البرنامج على كافة الصفوف الموجودة في جدول البيانات، لذا أضِف الشيفرة البرمجية التالية إلى نهاية الملف updateProduce.py:

```
#! python3

# updateProduce.py - تصحيح أسعار المنتجات في جدول بيانات المبيعات

--snip--

# التكرار على جميع الصفوف وتحديث الأسعار

❶ for rowNum in range(2, sheet.max_row):      # تخطي الصف الأول

  ❷ produceName = sheet.cell(row=rowNum, column=1).value

  ❸ if produceName in PRICE_UPDATES:

      sheet.cell(row=rowNum, column=2).value =
PRICE_UPDATES[produceName]


❹ wb.save('updatedProduceSales.xlsx')
```

نكرّر الشيفرة البرمجية على الصفوف بدءًا من الصف 2، لأن الصف 1 هو ترويسة الجدول ❶، ونخزّن الخلية الموجودة في العمود 1 (أي العمود A) في المتغير produceName ❷، حيث إذا كان هذا المتغير موجودًا بوصفه مفتاحًا في قاموس PRICE_UPDATES ❸، فيجب أن تعلم أن هذا الصف يجب تصحيح سعره، وسيكون السعر الصحيح في PRICE_UPDATES[produceName].

لاحظ مدى نظافة شيفرتك باستخدام قاموس PRICE_UPDATES، إذ لا توجد سوى تعليمة if واحدة فقط بدلًا من استخدام شيفرة تحتوي التعليمة if produceName == 'Garlic': مثلًا التي تكون ضرورية لكل نوعٍ من المنتجات يجب تحديثه. بما أن هذه الشيفرة البرمجية تستخدم قاموس PRICE_UPDATES بدلًا من كتابة شيفرة ثابتة لأسماء المنتجات وأسعارها المُحدَّثة ضمن حلقة for، فهذا يعني أنك ستعدّل قاموس PRICE_UPDATES فقط من دون تعديل على الشيفرة البرمجية، إذا احتاج جدول بيانات مبيعات المنتجات تغييراتٍ إضافية.

تحفظ الشيفرة البرمجية كائن Workbook في الملف updatedProduceSales.xlsx ❹ بعد المرور على جدول البيانات بأكمله وإجراء التغييرات، ولا تكتب معلوماتٍ جديدة مكان معلومات جدول البيانات القديم إذا احتوى برنامجك خطأً وكان جدول البيانات المُحدَّث خاطئًا. يمكنك حذف جدول البيانات القديم بعد التحقق من أن جدول البيانات المُحدَّث صحيحًا.

13.6.3 أفكار لبرامج مماثلة

يستخدم العديد من العاملين في وظائف مكتبية جداولَ بيانات إكسل طوال الوقت، لذا قد يكون البرنامج الذي يمكنه تعديل ملفات إكسل وكتابتها تلقائيًا مفيدًا جدًا لهم، إذ يمكنه فعل مثل هذا البرنامج الأمور التالية:

- قراءة البيانات من جدول بيانات واحد وكتابتها في أجزاء من جداول بيانات أخرى.
- قراءة البيانات من مواقع الويب أو الملفات النصية أو الحافظة وكتابتها في جدول بيانات.
- تنظيف البيانات تلقائيًا في جداول البيانات، فمثلًا يمكنه استخدام التعابير النمطية Regular Expressions لقراءة تنسيقات متعددة لأرقام الهواتف وتعديلها إلى تنسيق معياري واحد.

13.7 ضبط نمط الخط Font Style في الخلايا

قد يساعدك تنسيق خلايا أو صفوف أو أعمدة معينة في إبراز المناطق المهمة في جدول البيانات، إذ يمكن لبرنامجك في جدول بيانات المنتجات مثلًا تطبيق نص عريض على صفوف البطاطا Potato والثوم Garlic والجزر الأبيض Parsnip، أو يمكنه كتابة الصفوف التي تكلفة الكيلوجرام الواحد منها أكبر من 5 دولارات بخط مائل. قد يكون تنسيق أجزاء من جدول بيانات كبير أمرًا مملًا يدويًا، ولكن يمكن لبرامجك تطبيق ذلك مباشرةً.

بإمكانك تخصيص أنماط الخطوط في الخلايا من خلال استيراد الدالة Font() من الوحدة openpyxl.styles، مما يتيح لك كتابة Font() بدلًا من openpyxl.styles.Font() اختصارًا:

```
from openpyxl.styles import Font
```

ينشئ المثال التالي مصنفًا جديدًا ويضبط الخلية A1 ليكون الخط فيها خطًا مائلًا وبحجم 24 نقطة. إذًا لندخِل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> from openpyxl.styles import Font
>>> wb = openpyxl.Workbook()
>>> sheet = wb['Sheet']
❶ >>> italic24Font = Font(size=24, italic=True) # إنشاء الخط
❷ >>> sheet['A1'].font = italic24Font # تطبيق الخط في الخلية A1
>>> sheet['A1'] = 'Hello, world!'
>>> wb.save('styles.xlsx')
```

تعيد الدالة `Font(size=24, italic=True)` كائن `Font` المُخزَّن في المتغير `italic24Font` ❶، وتضبط وسطاء الكلمات المفتاحية الخاصة بالدالة `Font()` -مثل الوسيطين `size` و `italic`- معلومات تنسيق الكائن `Font`، وإذا أسندنا الكائن `italic24Font` ❷ إلى `sheet['A1'].font`، فستُطبَّق جميع معلومات تنسيق الخط على الخلية A1.

13.8 كائنات الخط Font

يمكن ضبط السمات `font` من خلال تمرير وسطاء الكلمات المفتاحية إلى الدالة `Font()`، حيث يوضّح الجدول التالي وسطاء الكلمات المفتاحية المُحتمَلة للدالة `Font()`:

وسيط الكلمة المفتاحية	نوع البيانات	الوصف
`name`	سلسلة نصية	اسم الخط مثل `'Calibri'` أو `'Times New Roman'`
`size`	عدد صحيح	حجم الخط مقاسًا بالنقاط
`bold`	قيمة منطقية	قيمته `True` إذا كان الخط عريضًا
`italic`	قيمة منطقية	قيمته `True` إذا كان الخط مائلًا

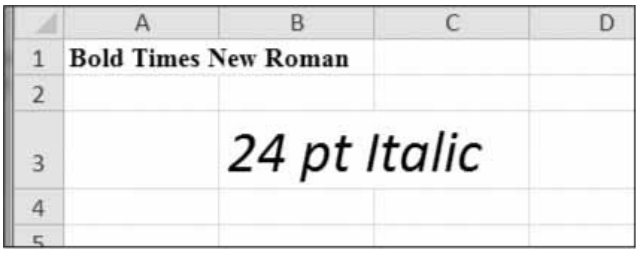
الجدول 19: وسطاء الكلمات المفتاحية المُحتمَلة للدالة Font()

يمكنك استدعاء الدالة `Font()` لإنشاء كائن `Font` وتخزينه في متغير، ثم تسند السمة `font` الخاصة بكائن `Cell` إلى هذا المتغير، فمثلًا تنشئ الشيفرة البرمجية التالية تنسيقات خطوط مختلفة:

```
>>> import openpyxl
>>> from openpyxl.styles import Font
>>> wb = openpyxl.Workbook()
```

```
>>> sheet = wb['Sheet']
>>> fontObj1 = Font(name='Times New Roman', bold=True)
>>> sheet['A1'].font = fontObj1
>>> sheet['A1'] = 'Bold Times New Roman'
>>> fontObj2 = Font(size=24, italic=True)
>>> sheet['B3'].font = fontObj2
>>> sheet['B3'] = '24 pt Italic'
>>> wb.save('styles.xlsx')
```

نخزّن كائن `Font` في المتغير `fontObj1` الذي نسنده إلى السمة `font` الخاصة بالكائن `Cell` للخليـة A1، ونكرر العملية نفسها مع كائن خط آخر لضبط خط الخلية الثانية. إذا نفّذنا الشيفرة البرمجيـة السـابقة، فسيُضـبَط تنسيق الخلايا A1 و B3 في جدول البيانات على تنسيقات الخطوط المُخصَّصة، وستبدو كما يلي:



الشكل 58: جدول بيانات يحتوي على أنماط خطوط مُخصَّصة

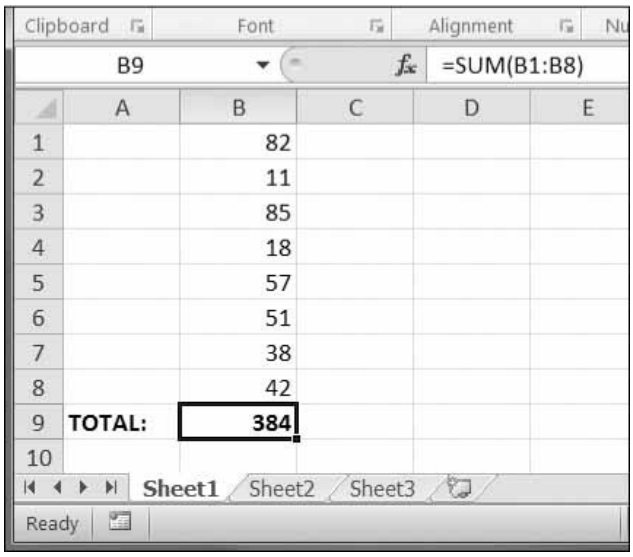
ضبطنا اسم الخط في الخلية A1 على القيمة `'Times New Roman'` والوسيط `bold` على القيمة `true`، حيث يظهر النص بالخط Times New Roman العريض، ولكننا لم نحـدد حجم الخـط، لـذلك اُسـتخدِم الحجم الافتراضي للوحدة `openpyxl`، وهو الحجم 11. استخدمنا الخط المائل وبحجم 24 في الخلية B3، ولكن لم نحدد اسم الخط، لذلك اُستخدِم الخط الافتراضي للوحدة `openpyxl`، وهو الخط Calibri.

13.9 صيغ إكسل

تضبط صيغ إكسل -التي تبدأ بإشارة يساوي- الخلايا لتحتـوي على قيم ناتجـة عن تطـبيق عمليـات حسـابية على خلايا أخرى، لذا سنستخدم في هذه الفقرة وحدة `openpyxl` لإضافة الصيغ إلى الخلايا برمجيًا مثل أيّ قيمة عادية أخرى كما يلي:

```
>>> sheet['B9'] = '=SUM(B1:B8)'
```

ستؤدي التعليمة السابقة إلى تخزين الصيغة `=SUM(B1:B8)` بوصفها قيمةً في الخلية B9، ممـا يـؤدي إلى ضبط الخلية B9 على صيغة تحسب مجموع القيم في الخلايا من B1 إلى B8 كما في الشكل التالي:



الشكل 59: تحتوي الخلية B9 على الصيغة =SUM(B1:B8)

تُضبَط صيغ إكسل مثل أيّ قيمة نصية أخرى في الخلية. إذًا لندخِل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.active
>>> sheet['A1'] = 200
>>> sheet['A2'] = 300
>>> sheet['A3'] = '=SUM(A1:A2)' # ضبط الصيغة
>>> wb.save('writeFormula.xlsx')
```

ضبطنا الخلايا A1 و A2 على القيمتين 200 و 300 على التوالي، وضبطنا القيمة الموجودة في الخلية A3 على صيغة تجمع القيم الموجودة في الخليتين A1 و A2، وبالتالي ستظهر قيمة الخلية A3 على أنها 500 عند فتح جدول البيانات في إكسل.

توفر صيغ إكسل مستوًى مقبولًا من البرمجة لجداول البيانات، ولكن تصبح هذه الصيغ غير قابلة للإدارة بسرعة بالنسبة للمهام المعقدة، فمثلًا حتى لو كنت على دراية كبيرة بصيغ إكسل، فمن الصعب محاولة تفسير ما تفعله الصيغة التالية:

```
=IFERROR(TRIM(IF(LEN(VLOOKUP(F7, Sheet2!$A$1:$B$10000, 2,
FALSE))>0,SUBSTITUTE(VLOOKUP(F7, Sheet2!$A$1:$B$10000, 2, FALSE), " ",
""),"")), "")
```

لاحظ أن شيفرة بايثون البرمجية أكثر قابلية للقراءة من الصيغة السابقة.

13.10 تعديل الصفوف والأعمدة

يُعَد تعديل أحجام الصفوف والأعمدة في برنامج إكسل أمرًا سهلًا مثل النقر على حواف ترويسة الصف أو العمود وسحبها، ولكن إذا أردتَ ضبط حجم صف أو عمود بناءً على محتويات خلاياه أو إذا أردتَ ضبط الأحجام في عدد كبير من ملفات جداول البيانات، فستكون كتابة برنامج بايثون لذلك أسرع بكثير.

يمكن أيضًا إخفاء الصفوف والأعمدة بصورة كاملة، أو يمكن تثبيتها بحيث تكون مرئية دائمًا على الشاشة وتظهر في جميع الصفحات عند طباعة جدول البيانات، إذ يُعَد ذلك مفيدًا للعناوين.

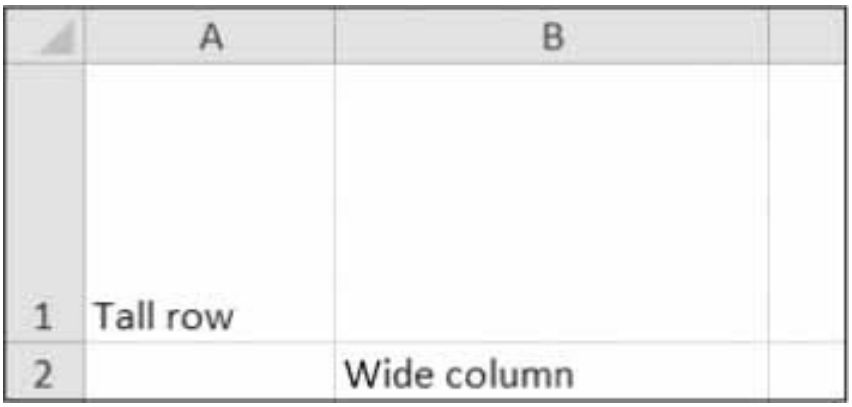
13.10.1 ضبط ارتفاع الصف وعرض العمود

تمتلك كائنات `Worksheet` سمات `row_dimensions` و `column_dimensions` التي تتحكم في ارتفاع الصفوف وعرض الأعمدة. إذًا لندخِل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.active
>>> sheet['A1'] = 'Tall row'
>>> sheet['B2'] = 'Wide column'
>>> # ضبط الارتفاع والعرض
>>> sheet.row_dimensions[1].height = 70
>>> sheet.column_dimensions['B'].width = 20
>>> wb.save('dimensions.xlsx')
```

تُعَد السمات `row_dimensions` و `column_dimensions` الخاصة بالورقة قيمًا تشبه القاموس، إذ تحتوي السمة `row_dimensions` على كائنات `RowDimension` وتحتوي السمة `column_dimensions` على كائنات `ColumnDimension`. يمكنك الوصول إلى أحد الكائنات باستخدام رقم الصف (في مثالنا 1 أو 2) في `row_dimensions`، ويمكنك الوصول إلى أحد الكائنات باستخدام حرف العمود (في مثالنا A أو B) في `column_dimensions`.

يبدو جدول البيانات `dimensions.xlsx` كما يلي:



الشكل 60: ضبطنا الصف 1 والعمود B على ارتفاع وعرض أكبر

بإمكانك ضبط ارتفاع الكائن `RowDimension` بعد الحصول عليه، كما يمكنك ضبط عرض الكائن `ColumnDimension` بعد الحصول عليه. يمكن ضبط ارتفاع الصف ليكون عددًا صحيحًا أو عشريًا قيمته بين 0 و 409، إذ تمثّل هذه القيمة الارتفاع المُقاس بالنقاط، وتساوي النقطة الواحدة 1/72 من البوصة، ويكون ارتفاع الصف الافتراضي 12.75. يمكن ضبط عرض العمود ليكون عددًا صحيحًا أو عشريًا قيمته بين 0 و255، إذ تمثّل هذه القيمة عدد المحارف التي يمكن عرضها في الخلية بحجم الخط الافتراضي (11 نقطة)، ويكون عرض العمود الافتراضي 8.43 محرفًا. تُخفَى الأعمدة التي يبلغ عرضها 0 أو الصفوف التي يبلغ ارتفاعها 0 عن المستخدم.

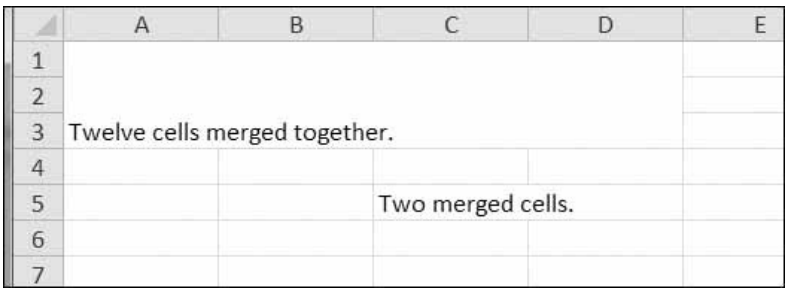
13.10.2 دمج وإلغاء دمج الخلايا

يمكن دمج منطقة مستطيلة من الخلايا في خلية واحدة باستخدام التابع `merge_cells()` الخاص بالورقة. إذًا لندخِل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.active
>>> sheet.merge_cells('A1:D3') # دمج جميع هذه الخلايا
>>> sheet['A1'] = 'Twelve cells merged together.'
>>> sheet.merge_cells('C5:D5') # دمج هاتين الخليتين
>>> sheet['C5'] = 'Two merged cells.'
>>> wb.save('merged.xlsx')
```

وسيط التابع `merge_cells()` هو سلسلة نصية واحدة من الخلايا العلوية اليسرى والسفلية اليمنى للمنطقة المستطيلة المُراد دمجها، حيث تدمج `'A1:D3'` اثنتا عشر خلية في خلية واحدة، ويمكنك ضبط قيمة هذه الخلايا المدموجة من خلال ضبط قيمة الخلية العلوية اليسرى لمجموعة الخلايا المدموجة.

سيبدو الملف `merged.xlsx` كما يلي عند تشغيل الشيفرة البرمجية السابقة:



الشكل 61: الخلايا المدموجة في جدول البيانات

يمكن إلغاء دمج الخلايا من خلال استدعاء التابع `unmerge_cells()` الخاص بالورقة. إذًا لندخِل ما يلي الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('merged.xlsx')
>>> sheet = wb.active
>>> sheet.unmerge_cells('A1:D3') # فصل هذه الخلايا عن بعضها
>>> sheet.unmerge_cells('C5:D5')
>>> wb.save('merged.xlsx')
```

إذا حفظتَ تغييراتك ثم ألقيتَ نظرة على جدول البيانات، فسترى أن الخلايا المدموجة عادت إلى كونها خلايا مفردة.

13.10.3 تثبيت الأجزاء

من المفيد تثبيت عددٍ من الصفوف العلوية أو الأعمدة الموجودة في أقصى اليسار على الشاشة في جداول البيانات الكبيرة جدًا التي لا يمكن عرضها كاملة، فمثلًا تكون ترويسات الأعمدة أو الصفوف المُثبَّتة مرئيةً للمستخدم دائمًا حتى أثناء التمرير في جدول البيانات، ويُعرَف ذلك بتثبيت الأجزاء Freeze Panes.

يحتوي كل كائن `Worksheet` في وحدة `OpenPyXL` على السمة `freeze_panes` التي يمكن ضبطها على كائن `Cell` أو سلسلة نصية من إحداثيات الخلية.

لاحظ تثبيت كافة الصفوف الموجودة أعلى هذه الخلية وجميع الأعمدة الموجودة على يسارها، ولكن لن يُثبَّت صف وعمود الخلية نفسها.

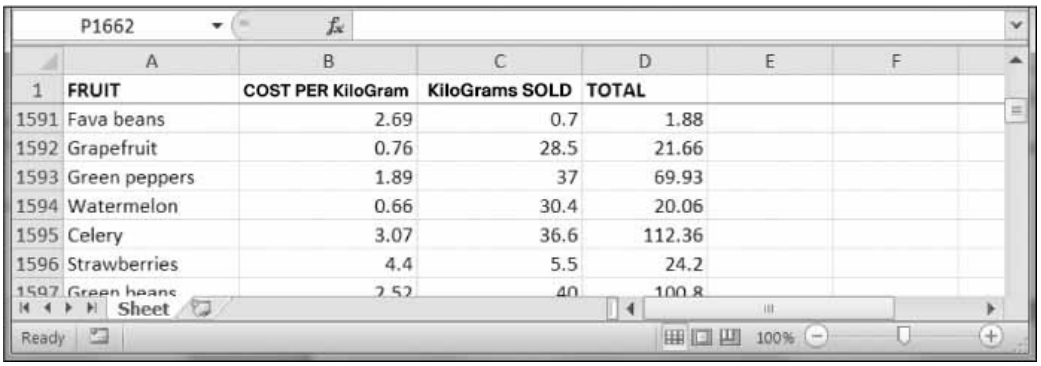
ضبط السمة freeze_panes	الصفوف والأعمدة المُثبَّتة
`sheet.freeze_panes = 'A2'`	الصف 1
`sheet.freeze_panes = 'B1'`	العمود A
`sheet.freeze_panes = 'C1'`	العمودان A و B
`sheet.freeze_panes = 'C2'`	الصف 1 والعمودان A و B
`sheet.freeze_panes = 'A1'` أو `sheet.freeze_panes = None`	لا توجد أجزاء مُثبَّتة

الجدول 20:الصفوف والأعمدة التي ستُثبَّت أحيانًا عند ضبط قيمة freeze_panes

تأكّد من حصولك على جدول بيانات مبيعات المنتجات، ثم أدخِل ما يلي في الصدفة التفاعلية:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('produceSales.xlsx')
>>> sheet = wb.active
>>> sheet.freeze_panes = 'A2' # تثبيت الصفوف الموجودة أعلى الصف A2
>>> wb.save('freezeExample.xlsx')
```

إذا ضبطتَ السمة `freeze_panes` على القيمة `'A2'`، فسيُعرَض الصف 1 دائمًا بغض النظر عن المكان الذي ينتقل إليه المستخدم عند التمرير في جدول البيانات، ويمكنك رؤية ذلك في الشكل التالي:



الشكل 62: ضبط السمة freeze_panes على القيمة 'A2'

يكون الصف 1 مرئيًا دائمًا حتى عندما يمرّر المستخدم جدول البيانات إلى الأسفل، إذا ضبطنا السمة `freeze_panes` على القيمة `'A2'`.

13.11 المخططات Charts

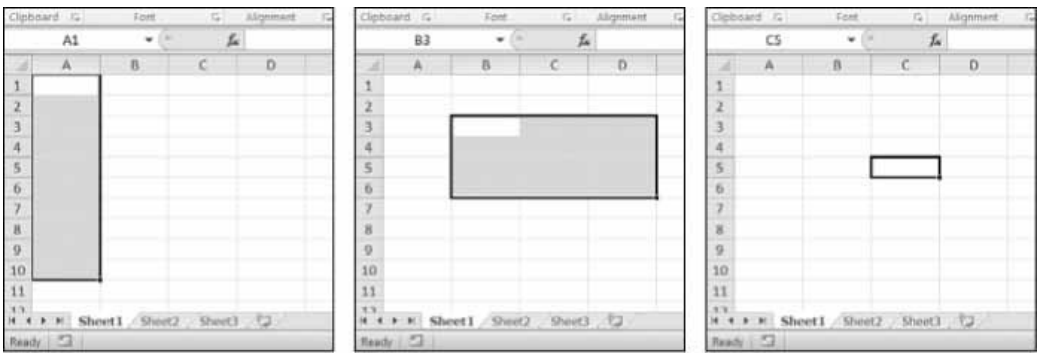
تدعم الوحدة `OpenPyXL` إنشاء مخططات شريطية وخطية ومبعثرة ودائرية باستخدام البيانات الموجودة في خلايا الورقة، حيث يمكنك إنشاء مخطط باتباع الخطوات التالية:

1. إنشاء كائن `Reference` من خلايا المنطقة المستطيلة المُحدّدة.
2. إنشاء كائن `Series` من خلال تمرير الكائن `Reference`.
3. إنشاء كائن `Chart`.
4. إلحاق كائن `Series` بكائن `Chart`.
5. إضافة الكائن `Chart` إلى الكائن `Worksheet` مع تحديد الخلية التي يجب أن تكون في الزاوية العلوية اليسرى من المخطط اختياريًا.

يمكنك إنشاء كائنات `Reference` من خلال استدعاء الدالة `openpyxl.chart.Reference()` وتمرير ثلاثة وسطاء هي:

1. كائن `Worksheet` الذي يحتوي على بيانات مخططك.
2. مجموعة `Tuple` مكونة من عددين صحيحين. تمثل هذه المجموعة الخلية العلوية اليسرى من خلايا المنطقة المستطيلة المُحدَّدة التي تحتوي على بيانات مخططك، ويمثل العددُ الصحيح الأول في المجموعة الصفَّ، ويمثل العدد الصحيح الثاني العمود. لاحظ أن العدد 1 هو الصف الأول وليس العدد 0.
3. مجموعة مكونة من عددين صحيحين، حيث تمثل هذه المجموعة الخلية السفلية اليمنى من خلايا المنطقة المستطيلة المُحدَّدة التي تحتوي على بيانات مخططك، ويمثل العدد الصحيح الأول في المجموعة الصف، ويمثل العدد الصحيح الثاني العمود.

يوضح الشكل التالي بعض النماذج من وسطاء الإحداثيات:



الشكل 63: بعض النماذج من وسطاء الإحداثيات

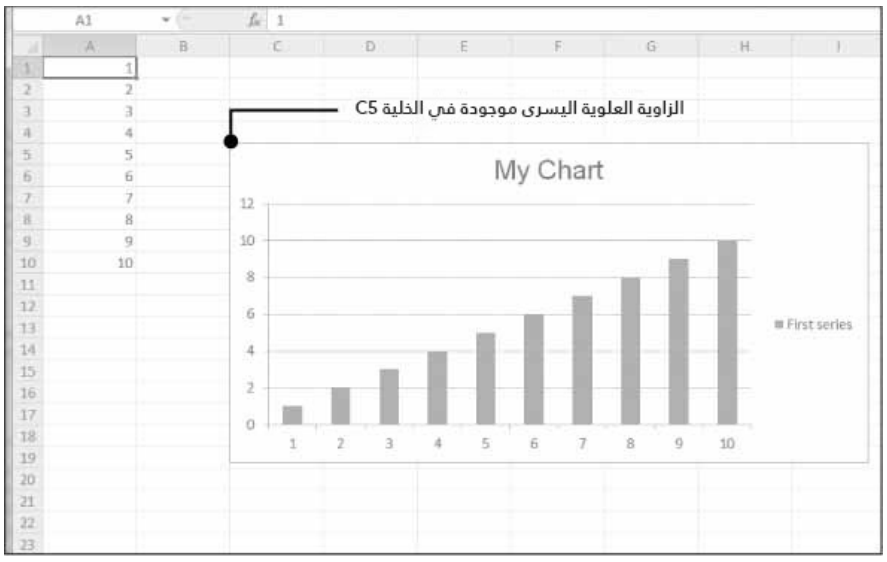
وهي من اليسار إلى اليمين:

```
(1, 1), (10, 1); (3, 2), (6, 4); (5, 3), (5, 3)
```

أدخِل مثال الصدفة التفاعلية التالي لإنشاء مخطط شريطي وإضافته إلى جدول البيانات:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.active
>>> for i in range(1, 11): # إنشاء بعض البيانات في العمود A
...     sheet['A' + str(i)] = i
...
>>> refObj = openpyxl.chart.Reference(sheet, min_col=1, min_row=1,
max_col=1,
max_row=10)
>>> seriesObj = openpyxl.chart.Series(refObj, title='First series')

>>> chartObj = openpyxl.chart.BarChart()
>>> chartObj.title = 'My Chart'
>>> chartObj.append(seriesObj)

>>> sheet.add_chart(chartObj, 'C5')
>>> wb.save('sampleChart.xlsx')
```

وينتج عن ذلك جدول بيانات يشبه ما يلي:



الشكل 64: جدول بيانات مع مخطط مضافٍ إليه

أنشأنا مخططًا شريطيًا من خلال استدعاء الدالة `openpyxl.chart.BarChart()`، ويمكنك إنشاء مخططات خطية ومخططات أخرى مبعثرة، وكذلك مخططات دائرية من خلال استدعاء الدوال

`openpyxl.charts.LineChart()` و `openpyxl.charts.LineChart()`
و`openpyxl.charts.LineChart()`.

13.12 أسئلة للتدريب

لنفترض أن لديك كائن `Workbook` في المتغير `wb`، وكائن `Worksheet` في المتغير `sheet`، وكائن `Cell` في المتغير `cell`، وكائن `Comment` في المتغير `comm`، وكائن `Image` في المتغير `img`.

أجب عن الأسئلة التالية:

1. ما الذي تعيده الدالة `openpyxl.load_workbook()`؟
2. ما الذي تحتويه سمة المصنف `wb.sheetnames`؟
3. كيف يمكنك استرداد كائن `Worksheet` لورقة اسمها `'Sheet1'`؟
4. كيف يمكنك استرداد كائن `Worksheet` لورقة المصنف النشطة؟
5. كيف يمكنك استرداد القيمة الموجودة في الخلية C5؟
6. كيف يمكنك ضبط القيمة الموجودة في الخلية C5 على القيمة `"Hello"`؟
7. كيف يمكنك استرداد صف وعمود الخلية كأعداد صحيحة؟
8. ما محتوى سمات الورقة `sheet.max_column` و `sheet.max_row`؟ وما نوع بيانات هذه السمات؟
9. إذا أردتَ الحصول على فهرس العمود `'M'` كعدد صحيح، فما هي الدالة التي يجب استدعاؤها؟
10. إذا أردتَ الحصول على اسم العمود 14 كسلسلة نصية، فما هي الدالة التي يجب استدعاؤها؟
11. كيف يمكنك استرداد مجموعة مؤلفة من جميع كائنات `Cell` من الخلية A1 إلى الخلية F1؟
12. كيف يمكنك حفظ المصنف بالاسم `example.xlsx`؟
13. كيف يمكنك ضبط صيغة في الخلية؟
14. إذا أردتَ استرداد نتيجة صيغة الخلية بدلًا من الصيغة الخاصة بالخلية، فماذا يجب أن تفعله أولًا؟
15. كيف يمكنك ضبط ارتفاع الصف 5 على القيمة 100؟
16. كيف يمكنك إخفاء العمود C؟
17. ما هي الأجزاء المُثبَّتة؟
18. ما هي الدوال والتوابع الخمسة التي يجب عليك استخدامها لإنشاء مخطط شريطي؟

13.13 مشاريع للتدريب

حاول كتابة البرامج التي تؤدي المهام التي سنوضّحها فيما يلي، لكسب خبرة عملية أكبر.

13.13.1 برنامج لإنشاء جدول الضرب

أنشئ برنامج `multiplicationTable.py` الذي يأخذ العدد `N` من سطر الأوامر وينشئ جدول الضرب N×N في جدول بيانات إكسل، فمثلًا إذا شغّلنا البرنامج كما يلي:

```
py multiplicationTable.py 6
```

يجب أن ينشئ جدول بيانات يشبه الشكل التالي:

	A	B	C	D	E	F	G	H
1		**1**	**2**	**3**	**4**	**5**	**6**	
2	**1**	1	2	3	4	5	6	
3	**2**	2	4	6	8	10	12	
4	**3**	3	6	9	12	15	18	
5	**4**	4	8	12	16	20	24	
6	**5**	5	10	15	20	25	30	
7	**6**	6	12	18	24	30	36	
8								
9								

الشكل 65: توليد جدول الضرب في جدول بيانات

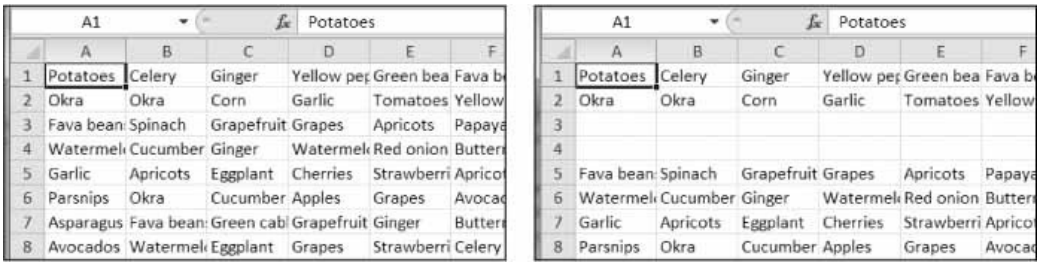
يجب عليك استخدام الصف 1 والعمود A للتسميات أو عناوين الصفوف والأعمدة، كما ويجب أن يكونا بالخط العريض.

13.13.2 برنامج لإدراج صف فارغ

أنشئ برنامج `blankRowInserter.py` الذي يأخذ عددين صحيحين وسلسلة نصية لاسم الملف كوسطاء لسطر الأوامر، حيث نسمّي العدد الصحيح الأول `N` والعدد الصحيح الثاني `M`. يجب على البرنامج إدراج صفوف فارغة بعدد `M` في جدول البيانات بدءًا من الصف `N`، فمثلًا إذا شغّلنا البرنامج كما يلي:

```
python blankRowInserter.py 3 2 myProduce.xlsx
```

يجب أن تبدو جداول البيانات "قبل" و"بعد" الإدراج كما في الشكل التالي:



	A	B	C	D	E	F
1	Potatoes	Celery	Ginger	Yellow pe	Green bea	Fava b
2	Okra	Okra	Corn	Garlic	Tomatoes	Yellow
3	Fava bean	Spinach	Grapefruit	Grapes	Apricots	Papaya
4	Watermel	Cucumber	Ginger	Watermel	Red onion	Butter
5	Garlic	Apricots	Eggplant	Cherries	Strawberri	Apricot
6	Parsnips	Okra	Cucumber	Apples	Grapes	Avocac
7	Asparagus	Fava bean	Green cab	Grapefruit	Ginger	Butter
8	Avocados	Watermel	Eggplant	Grapes	Strawberri	Celery

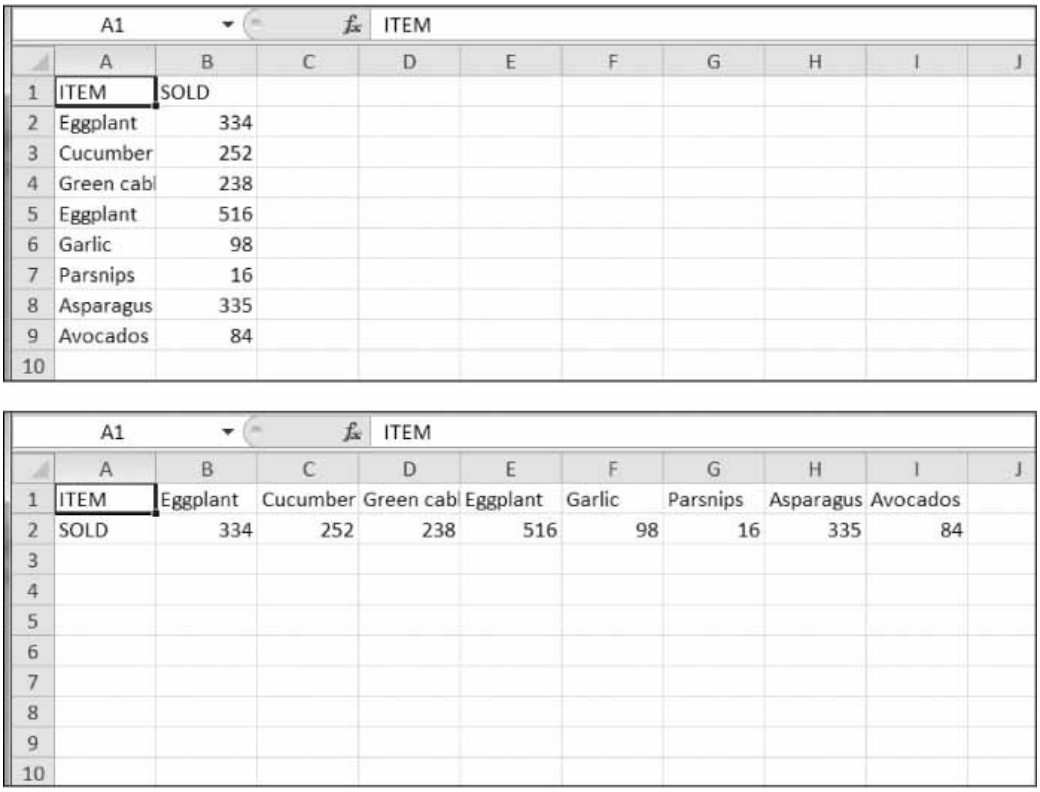
	A	B	C	D	E	F
1	Potatoes	Celery	Ginger	Yellow pe	Green bea	Fava b
2	Okra	Okra	Corn	Garlic	Tomatoes	Yellow
3						
4						
5	Fava bean	Spinach	Grapefruit	Grapes	Apricots	Papaya
6	Watermel	Cucumber	Ginger	Watermel	Red onion	Butter
7	Garlic	Apricots	Eggplant	Cherries	Strawberri	Apricot
8	Parsnips	Okra	Cucumber	Apples	Grapes	Avocac

الشكل 66: قبل (يسار) وبعد (يمين) إدراج الصفين الفارغين عند الصف 3

بإمكانك كتابة هذا البرنامج من خلال قراءة محتويات جدول البيانات، ثم استخدام حلقة `for` لنسخ الأسطر `N` الأولى عند كتابة جدول البيانات الجديد، وجمع العدد `M` مع رقم الصف في جدول البيانات الناتج بالنسبة للأسطر المتبقية.

13.13.3 برنامج لعكس خلايا جدول البيانات

اكتب برنامجًا لعكس الصف والعمود الخاص بالخلايا في جدول البيانات، فمثلًا ستكون القيمة في الصف 5 والعمود 3 موجودة في الصف 3 والعمود 5 والعكس صحيح، ويجب تطبيق ذلك على جميع الخلايا في جدول البيانات، إذ ستبدو جداول البيانات "قبل" و"بعد" العكس كما في الشكل التالي:



	A	B
1	ITEM	SOLD
2	Eggplant	334
3	Cucumber	252
4	Green cabl	238
5	Eggplant	516
6	Garlic	98
7	Parsnips	16
8	Asparagus	335
9	Avocados	84
10		

	A	B	C	D	E	F	G	H	I	J
1	ITEM	Eggplant	Cucumber	Green cabl	Eggplant	Garlic	Parsnips	Asparagus	Avocados	
2	SOLD	334	252	238	516	98	16	335	84	
3										
4										
5										
6										
7										
8										
9										
10										

الشكل 67: جدول البيانات قبل (العلوي) وبعد (السفلي)

يمكنك كتابة هذا البرنامج باستخدام حلقات `for` متداخلة لقراءة بيانات جدول البيانات في قائمة من قوائم هيكل البيانات، ويمكن أن يحتوي هيكل البيانات على `sheetData[x][y]` للخلية الموجودة في العمود x والصف y. استخدم بعد ذلك `sheetData[y][x]` للخلية الموجودة في العمود x والصف y عند كتابة جدول البيانات الجديد.

13.13.4 برنامج لتحويل الملفات النصية إلى جدول بيانات

اكتب برنامجًا لقراءة محتويات العديد من الملفات النصية (يمكنك إنشاء الملفات النصية بنفسك) وإدراج هذه المحتويات في جدول بيانات، بحيث يقابل سطر واحد من الملف النصي صفًا واحدًا من جدول البيانات. وهنا ستكون أسطر الملف النصي الأول في خلايا العمود A، وستكون أسطر الملف النصي الثاني في خلايا العمود B، وهكذا.

استخدم التابع `readlines()` الخاص بالكائن `File` لإعادة قائمة من السلاسل النصية، حيث تقابل كل سلسلة نصية واحدة سطرًا في الملف. يكون السطر الأول من الملف الأول مقابلًا للعمود 1 والصف 1، ويجب كتابة السطر الثاني في العمود 1 والصف 2، وما إلى ذلك. سيُكتَب الملف التالي المقروء باستخدام التابع `readlines()` في العمود 2، وسيُكتَب الملف الذي يليه في العمود 3، وهكذا.

13.13.5 برنامج لتحويل جدول بيانات إلى ملفات نصية

اكتب برنامجًا يؤدّي مهام البرنامج السابق بترتيب عكسي، إذ يجب أن يفتح البرنامج جدول بيانات ويكتب خلايا العمود A في ملف نصي واحد، وخلايا العمود B في ملف نصي آخر، وهكذا.

13.14 الخلاصة

لا يكون الجزء الصعب من معالجة المعلومات هو المعالجة بحد ذاتها في كثير من الأحيان، بل تتمثل الصعوبة ببساطة في الحصول على البيانات بالتنسيق الصحيح المناسب لبرنامجك، ولكن يمكنك استخراج بيانات جدول بياناتك ومعالجتها بصورة أسرع بكثير مما يمكنك فعله يدويًا بعد تحميله إلى ملف بايثون.

يمكنك أيضًا توليد جداول بيانات بوصفها خرجًا لبرامجك، لذا إذا كان زملاؤك بحاجة إلى نقل ملفك النصي أو ملف PDF الذي يحتوي على آلاف جهات اتصال المبيعات إلى ملف جدول بيانات، فلن تضطر إلى نسخه ولصقه بالكامل في ملف إكسل. إذا كان لديك وحدة `openpyxl` وبعض المعرفة البرمجية، فستجد أن معالجة جداول البيانات الكبيرة تُعَد أمرًا سهلًا للغاية.

سنلقي نظرة في الفصل التالي على استخدام لغة بايثون للتفاعل مع برنامج آخر لجداول بيانات، وهو تطبيق جداول بيانات جوجل Google Sheets الشهير على الإنترنت.

دورة إدارة تطوير المنتجات

مميزات الدورة

- بناء معرض أعمال قوي بمشاريع حقيقية
- وصول مدى الحياة لمحتويات الدورة
- تحديثات مستمرة على الدورة مجانًا
- شهادة معتمدة من أكاديمية حسوب
- إرشادات من المدربين على مدار الساعة
- من الصفر دون الحاجة لخبرة مسبقة

اشترك الآن

14. العمل مع جداول بيانات جوجل

يُعَد تطبيق جداول بيانات جوجل Google Sheets تطبيقًا مجانيًا ومستندًا إلى الويب ومتاحًا لأيّ شخص لديه حساب جوجل Google أو عنوان جيميل Gmail، وأصبح منافسًا مفيدًا وغنيًا بالميزات لبرنامج إكسل Excel. تحتوي جداول بيانات جوجل على واجهة برمجة تطبيقات API خاصة بها، ولكن يمكن أن تكون هذه الواجهة مربكةً بعض الشيء في عملية التعلم والاستخدام.

سنغطّي في هذا الفصل وحدة `EZSheets` الخارجية والموثقة على موقعها الرسمي، والتي لا تُعَد كاملة الميزات مثل واجهة برمجة تطبيقات جداول البيانات الرسمية من جوجل، ولكنها تسهّل تنفيذ مهام جداول البيانات الشائعة.

14.1 تثبيت وإعداد وحدة EZSheets

يمكنك تثبيت وحدة `EZSheets` من خلال فتح نافذة طرفية جديدة وتشغيل الأمر

```
pip install --user ezsheets
```

وستثبِّت وحدة `EZSheets` أيضًا كجزء من هذا التثبيت الوحدات `google-api-python-client` و`google-auth-httplib2` و`google-auth-oauthlib`، حيث تسمح هذه الوحدات لبرنامجك بتسجيل الدخول إلى خوادم جوجل وإنشاء طلبات واجهة برمجة التطبيقات API. تعالج وحدة `EZSheets` عملية التفاعل مع هذه الوحدات، لذلك لا داعي للقلق بشأن كيفية عملها.

14.1.1 الحصول على الاعتماديات والملفات المفتاحية

يجب تفعيل جداول بيانات جوجل وواجهات برمجة تطبيقات جوجل درايف Google Drive على حسابك على جوجل قبل أن تتمكّن من استخدام وحدة `EZSheets`.

انتقل إلى صفحتي الويب التاليتين وانقر على زر التفعيل `Enable API` الموجودة في أعلى كل منهما:

- sheets.googleapi
- drive.googleapis

يجب أيضًا أن تحصل على ثلاثة ملفات، والتي يجب حفظها في المجلد نفسه لسكربت بايثون Python الذي امتداده `.py` ويستخدم وحدة `EZSheets`، وهذه الملفات هي:

- ملف الاعتماديات واسمه `credentials-sheets.json`.
- مفتاح Token جداول بيانات جوجل واسمه `token-sheets.pickle`.
- مفتاح Token جوجل درايف واسمه `token-drive.pickle`.

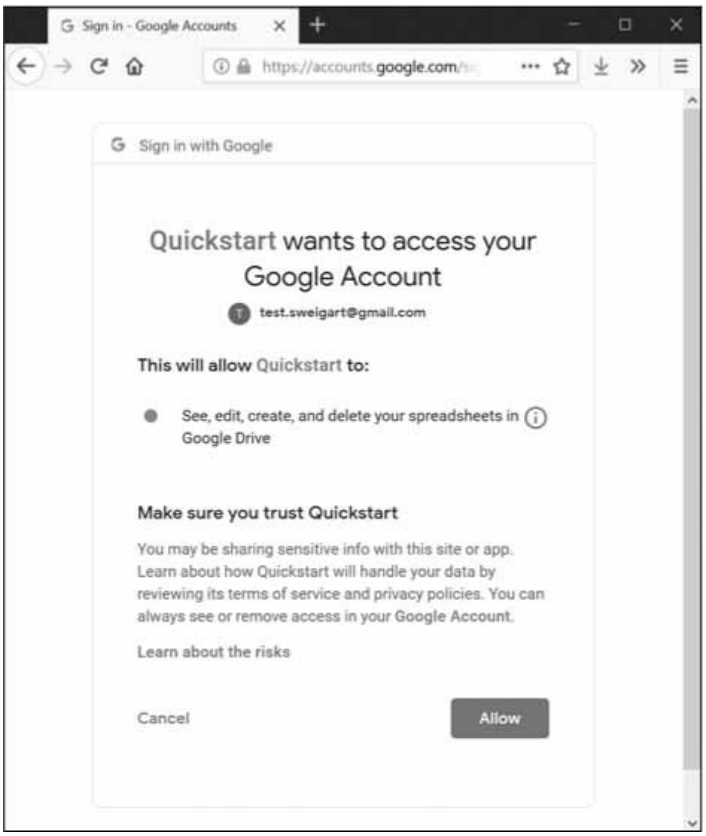
يولّد ملف الاعتماديات ملفات المفاتيح، وأسهل طريقة للحصول على ملف الاعتماديات هي الانتقال إلى صفحة Google Sheets Python Quickstart والنقر على زر التفعيل الملون باللون الأزرق `Enable the Google Sheets API` كما هو موضح في الشكل التالي، ولكن يجب أن تسجّل الدخول إلى حسابك في جوجل لعرض هذه الصفحة:

الشكل 68: الحصول على ملف credentials.json

سيؤدي النقر على هذا الزر إلى ظهور نافذة تحتوي على رابط تنزيل ضبط العميل `Download Client Configuration` الذي يتيح لك تنزيل ملف `credentials.json`. أعِد تسمية هذا الملف إلى الاسم `credentials-sheets.json` وضعه في المجلد نفسه لسكربتات بايثون الخاصة بك.

شـغّل الأمـر `import ezsheets` من أجـل اسـتيراد وحـدة `EZSheets` بعـد الحصـول على الملـف `credentials-sheets.json`. ستفتح نافذة متصفح جديدة لتتمكّن من تسجيل الدخول إلى حسـابك على جوجل عند استيراد وحدة `EZSheets` في المرة الأولى.

انقر بعد ذلك على زر السماح `Allow` كما هو موضح في الشكل التالي:



الشكل 69: السماح لصفحة Python Quickstart بالوصول إلى حسابك على جوجل

سبب ظهور الرسالة السـابقة هـو أنـك نـزّلتَ ملـف الاعتماديـات من صـفحة Google Sheets Python Quickstart، وستفتح هذه النافذة مرتين: الأولى للوصول إلى جداول بيانات جوجل والثانية للوصول إلى جوجـل درايف، حيث تستخدم وحدةُ `EZSheets` الوصولَ إلى جوجل درايف لرفع جداول البيانات وتنزيلها وحذفها.

ستطالبك نافذة المتصـفح بإغلاقـه بعـد تسـجيل الـدخول، وسـيظهر الملفـان `token-sheets.pickle` و`token-drive.pickle` في المجلـد نفسـه الـذي يوجـد فيـه الملـف `credentials-sheets.json`. ستجري هذه العملية فقط في المرة الأولى التي تشغِّل فيها الأمر `import ezsheets`.

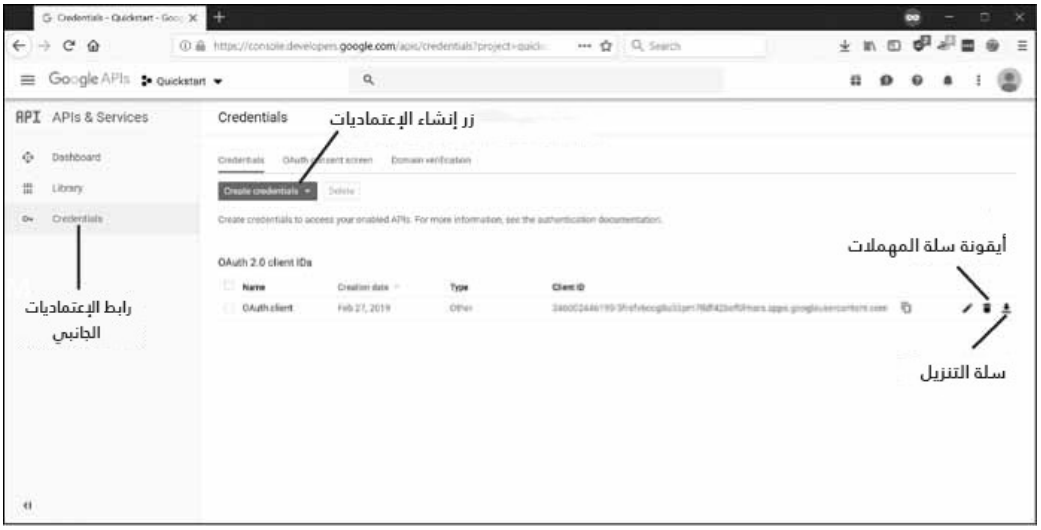
إذا واجهتَ خطأً بعد النقر على زر السماح "Allow" وكانت الصـفحة معطَّلـة، فتأكّـد أولًا من تفعيـل جـداول بيانات جوجل وواجهات برمجة تطبيقات جوجل درايف من الروابط الموجودة في بداية هذا القسم. قـد يسـتغرق

الأمر بضع دقائق حتى تتمكّن خوادم جوجل من تسجيل هذا التغيير، لذا قد تضطر إلى الانتظار قبل أن تتمكّن من استخدام وحدة `EZSheets`.

> **ملاحظة**: لا تشارك ملفات الاعتماديات أو المفاتيح مع أيّ شخص، وتعامل معها مثل كلمات المرور.

14.1.2 إبطال ملف الاعتماديات

إذا شاركتَ ملفات الاعتماديات أو المفاتيح مع شخصٍ ما عن طريق الخطأ، فلن يتمكّن هذا الشخص من تغيير كلمة مرور حسابك على جوجل، ولكن سيكون لديه حق الوصول إلى جداول بياناتك. يمكنك إبطال هذه الملفات بالانتقال إلى صفحة طرفية المطور على منصة سحابة جوجل Google Cloud Platform، ولكن يجب تسجيل الدخول إلى حسابك على جوجل لعرض هذه الصفحة. انقر على رابط الاعتماديات Credentials في الشريط الجانبي، ثم انقر على أيقونة سلة المهملات بجانب ملف الاعتماديات الذي شاركته عن طريق الخطأ، كما هو موضح في الشكل التالي:



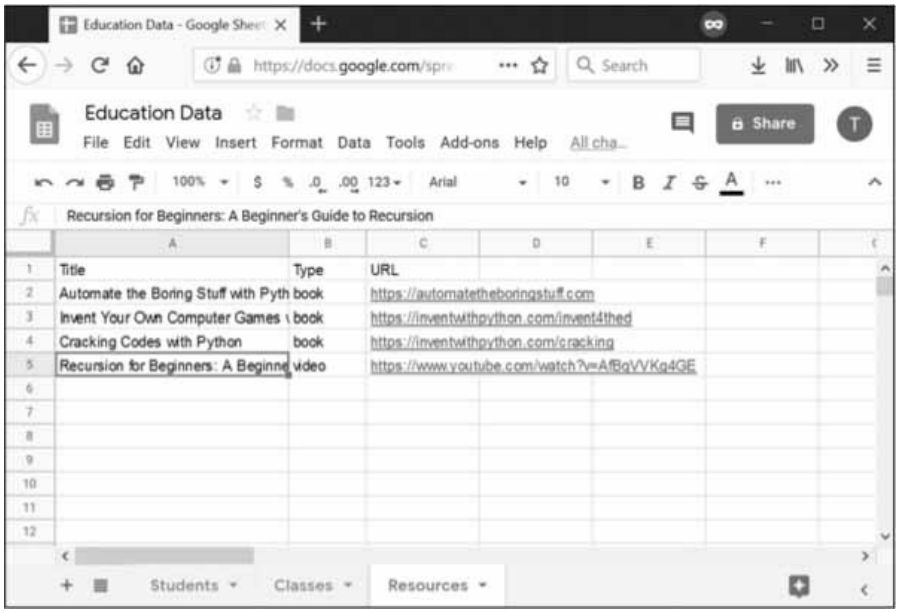
الشكل 70: صفحة الاعتماديات في طرفية المطور على منصة سحابة جوجل

يمكن إنشاء ملف اعتماديات جديد من هذه الصفحة من خلال النقر على زر إنشاء الاعتماديات "Create Credentials" وتحديد خيار معرّف عميل OAuth أو "OAuth Client ID" كما هو موضح في الشكل السابق، ثم حدّد الخيار "أخرى Other" بالنسبة لنوع التطبيق وسمِّ الملف بأيّ اسم تريده. سيُدرَج بعد ذلك ملف الاعتماديات الجديد في الصفحة، ويمكنك النقر على أيقونة التنزيل لتنزيله. سيكون للملف الذي ستنزِّله اسم ملف طويل ومعقد، لذا يجب إعادة تسميته إلى اسم الملف الافتراضي الذي تحاول الوحدة `EZSheets` تحميله وهو `credentials-sheets.json`. يمكنك أيضًا إنشاء ملف اعتماديات جديد من خلال النقر على زر تفعيل واجهة برمجة تطبيقات جداول بيانات جوجل "Enable the Google Sheets API" المذكور في القسم السابق.

14.2 كائنات جدول البيانات Spreadsheet

يمكن أن يحتـوي جـدول البيانـات Spreadsheet في جـداول بيانـات جوجـل على أوراق Sheets متعـددة والتي تُسمَّى أيضًا أوراق عمـل Worksheets، وتحتـوي كـل ورقـة على أعمـدة Columns وصـفوف Rows من القيم.

يوضح الشكل التالي جدول بيانات بعنـوان بيانـات التعليم "Education Data"، والـذي يحتـوي على ثلاث أوراق بعنوان الطلاب "Students" والصفوف "Classes" والموارد "Resources"، ويُسمَّى العمود الأول من كل ورقة A، ويسمى الصف الأول 1:



الشكل 71: جدول بيانات بعنوان "Education Data" مكوَّن من ثلاث أوراق

سيتمثّل معظم عملك في تعديل كائنات الورقة `Sheet`، ولكن يمكنك أيضًا تعديل كائنات جـدول البيانـات `Spreadsheet`، كما سنوضّح في القسم التالي.

14.2.1 إنشاء جداول البيانات وتحميلها وسردها

يمكنك إنشاء كائن `Spreadsheet` جديد من جدول بيانات موجود مسبقًا أو جدول بيانات فـارغ أو جـدول بيانات مرفوع على جداول بيانات جوجـل، حيث يمكن إنشـاء كـائن `Spreadsheet` من جـدول بيانـاتٍ موجـود مسبقًا على جداول بيانات جوجل، ولكن أن تعرف السلسلة النصـية لمعـرّف جـدول البيانـات. يمكن العثـور على المعرّف الفريد لجداول بيانات جوجل في عنوان URL، بعـد الجـزء `spreadsheets/d/` وقبـل الجـزء `/edit`، فمثلًا يوجد جدول البيانات الموضّح في الشكل السابق على عنوان URL الذي هو:

```
https://docs.google.com/spreadsheets/d/1J-Jx6Ne2K_vqI9J2SO-
TAXOFbxx_9tUjwnkPC22LjeU/edit#gid=151537240/
```

وبالتالي يكون معرّفه `1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU`.

> **ملاحظة**: معرّفات جداول البيانات المُستخدَمة في هذا الفصل خاصة بجداول بيانات حساب جوجل الخاص بالكاتب، إذ لن تعمل إذا أدخلتها في صدفتك التفاعلية Interactive Shell، لذا انتقل إلى جداول بيانات جوجل لإنشاء جداول بيانات ضمن حسابك ثم احصل على المعرّفات من شريط العناوين.

مرّر معرّف جدول بياناتك بوصفه سلسلةً نصية إلى الدالة `ezsheets.Spreadsheet()` للحصول على كائن `Spreadsheet` لجدول البيانات الخاص بهذا المعرّف:

```
>>> import ezsheets
>>> ss = ezsheets.Spreadsheet('1J-Jx6Ne2K_vqI9J2SO-
TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss
Spreadsheet(spreadsheetId='1J-Jx6Ne2K_vqI9J2SO-
TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss.title
'Education Data'
```

يمكنك أيضًا الحصول على كائن `Spreadsheet` لجدول بيانات موجود مسبقًا من خلال تمرير عنوان URL الكامل لجدول البيانات إلى تلك الدالة، أو إذا كان هناك جدول بيانات واحد فقط في حسابك على جوجل له العنوان نفسه، فيمكنك تمرير عنوان جدول البيانات بوصفه سلسلة نصية.

يمكنك إنشاء جدول بيانات جديد وفارغ من خلال استدعاء الدالة `ezsheets.createSpreadsheet()` وتمرير سلسلةٍ نصية إليها، حيث تمثل هذه السلسلة النصية عنوان جدول البيانات الجديد. لندخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import ezsheets
>>> ss = ezsheets.createSpreadsheet('Title of My New Spreadsheet')
>>> ss.title
'Title of My New Spreadsheet'
```

يمكنك رفع جدول بيانات إكسل Excel أو أوبن أوفيس OpenOffice أو CSV أو TSV موجود مسبقًا إلى جداول بيانات جوجل من خلال تمرير اسم ملف جدول البيانات إلى الدالة `ezsheets.upload()`. إذًا لندخِل ما يلي في الصدفة التفاعلية مع وضع اسم ملف جدول بياناتك مكان الملف `my_spreadsheet.xlsx`:

```
>>> import ezsheets
>>> ss = ezsheets.upload('my_spreadsheet.xlsx')
>>> ss.title
'my_spreadsheet'
```

يمكنك سرد جداول البيانات الموجودة على حسابك على جوجل من خلال استدعاء الدالة `listSpreadsheets()` التي تعيد قاموسًا Dictionary مفاتيحه هي معرّفات جداول البيانات وقيمه هي عناوين جداول البيانات. إذًا لندخِل ما يلي في الصدفة التفاعلية بعد رفع جدول البيانات:

```
>>> ezsheets.listSpreadsheets()
{'1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU': 'Education Data'}
```

يمكنك بعد الحصول على كائن `Spreadsheet` استخدام سماته وتوابعه للتعامل مع جدول البيانات المُستضاف على جداول بيانات جوجل عبر الإنترنت.

14.2.2 سمات Attributes كائن جدول البيانات Spreadsheet

توجد البيانات الفعلية في الأوراق الخاصة بجدول البيانات، ولكن يحتوي كائن `Spreadsheet` على السمات `title` و `spreadsheetId` و `url` و `sheetTitles` و `sheets` للتعامل مع جدول البيانات. لندخِل ما يلي في الصدفة التفاعلية:

```
>>> import ezsheets
>>> ss = ezsheets.Spreadsheet('1J-Jx6Ne2K_vqI9J2SO-
TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss.title          # عنوان جدول البيانات
'Education Data'
>>> ss.title = 'Class Data' # تغيير العنوان
>>> ss.spreadsheetId # المعرّف الفريد (وهو سمة للقراءة فقط)
'1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU'
>>> ss.url            # عنوان URL الأصلي (وهو سمة للقراءة فقط)
'https://docs.google.com/spreadsheets/d/1J-Jx6Ne2K_vqI9J2SO-
TAXOFbxx_9tUjwnkPC22LjeU/'
>>> ss.sheetTitles      # عناوين جميع كائنات الورقة Sheet
('Students', 'Classes', 'Resources')
>>> ss.sheets           # كائنات الورقة Sheet في جدول البيانات بالترتيب
```

```
(<Sheet sheetId=0, title='Students', rowCount=1000, columnCount=26>,
<Sheet
sheetId=1669384683, title='Classes', rowCount=1000, columnCount=26>,
<Sheet
sheetId=151537240, title='Resources', rowCount=1000, columnCount=26>)
>>> ss[0]               # كائن الورقة الأول في جدول البيانات
<Sheet sheetId=0, title='Students', rowCount=1000, columnCount=26>
>>> ss['Students']      # يمكن أيضًا الوصول إلى الأوراق باستخدام العنوان
<Sheet sheetId=0, title='Students', rowCount=1000, columnCount=26>
>>> del ss[0]           # حذف كائن الورقة الأول في جدول البيانات
>>> ss.sheetTitles      # أصبح كائن الورقة "Students" محذوفًا
('Classes', 'Resources')
```

إذا عدّل شخصٌ ما جدول البيانات من موقع جداول بيانات جوجل، فيمكن للسكربت الخاص بك تحديث كائن `Spreadsheet` ليطابق البيانات الموجودة على الإنترنت من خلال استدعاء التابع `refresh()`:

```
>>> ss.refresh()
```

لن يحدّث هذا التابع سمات كائن `Spreadsheet` فحسب، بل سيحدّث البيانات الموجودة في كائنات `Sheet` التي يحتوي عليها كائن `Spreadsheet`، وستنعكس التغييرات التي تجريها على كائن `Spreadsheet` في جدول البيانات الموجود على الإنترنت ضمن الزمن الحقيقي.

14.2.3 تنزيل ورفع جداول البيانات

يمكنك تنزيل جدول بيانات جوجل بعددٍ من التنسيقات مثل: إكسل وأوبن أوفيس OpenOffice وCSV وTSV وPDF، ويمكنك أيضًا تنزيله كملف مضغوط ZIP يحتوي على ملفات HTML لبيانات جدول البيانات، حيث تحتوي الوحدة `EZSheets` على دوالٍ لكل خيار من هذه الخيارات كما سنوضح فيما يلي:

```
>>> import ezsheets
>>> ss = ezsheets.Spreadsheet('1J-Jx6Ne2K_vqI9J2SO-
TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss.title
'Class Data'
>>> ss.downloadAsExcel() # تنزيل جدول البيانات كملف إكسل
'Class_Data.xlsx'
```

```
>>> ss.downloadAsODS() # تنزيل جدول البيانات كملف أوبن أوفيس
'Class_Data.ods'
>>> ss.downloadAsCSV() # تنزيل الورقة الأولى فقط كملف CSV
'Class_Data.csv'
>>> ss.downloadAsTSV() # تنزيل الورقة الأولى فقط كملف TSV
'Class_Data.tsv'
>>> ss.downloadAsPDF() # تنزيل جدول البيانات كملف PDF
'Class_Data.pdf'
>>> ss.downloadAsHTML() # تنزيل جدول البيانات كملف مضغوط ZIP مؤلَّفٍ من ملفات HTML
'Class_Data.zip'
```

لاحظ أن الملفات التي لها تنسيق CSV و TSV يمكن أن تحتوي على ورقة واحدة فقط، لذلك إذا نزّلت جدول بيانات من جداول بيانات جوجل بهذا التنسيق، فستحصل على الورقة الأولى فقط، ولكن يمكنك تنزيل أوراق أخرى من خلال تغيير السمة `index` الخاصة بكائن `Sheet` إلى القيمة 0.

تعيد جميع دوال التنزيل سلسلة نصية لاسم الملف الذي جرى تنزيله، ويمكنك تحديد اسم ملفك لجدول البيانات عبر تمرير اسم الملف الجديد إلى دالة التنزيل كما يلي، ويجب أن تعيد الدالة اسم الملف المُحدَّث:

```
>>> ss.downloadAsExcel('a_different_filename.xlsx')
'a_different_filename.xlsx'
```

14.2.4 حذف جداول البيانات

يمكننا حذف جدول بيانات من خلال استدعاء التابع `delete()`:

```
>>> import ezsheets
>>> ss = ezsheets.createSpreadsheet('Delete me') # إنشاء جدول البيانات
>>> ezsheets.listSpreadsheets() # التأكد من إنشاء جدول بيانات
{'1aCw2NNJSZblDbhygVv77kPsL3djmgV5zJZllSOZ_mRk': 'Delete me'}
>>> ss.delete() # حذف جدول البيانات
>>> ezsheets.listSpreadsheets()
{}
```

سينقل التابع `delete()` جدول بياناتك إلى مجلد سلة المهملات على جوجل درايف، حيث يمكنك عـرض محتويات مجلد سلة المهملات، ولكن يمكن حذف جدول البيانات نهائيًا من خلال تمريـر القيمـة `True` لوسـيط الكلمة المفتاحية Keyword Argument الذي هو `permanent` كما يلي:

```
>>> ss.delete(permanent=True)
```

لا يُعَد حذف جداول البيانات حذفًا نهائيًا فكرةً جيدة، فمن المستحيل استرداد جدول البيانات الذي أدّى خطأٌ في سكربتك إلى حذفه عن غير قصد. ليس هناك داعٍ للقلق بشـأن تحريـر المسـاحة، إذ تتـوفر مسـاحة تخزينيـة بالجيجابايتات حتى في حسابات جوجل درايف المجانية.

14.3 كائنات الورقة Sheet

يحتوي كائن `Spreadsheet` على كائن `Sheet` واحد أو أكثر، حيث تمثّل كائنـات `Sheet` صـفوف وأعمـدة البيانات الموجودة في الورقة، ويمكنك الوصول إلى هذه الأوراق باستخدام عامل الأقواس المربعة وعـدد صـحيح يمثل الفهرس. تحتوي السمة `sheets` على مجموعة Tuple من كائنـات `Sheet` بـالترتيب الـذي تظهـر بـه في جدول البيانات. يمكنك الوصول إلى كائنات `Sheet` في جدول البيانات عبر إدخال ما يلي في الصدفة التفاعلية:

```
>>> import ezsheets
>>> ss = ezsheets.Spreadsheet('1J-Jx6Ne2K_vqI9J2SO-
TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss.sheets     # كائنات الورقة Sheet في جدول البيانات بالترتيب
(<Sheet sheetId=1669384683, title='Classes', rowCount=1000,
columnCount=26>,
<Sheet sheetId=151537240, title='Resources', rowCount=1000,
columnCount=26>)
>>> ss.sheets[0] # الحصول على كائن الورقة الأول في جدول البيانات
<Sheet sheetId=1669384683, title='Classes', rowCount=1000,
columnCount=26>
>>> ss[0]         # الحصول أيضًا على كائن الورقة الأول في جدول البيانات
<Sheet sheetId=1669384683, title='Classes', rowCount=1000,
columnCount=26>
```

يمكنك أيضًا الحصول على كائن `Sheet` باستخدام عامل الأقواس المربعة وسلسلة نصية تمثّل اسم الورقة، وتحتوي السمة `sheetTitles` الخاصة بكائن `Spreadsheet` على مجموعةٍ تمثّل جميع عناوين الأوراق.

إذًا لندخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> ss.sheetTitles # عناوين جميع كائنات الورقة Sheet في جدول البيانات
('Classes', 'Resources')
>>> ss['Classes'] # يمكن أيضًا الوصول إلى الأوراق باستخدام العنوان
<Sheet sheetId=1669384683, title='Classes', rowCount=1000,
columnCount=26>
```

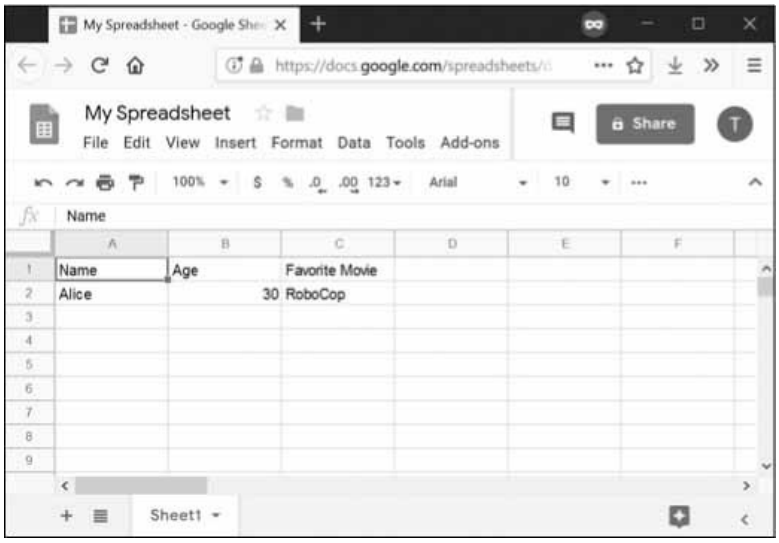
يمكنك بعد الحصول على كائن Sheet قراءة البيانات منه وكتابة البيانات فيه باستخدام توابع كـائن Sheet كما سنوضّح في القسم التالي.

14.3.1 قراءة وكتابة البيانات

تحتوي أوراق عمل جداول بيانات جوجل على أعمدة وصفوف من الخلايا التي تحتـوي على بيانـات كمـا هـو الحال في جداول بيانات إكسل، حيث يمكنك استخدام عامل الأقواس المربعة لقـراءة البيانـات من هـذه الخلايـا وكتابتها فيها. أنشئ مثلًا جدول بيانات جديد وأضِف البيانات إليه من خلال إدخال ما يلي في الصدفة التفاعلية:

```
>>> import ezsheets
>>> ss = ezsheets.createSpreadsheet('My Spreadsheet')
>>> sheet = ss[0] # الحصول على الورقة الأولى في جدول البيانات
>>> sheet.title
'Sheet1'
>>> sheet = ss[0]
>>> sheet['A1'] = 'Name' # ضبط القيمة في الخلية A1
>>> sheet['B1'] = 'Age'
>>> sheet['C1'] = 'Favorite Movie'
>>> sheet['A1'] # قراءة القيمة في الخلية A1
'Name'
>>> sheet['A2'] # تعيد الخلايا الفارغة سلسلة نصية فارغة
''
>>> sheet[2, 1] # العمود 2 والصف 1 هو عنوان الخلية B1 نفسه
'Age'
>>> sheet['A2'] = 'Alice'
>>> sheet['B2'] = 30
>>> sheet['C2'] = 'RoboCop'
```

يجب أن ينتج عن هذه التعليمات جدول بيانات جوجل يشبه الشكل التالي:



الشكل 72: جدول البيانات الذي أنشأناه باستخدام تعليمات المثال السابق

يمكن لعدة مستخدمين تحديث الورقة في الوقت ذاته، لذا يمكنك تحديث البيانات المحلية في كائن `Sheet` من خلال استدعاء التابع `refresh()` الخاص بهذا الكائن:

```
>>> sheet.refresh()
```

تُحمَّل كافة البيانات الموجودة في كائن `Sheet` عند تحميل كائن `Spreadsheet` لأول مرة، وبالتالي يمكن قراءة البيانات مباشرةً، ولكن تتطلب كتابة القيم في جدول البيانات عبر الإنترنت اتصالًا بالشبكة ويمكن أن تستغرق حوالي ثانية واحدة، حيث إذا كان لديك آلاف الخلايا التي تريد تحديثها، فقد يكون تحديثها واحدةً تلو الأخرى بطيئًا جدًا.

14.3.2 عنونة الأعمدة والصفوف

تعمل عنونة الخلايا في جداول بيانات جوجل كما هو الحال في إكسل، ولكن الفرق الوحيد بينهما هو احتواء جداول بيانات جوجل على أعمدة وصفوف تستند إلى القيمة `1`، أي أن العمود أو الصف الأول موجود في الفهرس 1 وليس في الفهرس 0 على عكس فهارس القائمة المستندة إلى القيمة `0` في لغة بايثون.

يمكنك تحويل العنوان الذي تنسيقه سلسلة نصية `'A2'` إلى عنوانٍ تنسيقه مجموعة `(column, row)` (والعكس صحيح) باستخدام الدالة `convertAddress()`.

تحوّل الدالتان `getColumnLetterOf()` و`getColumnNumberOf()` أيضًا عنوان العمود من الحروف إلى الأعداد وبالعكس.

لندخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import ezsheets
>>> ezsheets.convertAddress('A2') # تحويل العناوين...
(1, 2)
>>> ezsheets.convertAddress(1, 2) # ... وتحويلها بالعكس مرة أخرى
'A2'
>>> ezsheets.getColumnLetterOf(2)
'B'
>>> ezsheets.getColumnNumberOf('B')
2
>>> ezsheets.getColumnLetterOf(999)
'ALK'
>>> ezsheets.getColumnNumberOf('ZZZ')
18278
```

تُعَد العناوين التي لها تنسيق السلسلة النصية `'A2'` ملائمةً لكتابة العناوين في شيفرتك المصدرية، وتكون العناوين التي لها تنسيق المجموعة `(column, row)` ملائمةً إذا أردتَ التكرار على مجالٍ من العناوين واحتجتَ صيغةً رقمية للعمود، لذا تُعَد الدوال `convertAddress()` و `getColumnLetterOf()` و`getColumnNumberOf()` مفيدةً عندما تريد التحويل بين هذين التنسيقين.

14.3.3 قراءة وكتابة الأعمدة والصفوف بأكملها

قد تستغرق كتابة البيانات ضمن خلية واحدة في كل مرة وقتًا طويلًا كما ذكرنا سابقًا، ولكن تحتوي وحدة `EZSheets` على توابع خاصة بكائن `Sheet` لقراءة وكتابة الأعمدة والصفوف بأكملها في الوقت ذاته، حيث يقرأ التابعان `getColumn()` و `getRow()` من الأعمدة والصفوف ويكتب التابعان `updateColumn()` و`updateRow()` في الأعمدة والصفوف.

تنشِئ هذه التوابع طلبات إلى خوادم جداول بيانات جوجل لتحديث جدول البيانات، لذا يجب أن تكون متصلًا بالإنترنت.

سنرفع في مثالنا جدول بيانات أسعار المنتجات `produceSales.xlsx` من الفصل السابق إلى جداول بيانات جوجل، حيث تبدو الصفوف الثمانية الأولى كما في الشكل التالي:

	A	B	C	D
1	PRODUCE	COST PER KiloGram	KiloGrams SOLD	TOTAL
2	Potatoes	0.86	21.6	18.58
3	Okra	2.26	38.6	87.24
4	Fava beans	2.69	32.8	88.23
5	Watermelon	0.66	27.3	18.02
6	Garlic	1.19	4.9	5.83
7	Parsnips	2.27	1.1	2.5
8	Asparagus	2.49	37.9	94.37

الشكل 73: الصفوف الثمانية الأولى

يمكنك رفع جدول البيانات `produceSales.xlsx` من خلال إدخال ما يلي في الصدفة التفاعلية:

```
>>> import ezsheets
>>> ss = ezsheets.upload('produceSales.xlsx')
>>> sheet = ss[0]
>>> sheet.getRow(1) # الصف الأول هو الصف 1 وليس الصف 0
['PRODUCE', 'COST PER KiloGram', 'KiloGrams SOLD', 'TOTAL', '', '']
>>> sheet.getRow(2)
['Potatoes', '0.86', '21.6', '18.58', '', '']
>>> columnOne = sheet.getColumn(1)
>>> sheet.getColumn(1)
['PRODUCE', 'Potatoes', 'Okra', 'Fava beans', 'Watermelon', 'Garlic',
--snip--
>>> sheet.getColumn('A') # النتيجة نفسها للتعليمة getColumn(1)
['PRODUCE', 'Potatoes', 'Okra', 'Fava beans', 'Watermelon', 'Garlic',
--snip--
>>> sheet.getRow(3)
['Okra', '2.26', '38.6', '87.24', '', '']
>>> sheet.updateRow(3, ['Pumpkin', '11.50', '20', '230'])
>>> sheet.getRow(3)
['Pumpkin', '11.50', '20', '230', '', '']
```

```
>>> columnOne = sheet.getColumn(1)
>>> for i, value in enumerate(columnOne):
...     # اجعل قائمة بايثون تحتوي على سلاسل نصية بأحرف كبيرة:
...     columnOne[i] = value.upper()
...
>>> sheet.updateColumn(1, columnOne) # تحديث العمود بأكمله في طلب واحد
```

تسترد الدالتان `getRow()` و `getColumn()` البيانات من جميع الخلايا الموجودة في صف أو عمود محدد بوصفها قائمةً من القيم، وتصبح الخلايا الفارغة قيمًا لسلاسل نصية فارغة في القائمة. يمكنك تمرير رقم أو حرف العمود إلى الدالة `getColumn()` لإخبارها باسترداد بيانات عمـودٍ معين، حيث وضّـحنا في المثـال السـابق أن التعلمتين `getColumn(1)` و `getColumn('A')` تعيدان القائمة نفسها.

تكتب الدالتان `updateRow()` و `updateColumn()` فوق البيانات الموجودة في الصـف أو العمـود على التوالي باستخدام قائمة القيم المُمرّرة إليهما، فمثلًا احتوى الصف الثالث في المثال السابق على معلومات حول البامية Okra في البداية، لكن أدّى استدعاء الدالة `updateRow()` إلى وضع بيانـات حـول اليقطين Pumpkin مكانها، ثم استدعينا الدالة `sheet.getRow(3)` مرةً أخرى لعرض القيم الجديدة في الصف الثالث.

لنحدّث بعد ذلك جدول بيانات "produceSales"، إذ يُعَد تحديث خلية واحدة في كل مرة أمرًا بطيئًا إذا كان لديك العديد من الخلايا التي تريد تحديثها، بينما يُعَد الحصول على عمود أو صـف كقائمـة وتحـديث القائمـة ثم تحديث العمود أو الصف بأكمله باستخدام القائمة أسرع بكثير، إذ يمكن إجراء جميع التغييرات في طلبٍ واحد.

يمكن الحصول على كافة الصفوف دفعةً واحدة من خلال استدعاء التابع `getRows()` لإعادة قائمـةٍ بجميـع القوائم، حيث تمثل كل قائمة من القوائم الداخلية الموجودة ضمن القائمة الخارجية صفًا واحدًا من الورقة.

يمكنـك تعـديل هـذه القيم الموجـودة في هيكـل البيانـات لتغيـير اسـم المنتج Produce Name وعـدد الكيلوجرامـات المباعـة Kilograms Sold والتكلفـة الإجماليـة Total لبعض الصـفوف، ثم تمرّرهـا إلى التـابع `updateRows()` من خلال إدخال ما يلي في الصدفة التفاعلية:

```
>>> rows = sheet.getRows() # الحصول على جميع الصفوف في جدول البيانات
>>> rows[0] # فحص القيم الموجودة في الصف الأول
['PRODUCE', 'COST PER KiloGrams', 'KiloGrams SOLD', 'TOTAL', '', '']
>>> rows[1]
['POTATOES', '0.86', '21.6', '18.58', '', '']
>>> rows[1][0] = 'PUMPKIN' # تغيير اسم المنتج
```

```
>>> rows[1]
['PUMPKIN', '0.86', '21.6', '18.58', '', '']
>>> rows[10]
['OKRA', '2.26', '40', '90.4', '', '']
>>> rows[10][2] = '400' # تغيير عدد الكيلوجرامات المباعة
>>> rows[10][3] = '904' # تغيير التكلفة الإجمالية
>>> rows[10]
['OKRA', '2.26', '400', '904', '', '']
>>> sheet.updateRows(rows) # تحديث جدول البيانات عبر الإنترنت بالتغييرات التي أجريناها
```

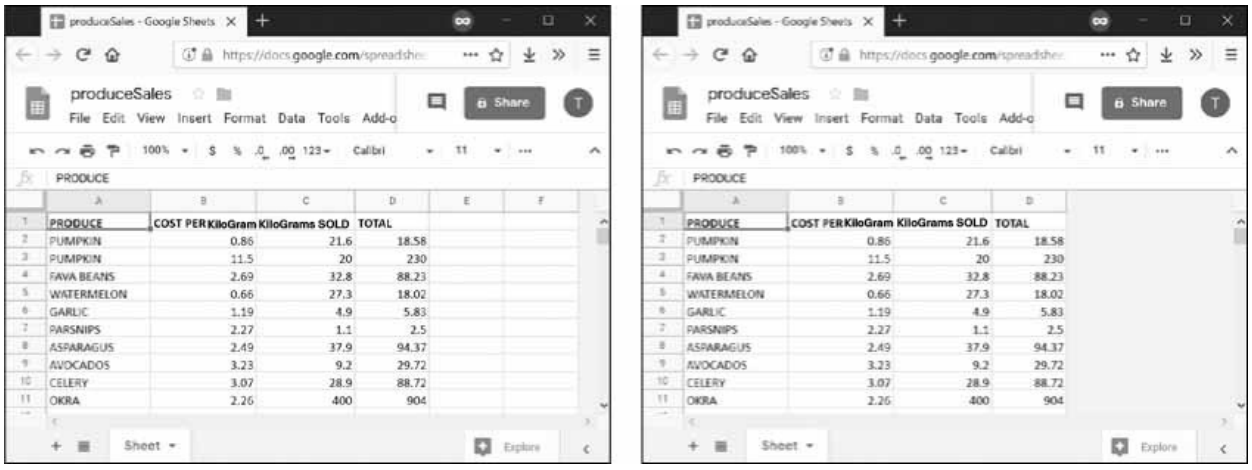
يمكنك تحديث الورقة بأكملها في طلب واحد من خلال تمرير قائمةٍ من القوائم المُعادة من الدالة `getRows()` والمُعدَّلة بالتغييرات التي أجريناها على الصفين 1 و 10 إلى الدالة `updateRows()`.

لاحظ أن الصفوف الموجودة في ورقة جداول بيانات جوجل تحتوي على سلاسل نصية فارغة في نهايتها، لأن الورقة التي رفعناها تحتوي على 6 أعمدة، ولدينا 4 أعمدة فقط من البيانات.

يمكنك قراءة عدد الصفوف والأعمدة في الورقة باستخدام السمتين `rowCount` و `columnCount`، ثم يمكنك تغيير حجم الورقة من خلال ضبط هاتين القيمتين.

```
>>> sheet.rowCount          # عدد الصفوف في الورقة
23758
>>> sheet.columnCount       # عدد الأعمدة في الورقة
6
>>> sheet.columnCount = 4 # تغيير عدد الأعمدة إلى 4
>>> sheet.columnCount       # يصبح الآن عدد الأعمدة في الورقة 4
4
```

يجب أن تحذف التعليمات السابقة العمودين الخامس والسادس من جدول بيانات "produceSales" كما هو موضّح في الشكل التالي:



الشكل 74: الورقة قبل (على اليسار) وبعد (على اليمين) تغيير عدد الأعمدة إلى 4

يمكن أن تحتوي جداول بيانات جوجل على ما يصل إلى 10 ملايين خلية وفقًا لمركز المساعدة في جوجل درايف، ولكن يُفضَّل أن تجعل الأوراق بالحجم الذي تحتاجه فقط من أجل تقليل الوقت الذي يستغرقه تعديل البيانات وتحديثها.

14.3.4 إنشاء وحذف الأوراق

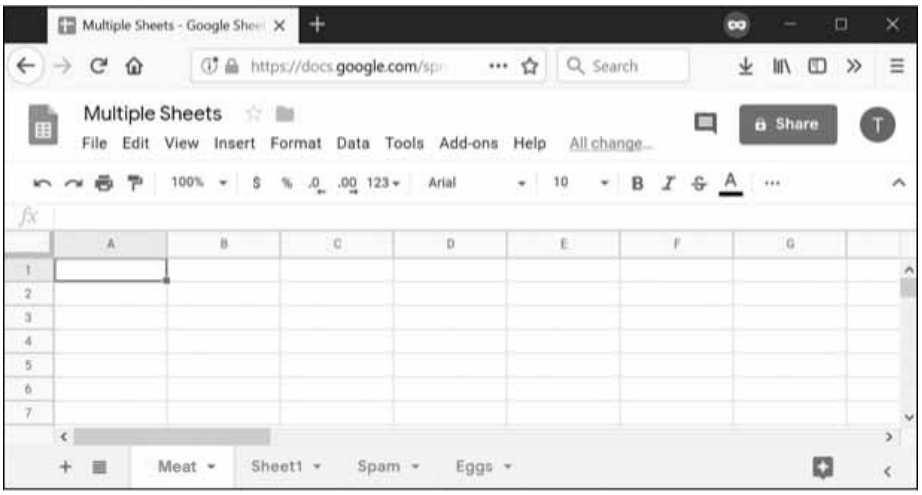
تبدأ جميع جداول بيانات جوجل بورقة واحدة اسمها "Sheet1"، ولكن يمكنك إضافة أوراق إضافية إلى نهاية قائمة الأوراق باستخدام التابع `createSheet()` الذي تمرّر إليه سلسلة نصية لاستخدامها كعنوان للورقة الجديدة، ويمكن للوسيط الثاني الاختياري الخاص بهذا التابع تحديد فهرس العدد الصحيح للورقة الجديدة.

يمكنك إنشاء جدول بيانات ثم إضافة أوراق جديدة إليه من خلال إدخال ما يلي في الصدفة التفاعلية:

```
>>> import ezsheets
>>> ss = ezsheets.createSpreadsheet('Multiple Sheets')
>>> ss.sheetTitles
('Sheet1',)
>>> ss.createSheet('Spam') # إنشاء ورقة جديدة في نهاية قائمة الأوراق
<Sheet sheetId=2032744541, title='Spam', rowCount=1000,
columnCount=26>
>>> ss.createSheet('Eggs') # إنشاء ورقة جديدة أخرى
<Sheet sheetId=417452987, title='Eggs', rowCount=1000, columnCount=26>
>>> ss.sheetTitles
('Sheet1', 'Spam', 'Eggs')
>>> ss.createSheet('Meat', 0) # إنشاء ورقة عند الفهرس 0 في قائمة الأوراق
```

```
<Sheet sheetId=814694991, title='Meat', rowCount=1000, columnCount=26>
>>> ss.sheetTitles
('Meat', 'Sheet1', 'Spam', 'Eggs')
```

تضـيف التعليمـات السـابقة ثلاث أوراق جديـدة إلى جـدول البيانـات هي: "Meat" و"Spam" و"Eggs" بالإضافة إلى الورقة الافتراضية "Sheet1". تُرتَّب الأوراق الموجودة في جدول البيانات، وتضاف الأوراق الجديـدة إلى نهاية القائمة إن لم تمرِّر وسيطًا ثانيًا إلى الدالة `createSheet()`، حيث يحدّد هذا الوسيط فهرس الورقـة. أنشأنا في المثال السابق الورقة الـتي عنوانهـا "Meat" في الفهـرس 0، ممـا يجعـل الورقـة "Meat" هي الورقـة الأولى في جدول البيانات وإزاحة الأوراق الثلاث الأخرى بمقدار موضعٍ واحـد، ويشـبه ذلـك سـلوك تـابع القائمـة `insert()`. يمكنك رؤية الأوراق الجديدة على التبويبات الموجودة أسـفل الشاشـة، تمامًـا كمـا هـو موضَّـح في الشكل التالي:



الشكل 75: جدول بيانات الأوراق المتعددة Multiple Sheets بعد إضافة أوراق أخرى

يحذف التابع `delete()` الخاص بالكائن `Sheet` ورقةً من جدول البيانات، ولكن إذا أردتَ الاحتفاظ بالورقة مع حذف البيانات الموجودة فيها، فاستدعِ التابع `clear()` لمسح جميع الخلايا وجعل هذه الورقة ورقـةً فارغـة. إذا لندخِل ما يلي في الصدفة التفاعلية:

```
>>> ss.sheetTitles
('Meat', 'Sheet1', 'Spam', 'Eggs')
>>> ss[0].delete()       # حذف الورقة الموجودة في الفهرس 0 أي الورقة "Meat"
>>> ss.sheetTitles
('Sheet1', 'Spam', 'Eggs')
>>> ss['Spam'].delete() # حذف الورقة "Spam"
>>> ss.sheetTitles
```

```
('Sheet1', 'Eggs')
>>> sheet = ss['Eggs']  # إسناد الورقة "Eggs" إلى متغير
>>> sheet.delete()      # حذف الورقة "Eggs"
>>> ss.sheetTitles
('Sheet1',)
>>> ss[0].clear()       # مسح جميع الخلايا الموجودة في الورقة "Sheet1"
>>> ss.sheetTitles      # الورقة "Sheet1" فارغة ولكنها لا تزال موجودة
('Sheet1',)
```

يكون حذف الأوراق حذفًا نهائيًا، إذ لا توجد طريقة لاستعادة البيانات، ولكن يمكنك إنشاء نسخة احتياطية من الأوراق من خلال نسخها إلى جدول بيانات آخر باستخدام التابع `copyTo()` كما سنوضّح في القسم التالي.

14.3.5 نسخ الأوراق

يحتوي كل كائن `Spreadsheet` على قائمةٍ مرتبة من كائنات `Sheet` الموجودة ضمنه، حيث يمكنك استخدام هذه القائمة لإعادة ترتيب الأوراق (كما وضّحنا في القسم السابق) أو نسخها إلى جداول بيانات أخرى، إذ يمكن نسخ كائن `Sheet` إلى كائن `Spreadsheet` آخر من خلال استدعاء التابع `copyTo()` الذي نمرّر إليه كائن `Spreadsheet` الهدف كوسيط. لندخِل ما يلي في الصدفة التفاعلية لإنشاء جدولي بيانات ونسخ بيانات جدول البيانات الأول إلى الورقة الأخرى:

```
>>> import ezsheets
>>> ss1 = ezsheets.createSpreadsheet('First Spreadsheet')
>>> ss2 = ezsheets.createSpreadsheet('Second Spreadsheet')
>>> ss1[0]
<Sheet sheetId=0, title='Sheet1', rowCount=1000, columnCount=26>
>>> ss1[0].updateRow(1, ['Some', 'data', 'in', 'the', 'first', 'row'])
>>> ss1[0].copyTo(ss2) # نسخ الورقة Sheet1 الخاصة بجدول البيانات ss1 إلى جدول البيانات ss2
>>> ss2.sheetTitles    # سيحتوي جدول البيانات ss2 على نسخة من الورقة Sheet1 الخاصة بجدول
ss1 Sheet1 البيانات
('Sheet1', 'Copy of Sheet1')
```

لاحظ تسمية الورقة المنسوخة بالاسم `Copy of Sheet1`، لأن جدول البيانات الهدف (`ss2` في المثال السابق) يحتوي مسبقًا على ورقة بالاسم `Sheet1`. تظهر الأوراق المنسوخة في نهاية قائمة أوراق جدول البيانات الهدف، ولكن يمكنك تغيير السمة `index` لإعادة ترتيبها في جدول البيانات الجديد.

14.4 التعامل مع الحصص Quotas في جداول بيانات جوجل

تُعَد جداول بيانات جوجل متاحةً عبر الإنترنت، لذا من السهل مشاركة الأوراق بين عدة مستخدمين يمكنهم جميعًا الوصول إلى الأوراق في وقتٍ واحد، ولكن سيؤدّي ذلك إلى أن تكون قراءة الأوراق وتحديثها أبطأ من قراءة وتحديث ملفات إكسل المخزَّنة محليًا على قرص حاسوبك الصلب. تفرض جداول بيانات جوجل أيضًا قيودًا على عدد عمليات القراءة والكتابة التي يمكنك إجراؤها.

يُقيِّد مستخدمو جداول بيانات جوجل بإنشاء 250 جدول بيانات جديد يوميًا، ويمكن لحسابات جوجل المجانية إجراء 100 طلب قراءة و100 طلب كتابة في كل 100 ثانية وفقًا لإرشادات مطوري جوجل، إذ ستؤدي محاولة تجاوز هذه الحصة إلى رفع الاستثناء `googleapiclient.errors.HttpError` أو "Quota exceeded for quota group" الذي يمثّل تجاوز الحصة المتاحة، حيث تلتقط الوحدة `EZSheets` تلقائيًا هذا الاستثناء وتعيد محاولة الطلب. إذا حدث ذلك، فستستغرق استدعاءات الدوال لقراءة البيانات أو كتابتها عدة ثوانٍ أو حتى دقيقة أو دقيقتين قبل أن تعيد شيئًا ما، وإذا استمر الطلب في الفشل، وهو أمرٌ ممكن إذا أجرى سكربتٌ آخر يمتلك الاعتماديات نفسها طلباتٍ أيضًا، فستعيد الوحدة `EZSheets` رفعَ هذا الاستثناء.

يؤدي ذلك إلى أنه قد تستغرق استدعاءات توابع الوحدة `EZSheets` عدة ثوانٍ قبل أن تعيد شيئًا ما. إذا أردتَ عرضَ حجم استخدامك لواجهة برمجة التطبيقات أو زيادةَ حصتك، فانتقل إلى صفحة IAM & Admin Quotas للتعرف على كيفية الدفع مقابل زيادة حجم الاستخدام. إذا أردتَ التعامل مع استثناءات `HttpError` بنفسك، فيمكنك ضبط `ezsheets.IGNORE_QUOTA` على القيمة `True`، وسترفع توابع الوحدة `EZSheets` هذه الاستثناءات عندما تواجهها.

14.5 أسئلة للتدريب

1. ما الملفات الثلاثة التي تحتاجها حتى تتمكّن وحدة `EZSheets` من الوصول إلى جداول بيانات جوجل؟
2. ما هما نوعا الكائنات الموجودة في الوحدة `EZSheets`؟
3. كيف يمكنك إنشاء ملف إكسل من جدول بيانات جوجل؟
4. كيف يمكنك إنشاء جدول بيانات جوجل من ملف إكسل؟
5. إذا احتوى المتغير `ss` على كائن `Spreadsheet`، فما هي الشيفرة البرمجية التي ستقرأ البيانات من الخلية B2 في ورقة عنوانها "Students"؟
6. كيف يمكنك معرفة حروف الأعمدة المقابلة للعمود 999؟
7. كيف يمكنك معرفة عدد الصفوف والأعمدة الموجودة في الورقة؟
8. كيف يمكنك حذف جدول بيانات، وهل هذا الحذف نهائي؟

9. ما هي الدوال التي ستنشئ كائن `Spreadsheet` جديد وكائن `Sheet` جديد؟

10. ماذا سيحدث إذا تجاوزتَ حصة حسابك على جوجل من خلال إجراء طلبات القراءة والكتابة المتكررة باستخدام الوحدة `EZSheets`؟

14.6 مشاريع للتدريب

حاول كتابة البرامج التي تؤدي المهام التي سنوضّحها فيما يلي لكسب خبرة عملية أكبر.

14.6.1 برنامج لتنزيل بيانات نماذج جوجل Google Forms

تتيح لك نماذج جوجل إنشاء نماذج بسيطة عبر الإنترنت تسهّل جمع المعلومات من الأشخاص، حيث تُخزَّن المعلومات التي يدخلها هؤلاء الأشخاص في النموذج ضمن جداول بيانات جوجل. جرّب كتابة برنامجٍ يمكنه تنزيل معلومات النموذج التي أرسلها المستخدمون تلقائيًا، لذا انتقل إلى نماذج جوجل وأنشئ نموذجًا جديدًا، حيث سيكون هذا النموذج فارغًا، ثم أضِف الحقول إلى النموذج الذي يطلب من المستخدم اسمه وعنوان بريده الإلكتروني، ثم انقر على زر "إرسال Send" في الجزء العلوي الأيمن للحصول على رابط لنموذجك الجديد مثل الرابط `https://goo.gl/forms/QZsq5sC2Qe4fYO592/`، وحاول إدخال بعض الأمثلة على الردود في هذا النموذج.

انقر على الزر الأخضر "Create Spreadsheet" في تبويب "الردود Responses" في نموذجك لإنشاء جدول بيانات جوجل الذي سيحتوي على الردود التي يرسلها المستخدمون. يُفترَض أن تشاهد إجاباتك في الصفوف الأولى من جدول البيانات. اكتب بعد ذلك سكربت بايثون مع استخدام الوحدة EZSheets لجمع قائمة بعناوين البريد الإلكتروني في جدول البيانات.

14.6.2 برنامج لتحويل جداول البيانات إلى تنسيقات أخرى

يمكنك استخدام جداول بيانات جوجل لتحويل ملف جدول بيانات إلى تنسيقات أخرى، لذا جرّب كتابة سكربت يمرر ملفًا مُرسَلًا إلى الدالة `upload()`.

نزّل جدول البيانات بعد رفعه على جداول بيانات جوجل باستخدام الدوال `downloadAsExcel()` و`downloadAsODS()` وغيرها من الدوال المماثلة لإنشاء نسخة من جدول البيانات بتنسيقات أخرى.

14.6.3 برنامج للعثور على الأخطاء في جدول البيانات

لنفترض أن لدينا جدول بيانات يحتوي على إجمالي عدد حبات الفاصولياء مرفوع على جداول بيانات جوجل، حيث يكون جدول البيانات قابلًا للعرض، ولكنه غير قابل للتحرير، ويمكنك الاطلاع عليه في متصفحك والحصول عليه باستخدام الشيفرة التالية:

```
>>> import ezsheets
>>> ss = ezsheets.Spreadsheet('1jDZEdvSIh4TmZxccyyOZXrH-
ELlrwq8_YYiZrEOB4jg')
```

أعمدة الورقة الأولى في جدول البيانات هي عدد حبات الفاصولياء في الجرة "Beans per Jar" وعدد الجِرار "Jars" وعدد حبات الفاصولياء الكلي "Total Beans"، حيث ينتج العمود "Total Beans" من ضرب الأعداد الموجودة في العمودين "Beans per Jar" و "Jars"، ولكن يوجد خطأ في أحد الصفوف البالغ عددها 15000 صفًا في هذه الورقة. يُعَد ذلك عددًا كبيرًا جدًا من الصفوف التي لا يمكن التحقق منها يدويًا، ولكن يمكنك كتابة سكربت يتحقق من العمود "Total Beans".

يمكنك الوصول إلى الخلايا الفردية في صف باستخدام `ss[0].getRow(rowNum)`، حيث `ss` هو كائن `Spreadsheet` و `rowNum` هو رقم الصف، وتذكّر أن أرقام الصفوف في جداول بيانات جوجل تبدأ من العدد 1 وليس من 0. ستكون قيم الخلايا سلاسلًا نصية، لذا يجب تحويلها إلى أعداد صحيحة حتى يتمكّن برنامجك من العمل معها.

يُقيَّم التعبير الآتي:

```
int(ss[0].getRow(2)[0])*int(ss[0].getRow(2)[1]) == int(ss[0].getRow(2)
[2])
```

على القيمة `True` إذا احتوى الصف على القيمة الإجمالية الصحيحة، لذا ضع هذا الشيفرة البرمجية في حلقة لتحديد الصف الموجود في الورقة الذي يحتوي على القيمة الإجمالية غير الصحيحة.

14.7 الخلاصة

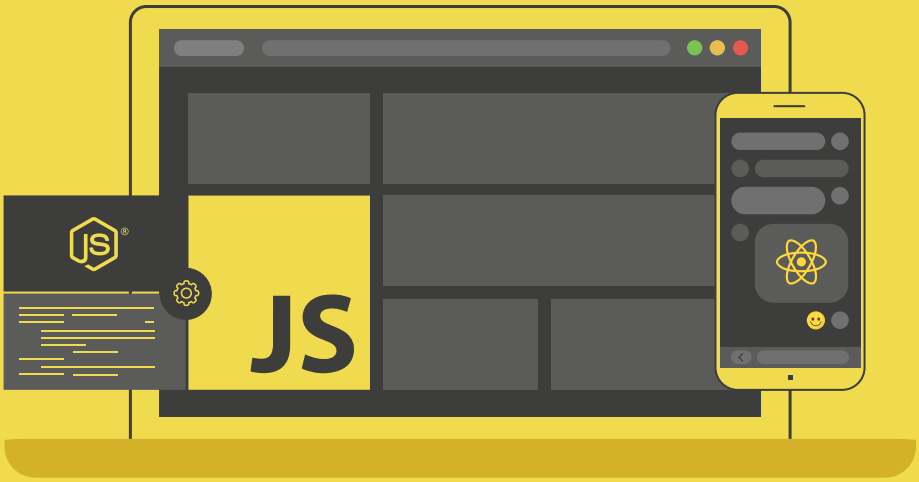
يُعَد تطبيق جداول بيانات جوجل تطبيقًا شائعًا لجداول البيانات عبر الإنترنت التي تعمل في متصفحك. يمكنك تنزيل جداول البيانات وإنشاؤها وقراءتها وتعديلها باستخدام الوحدة الخارجية `EZSheets` التي تمثّل جداول البيانات بوصفها كائنات `Spreadsheet` التي تحتوي على قائمة مرتبة من كائنات `Sheet`، وتحتوي كل ورقة على أعمدة وصفوف من البيانات التي يمكنك قراءتها وتحديثها بطرق متعددة.

تسهّل جداول بيانات جوجل مشاركة البيانات وتعديلها بصورة جماعية، ولكن عيبها الرئيسي هو السرعة، إذ يجب عليك تحديث جداول البيانات باستخدام طلبات الويب، مما يؤدي إلى أن يستغرق التنفيذ بضع ثوانٍ، ولكن لن يؤثر هذا القيد على سكربتات بايثون التي تستخدم وحدة `EZSheets` بالنسبة لمعظم الأغراض.

تحدّ جداول بيانات جوجل أيضًا من عدد المرات التي يمكنك فيها إجراء التغييرات.

> **ملاحظة**: يمكنك الحصول على التوثيق الكامل لميزات الوحدة EZSheet من موقعها الرسمي.

دورة تطوير التطبيقات
باستخدام لغة JavaScript



احترف تطوير التطبيقات بلغة جافا سكريبت
انطلاقًا من أبسط المفاهيم وحتى بناء تطبيقات حقيقية

التحق بالدورة الآن

15. العمل بمستندات PDF وWordg بايثون

تُعَد مستندات بي دي إف PDF ومستندات وورد Word ملفات ثنائية، مما يجعلها أكثر تعقيدًا من الملفات النصية العادية، إذ تخزِّن الكثير من المعلومات المتعلقة بالخطوط والألوان وتخطيط الصفحات بالإضافة إلى النصوص. إذا أدرتَ أن تقرأ برامجك أو تكتبها في ملفات PDF أو مستندات وورد، فستحتاج إلى تطبيق أكثر من مجرد تمرير أسماء الملفات إلى الدالة `open()`، لذا توجد وحدات بايثون Python التي تسهّل عليك التفاعل مع ملفات PDF ومستندات وورد مثل الوحدتين `PyPDF2` و `Python-Docx` اللتين سنوضّحهما في هذا الفصل.

15.1 مستندات PDF

يرمز الاختصار PDF إلى صيغة المستندات المنقولة Portable Document Format التي تستخدم امتداد الملف `.pdf`. تدعم ملفات PDF العديد من الميزات، ولكن سنركز في هذا الفصل على المهمتين اللتين ستفعلهما باستخدام هذه الملفات في أغلب الأحيان وهما: قراءة محتوى النصوص من ملفات PDF وإنشاء ملفات PDF جديدة من مستندات موجودة مسبقًا.

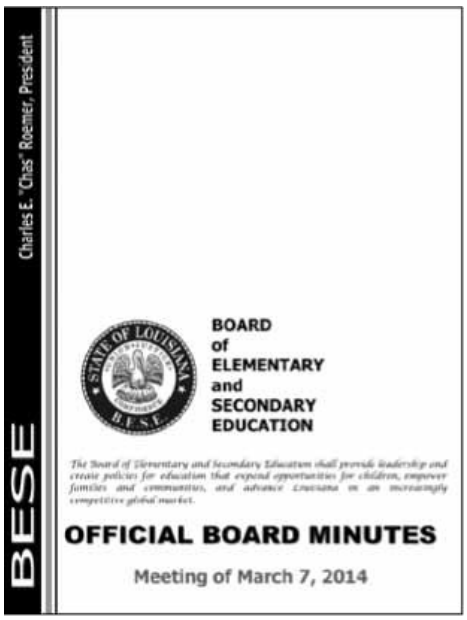
سنستخدم الوحدة `PyPDF2` ذات الإصدار 1.26.0 للعمل مع ملفات PDF، لذا من المهم أن تثبّت هذا الإصدار لأن الإصدارات اللاحقة من وحدة `PyPDF2` قد تكون غير متوافقة مع شيفرتنا البرمجية، إذًا شغّل الأمر `pip install --user PyPDF2==1.26.0` من سطر الأوامر لتثبيت هذه الوحدة، ولاحظ أن اسم الوحدة حساس لحالة الحروف، لذا تأكد من أن الحرف y صغير والحروف الأخرى كبيرة. اتبع الإرشادات الخاصة بتثبيت الوحدات الخارجية التي سنوضّحها في الملحق 1.

إذا جرى تثبيت هذه الوحدة بصورة صحيحة، فيُفترض ألّا يؤدي تشغيل الأمر `import PyPDF2` في الصدفة التفاعلية Interactive Shell إلى عرض أيّ أخطاء.

ملاحظة: تُعَد ملفات PDF رائعة لتجهيز النصوص بحيث تسهل طباعتها وقراءتها، ولكن ليس من السهل على البرمجيات تحليلها إلى نصوص عادية، لذلك قد ترتكب الوحدة PyPDF2 أخطاءً عند استخراج النص من ملف PDF، وقد لا تتمكّن من فتح بعض ملفات PDF، ولا يوجد الكثير لفعله لحل هذه المشكلة، إذ قد تكون وحدة PyPDF2 ببساطة غير قادرة على العمل مع بعض ملفات PDF، ولكننا لم نجد بعد أيّ ملفات PDF لا يمكن فتحها باستخدام وحدة PyPDF2.

15.1.1 استخراج النص من ملفات PDF

لا تمتلك وحدة PyPDF2 طريقةً لاستخراج الصور أو المخططات أو الوسائط الأخرى من مستندات PDF، ولكن يمكنها استخراج النص وإعادته بوصفه سلسلة بايثون. سنستخدم في مثالنا مستند PDF الموضّح في الشكل التالي للبدء في تعلم كيفية عمل وحدة PyPDF2:



الشكل 76: صفحة PDF التي سنستخرج النص منها

نزّل ملف PDF الموضّح في الشكل السابق وأدخِل ما يلي في الصدفة التفاعلية:

```
>>> import PyPDF2
>>> pdfFileObj = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
❶ >>> pdfReader.numPages
19
```

```
❷ >>> pageObj = pdfReader.getPage(0)

❸ >>> pageObj.extractText()

 'OOFFFFIICCIIAALL  BBOOAARRDD  MMIINNUUTTEESS   Meeting of March 7,

   2015        \n     The Board of Elementary and Secondary Education
shall

 provide leadership and create policies for education that expand
opportunities

 for children, empower families and communities, and advance Louisiana
in an

 increasingly competitive global market. BOARD  of ELEMENTARY and
SECONDARY

 EDUCATION  '

   >>> pdfFileObj.close()
```

نستورد أولًا وحدة `PyPDF2`، ثم نفتح الملف `meetingminutes.pdf` للقراءة في الوضع الثنائي ونخزّنه في المتغير `pdfFileObj`. يمكن الحصول على كائن `PdfFileReader` الذي يمثل ملف PDF من خلال استدعاء الدالة `PyPDF2.PdfFileReader()` وتمرير المتغير `pdfFileObj` إليها، ثم خزّن هذا الكائن في المتغير `pdfReader`.

يُخزَّن إجمالي عدد صفحات المستند في السمة Attribute التي هي `numPages` الخاصة بالكائن `PdfFileReader` ❶، ويحتوي ملف PDF في مثالنا على 19 صفحة، ولكن نريد استخراج النص من الصفحة الأولى فقط من خلال الحصول على كائن `Page` من كائن `PdfFileReader`، حيث يمثل كائن `Page` صفحة واحدة من ملف PDF. يمكنك الحصول على كائن `Page` من خلال استدعاء التابع `getPage()` ❷ الخاص بكائن `PdfFileReader` وتمرير رقم الصفحة التي تريدها إليه، ورقم الصفحة هو 0 في مثالنا.

تستخدم الوحدة `PyPDF2` فهرسًا مستنِدًا إلى القيمة `0` للحصول على الصفحات، فالصفحة الأولى هي الصفحة 0، والصفحة الثانية هي الصفحة 1 وإلخ، إذ تُستخدَم هذه الطريقة دائمًا حتى لو كانت الصفحات مرقَّمة بطريقة مختلفة في المستند. لنفترض مثلًا أن لديك ملف PDF يمثّل مقطعًا من تقرير أطول، ويتألف هذا المقطع من ثلاث صفحات، وأرقام الصفحات هي 42 و 43 و 44. يمكن الحصول على الصفحة الأولى من هذا المستند من خلال استدعاء التابع `pdfReader.getPage(0)` وليس من خلال استدعاء `getPage(42)` أو `getPage(1)`.

نحصل على كائن `Page`، ثم نستدعي التابع `extractText()` الخاص بهذا الكائن لإعادة سلسلة نصية تمثل النص الموجود في الصفحة ❸. لاحظ أن استخراج النص ليس مثاليًا، فالنص "Charles E. “Chas”"

"Roemer, President من ملف PDF غير موجود في السلسلة النصية التي يعيدها التابع `extractText()`، وتكون المسافات غير مفعّلة في بعض الأحيان، مع ذلك قد يكون هذا المحتوى التقريبي لنص ملف PDF كافيًا جدًا لبرنامجك.

15.1.2 فك تشفير ملفات PDF

تحتوي بعض مستندات PDF على ميزة تشفير تمنع قراءتها حتى يضع الشخص الذي يفتح المستند كلمة المرور. لندخِل ما يلي في الصدفة التفاعلية مع ملف PDF الذي نزلته، وهذا الملف مُشفَّر من خلال استخدام كلمة المرور `rosebud`:

```
>>> import PyPDF2
>>> pdfReader = PyPDF2.PdfFileReader(open('encrypted.pdf', 'rb'))
❶ >>> pdfReader.isEncrypted
True
>>> pdfReader.getPage(0)
❷ Traceback (most recent call last):
  File "<pyshell#173>", line 1, in <module>
    pdfReader.getPage()
  --snip--
  File "C:\Python34\lib\site-packages\PyPDF2\pdf.py", line 1173, in
getObject
    raise utils.PdfReadError("file has not been decrypted")
PyPDF2.utils.PdfReadError: file has not been decrypted
>>> pdfReader = PyPDF2.PdfFileReader(open('encrypted.pdf', 'rb'))
❸ >>> pdfReader.decrypt('rosebud')
1
>>> pageObj = pdfReader.getPage(0)
```

تحتوي جميع كائنات `PdfFileReader` على السمة `isEncrypted` التي تكون قيمتها `True` إذا كان ملف PDF مُشفرًا، وتكون قيمتها `False` إن لم يكن ملف PDF مُشفَّرًا ❶، وستؤدي أيّ محاولة لاستدعاء دالةٍ تقرأ الملف قبل فك تشفيره باستخدام كلمة المرور الصحيحة إلى حدوث خطأ ❷.

> **ملاحظة**: يوجد خطأ في الإصدار 1.26.0 من وحدة PyPDF2، حيث يؤدي استدعاء التابع `getPage()` لملف PDF مشفَّر قبل استدعاء الدالة `decrypt()` لهذا الملف إلى فشل استدعاءات التابع `getPage()` المستقبلية مع ظهور الخطأ `IndexError: list index out of range`، ولذلك أعاد المثال السابق فتح الملف باستخدام كائن `PdfFileReader` جديد.

يمكن قراءة ملف PDF مشفَّر من خلال استدعاء الدالة `decrypt()` وتمرير كلمة المرور بوصفها سلسلة نصية إليه ❸، وسترى أن استدعاء التابع `getPage()` لم يعُد يسبّب خطأً بعد استدعاء الدالة `decrypt()` مع كلمة المرور الصحيحة، بينما إذا أعطيتَ كلمة مرور خطأ، فستعيد الدالة `decrypt()` القيمة 0 وسيفشل التابع `getPage()`. لاحظ أن التابع `decrypt()` يفك تشفير الكائن `PdfFileReader` فقط وليس ملف PDF الفعلي، إذ يبقى الملف الموجود على قرص حاسوبك الصلب مشفَّرًا بعد انتهاء البرنامج، وبالتالي يجب أن يستدعي برنامجُك التابعَ `decrypt()` عند تشغيله مرة أخرى.

15.1.3 إنشاء ملفات PDF

يمكن للكائن `PdfFileWriter` الخاص بالوحدة `PyPDF2` إنشاء ملفات PDF جديدة، ولكن لا تستطيع وحدة `PyPDF2` كتابة نص عشوائي في ملف PDF كما تفعل شيفرة بايثون مع ملفات النصوص العادية، لذا تقتصر إمكانات كتابة ملفات PDF في وحدة `PyPDF2` على نسخ الصفحات من ملفات PDF الأخرى وتدوير الصفحات ودمجها وتشفير الملفات.

لا تسمح وحدة `PyPDF2` بتعديل ملف PDF مباشرةً، لذا يجب إنشاء ملف PDF جديد ثم نسخ المحتوى من مستند موجود مسبقًا. ستتبع الأمثلة الواردة في هذا القسم النهج العام التالي:

1. فتح ملفٍ أو أكثر من ملفات PDF الموجودة مسبقًا (ملفات PDF المصدر) في كائنات `PdfFileReader`.
2. إنشاء كائن `PdfFileWriter` جديد.
3. نسخ الصفحات من كائنات `PdfFileReader` إلى كائن `PdfFileWriter`.
4. استخدام كائن `PdfFileWriter` لكتابة ملف PDF الناتج.

يؤدي إنشاء كائن `PdfFileWriter` إلى إنشاء قيمةٍ تمثّل مستند PDF في شيفرة بايثون فقط دون إنشاء ملف PDF الفعلي، ولذلك يجب استدعاء التابع `write()` الخاص بهذا الكائن. يأخذ هذا التابع كائن `File` عادي مفتوح في وضع الكتابة الثنائي، حيث يمكنك الحصول على كائن `File` من خلال استدعاء الدالة `open()` الخاصة بلغة بايثون مع وسيطين هما: السلسلة النصية التي تريد أن تمثّل اسم ملف PDF والوسيط `'wb'` الذي يشير إلى أنه يجب فتح الملف في وضع الكتابة الثنائي. لا تقلق إذا كان ذلك مربكًا بعض الشيء، حيث سنرى كيفية ذلك في الأمثلة البرمجية التالية.

15.1.4 نسخ الصفحات

يمكنك استخدام وحدة PyPDF2 لنسخ الصفحات من مستند PDF إلى آخر، مما يتيح لك دمج ملفات PDF متعددة أو قص الصفحات غير المرغوب فيها أو إعادة ترتيب الصفحات.

نزّل الملفين meetingminutes.pdf و meetingminutes2.pdf وضعهما في مجلد العمل الحالي، ثم أدخِل ما يلي في الصدفة التفاعلية:

```
>>> import PyPDF2
>>> pdf1File = open('meetingminutes.pdf', 'rb')
>>> pdf2File = open('meetingminutes2.pdf', 'rb')
❶ >>> pdf1Reader = PyPDF2.PdfFileReader(pdf1File)
❷ >>> pdf2Reader = PyPDF2.PdfFileReader(pdf2File)
❸ >>> pdfWriter = PyPDF2.PdfFileWriter()
>>> for pageNum in range(pdf1Reader.numPages):
    ❹ pageObj = pdf1Reader.getPage(pageNum)
    ❺ pdfWriter.addPage(pageObj)
>>> for pageNum in range(pdf2Reader.numPages):
    ❹ pageObj = pdf2Reader.getPage(pageNum)
    ❺ pdfWriter.addPage(pageObj)
❻ >>> pdfOutputFile = open('combinedminutes.pdf', 'wb')
>>> pdfWriter.write(pdfOutputFile)
>>> pdfOutputFile.close()
>>> pdf1File.close()
>>> pdf2File.close()
```

افتح ملفي PDF في وضع القراءة الثنائي وخزّن كائني File الناتجين في المتغيرين pdf1File وpdf2File، ثم استدعِ الدالة PyPDF2.PdfFileReader() ومرّر المتغير pdf1File إليها للحصول على كائن PdfFileReader للملف meetingminutes.pdf ❶، ثم استدعيها مرةً أخرى ومرّر المتغير

`pdf2File` إليها للحصول على كائن `PdfFileReader` للملف `meetingminutes2.pdf` ❷، ثم أنشئ كائن `PdfFileWriter` جديد، والذي يمثل مستند PDF فارغ ❸.

انسخ بعد ذلك جميع الصفحات من ملفي PDF المصدر وأضِفها إلى كائن `PdfFileWriter`، واحصل على كائن `Page` من خلال استدعاء التابع `getPage()` لكائن `PdfFileReader` ❹، ثم مرّر كائن `Page` إلى التابع `addPage()` الخاص بالكائن `PdfFileReader` ❺. نفّذ هذه الخطوات أولًا للمتغير `pdf1Reader` ثم للمتغير `pdf2Reader` مرة أخرى، ثم اكتب ملف PDF جديد اسمه `combinedminutes.pdf` عند الانتهاء من نسخ الصفحات من خلال تمرير كائن `File` إلى التابع `write()` الخاص بالكائن `PdfFileWriter` ❻.

> **ملاحظة**: لا يمكن لوحدة PyPDF2 إدراج صفحات في منتصف كائن `PdfFileWriter`، إذ يضيف التابع `addPage()` الصفحات إلى نهاية الملف فقط.

لقد أنشأنا ملف PDF جديد يدمج صفحاتٍ من كلا الملفين `meetingminutes.pdf` و`meetingminutes2.pdf` في مستند واحد. تذكّر أنه يجب فتح كائن `File` الذي مرّرناه إلى الدالة `PyPDF2.PdfFileReader()` في وضع القراءة الثنائي من خلال تمرير الوسيط `'rb'` بوصفه وسيطًا ثانيًا للدالة `open()`، ويجب فتح كائن `File` الذي مرّرناه إلى الدالة `PyPDF2.PdfFileReader()` في وضع الكتابة الثنائي باستخدام الوسيط `'wb'`.

15.1.5 تدوير الصفحات

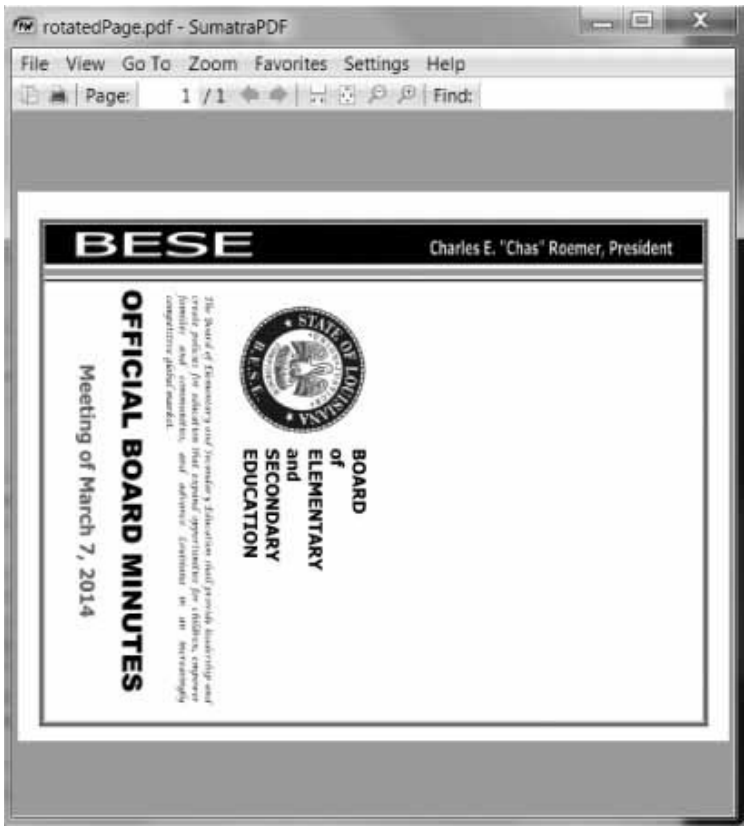
يمكن تدوير صفحات ملف PDF بمقدار مضاعفات 90 درجة باستخدام التوابع `rotateClockwise()` و`rotateCounterClockwise()`. لنمرّر أحد الأعداد الصحيحة 90 أو 180 أو 270 إلى هذه التوابع، ولندخِل ما يلي في الصدفة التفاعلية مع ملف `meetingminutes.pdf` الموجود في مجلد العمل الحالي:

```
>>> import PyPDF2
>>> minutesFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
❶ >>> page = pdfReader.getPage(0)
❷ >>> page.rotateClockwise(90)
{'/Contents': [IndirectObject(961, 0), IndirectObject(962, 0),
--snip--
}
```

```
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> pdfWriter.addPage(page)
❸ >>> resultPdfFile = open('rotatedPage.pdf', 'wb')
>>> pdfWriter.write(resultPdfFile)
>>> resultPdfFile.close()
>>> minutesFile.close()
```

استخدمنا التابع `getPage(0)` لتحديد الصفحة الأولى من ملف PDF ❶، بعد ذلك استدعينا التابع `rotateClockwise(90)` لتلك الصفحة ❷، ثم كتبنا ملف PDF جديد مع الصفحة التي دوّرناها وحفظناه بالاسم `rotatedPage.pdf` ❸.

سيحتوي ملف PDF الناتج على صفحة واحدة مع تدويرها بمقدار 90 درجة باتجاه عقارب الساعة كما هو موضّح في الشكل التالي، وتحتوي القيم المُعادة من التابعين `rotateClockwise()` و`rotateCounterClockwise()` على الكثير من المعلومات التي يمكنك تجاهلها.



الشكل 77: ملف rotatedPage.pdf مع تدوير الصفحة بمقدار 90 درجة باتجاه عقارب الساعة

15.1.6 دمج الصفحات

يمكن لوحدة PyPDF2 أن تدمج محتويات صفحة مع صفحة أخرى، ويُعَد ذلك مفيدًا لإضافة شعار أو علامة زمنية أو مائية إلى الصفحة، إذ تسهّل لغة بايثون إضافة علامات مائية إلى ملفات متعددة وعلى الصفحات التي يحدّدها برنامجك فقط.

نزّل الملف watermark.pdf، ثم ضعه في مجلد العمل الحالي مع الملف meetingminutes.pdf، ثم أدخِل ما يلي في الصدفة التفاعلية:

```
>>> import PyPDF2
>>> minutesFile = open('meetingminutes.pdf', 'rb')
❶ >>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
❷ >>> minutesFirstPage = pdfReader.getPage(0)
❸ >>> pdfWatermarkReader = PyPDF2.PdfFileReader(open('watermark.pdf',
'rb'))
❹ >>> minutesFirstPage.mergePage(pdfWatermarkReader.getPage(0))
❺ >>> pdfWriter = PyPDF2.PdfFileWriter()
❻ >>> pdfWriter.addPage(minutesFirstPage)
❼ >>> for pageNum in range(1, pdfReader.numPages):
        pageObj = pdfReader.getPage(pageNum)
        pdfWriter.addPage(pageObj)
>>> resultPdfFile = open('watermarkedCover.pdf', 'wb')
>>> pdfWriter.write(resultPdfFile)
>>> minutesFile.close()
>>> resultPdfFile.close()
```

أنشأنا في المثال السابق كائن PdfFileReader للملف meetingminutes.pdf ❶، واستدعينا التابع getPage(0) للحصول على كائن Page للصفحة الأولى وتخزين هذا الكائن في المتغير minutesFirstPage ❷. أنشأنا بعد ذلك كائن PdfFileReader للملف watermark.pdf ❸، واستدعينا

التابع `mergePage()` للمتغير `minutesFirstPage` ❹، فالوسيط الذي نمرّره إلى التابع `mergePage()` هو كائن `Page` للصفحة الأولى من الملف `watermark.pdf`.

استدعينا التابع `mergePage()` للمتغير `minutesFirstPage`، وبالتالي أصبح هذا المتغير يمثّل الصفحة الأولى التي وضعنا عليها علامة مائية، ثم أنشأنا كائن `PdfFileWriter` ❺ وأضفنا الصفحة الأولى التي وضعنا عليها علامة مائية ❻، ثم مررنا بحلقة على بقية الصفحات الموجودة في الملف `meetingminutes.pdf`، وأضفناها إلى الكائن `PdfFileWriter` ❼. أخيرًا، فتحنا ملف PDF جديد اسمه `watermarkedCover.pdf`، وكتبنا محتويات الكائن `PdfFileWriter` فيه.

يبين الشكل التالي النتائج، حيث يحتوي ملف PDF الجديد `watermarkedCover.pdf` على جميع محتويات الملف `meetingminutes.pdf`، وتحمل الصفحة الأولى فيه علامة مائية:

الشكل 78: محتويات ملف PDF الجديد watermarkedCover.pdf

15.1.7 تشفير ملفات PDF

يمكن لكائن `PdfFileWriter` أيضًا إضافة تشفيرٍ إلى مستند PDF. إذًا فلنجرب إدخال ما يلي في الصدفة التفاعلية:

```
>>> import PyPDF2
>>> pdfFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(pdfFile)
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> for pageNum in range(pdfReader.numPages):
    pdfWriter.addPage(pdfReader.getPage(pageNum))
```

```
❶ >>> pdfWriter.encrypt('swordfish')
 >>> resultPdf = open('encryptedminutes.pdf', 'wb')
 >>> pdfWriter.write(resultPdf)
 >>> resultPdf.close()
```

استدعِ التابع `encrypt()` ومرّر إليه سلسلة كلمة المرور ❶ قبل استدعاء التابع `write()` للحفظ في الملف. يمكن أن تحتوي ملفات PDF على كلمة مرور المستخدم user password التي تسمح لك بعرض ملف PDF وكلمة مرور المالك owner password التي تسمح لك بضبط أذونات للطباعة والتعليق واستخراج النص وميزات أخرى، حيث تُعَد كلمة مرور المستخدم وكلمة مرور المالك الوسيطين الأول والثاني للتابع `encrypt()` على التوالي. إذا مرّرنا سلسلة نصية واحدة فقط كوسيط إلى التابع `encrypt()`، فسنستخدمها لكلمتي المرور.

نسخنا في المثال السابق صفحات الملف `meetingminutes.pdf` إلى كائن `PdfFileWriter` الذي شفّرناه بكلمة المرور `swordfish`، وفتحنا ملف PDF جديد بالاسم `encryptedminutes.pdf`، وكتبنا محتويات الكائن `PdfFileWriter` في ملف PDF الجديد، ويجب إدخال كلمة المرور قبل التمكّن من عرض الملف `encryptedminutes.pdf`. قد ترغب في حذف الملف `meetingminutes.pdf` الأصلي غير المُشفَّر بعد التأكد من تشفير نسخته بصورة صحيحة.

15.2 تطبيق عملي: دمج صفحات مختارة من عدة ملفات PDF

لنفترض أن لديك مهمة مملة تتمثل في دمج عشرات من مستندات PDF في ملف PDF واحد، ويحتوي كل مستند على ورقة غلاف في الصفحة الأولى، ولكنك لا تريد تكرار ورقة الغلاف في النتيجة النهائية، إذ توجد الكثير من البرامج المجانية لدمج ملفات PDF، ولكن تدمج الكثير منها الملفات بأكملها مع بعضها البعض ببساطة. إذًا لنكتب برنامج بايثون لتخصيص الصفحات التي تريدها في ملف PDF الناتج عن دمج عدة مستندات PDF.

إليك الخطوات العامة التي سيطبّقها برنامجك:

1. البحث عن جميع ملفات PDF الموجودة في مجلد العمل الحالي.
2. فرز أسماء الملفات بحيث تُضاف ملفات PDF بالترتيب.
3. كتابة جميع الصفحات باستثناء الصفحة الأولى من كل ملف PDF في الملف الناتج.

ولكن ستحتاج شيفرتك البرمجية إلى تطبيق الخطوات التالية من ناحية التنفيذ:

1. استدعاء التابع `os.listdir()` للعثور على كافة الملفات الموجودة في مجلد العمل وإزالة الملفات التي ليست ملفات PDF.
2. استدعاء تابع فرز القائمة `sort()` الخاصة بلغة بايثون لترتيب أسماء الملفات أبجديًا.
3. إنشاء كائن `PdfFileWriter` لملف PDF الناتج.
4. التكرار ضمن حلقة على كل ملف PDF لإنشاء كائن `PdfFileReader` له.
5. التكرار ضمن حلقة على كل صفحة (ما عدا الصفحة الأولى) في كل ملف PDF.
6. إضافة الصفحات إلى ملف PDF الناتج.
7. كتابة ملف PDF الناتج في ملفٍ اسمه `allminutes.pdf`.

افتح تبويبًا جديدًا لإنشاء ملفٍ جديد في محرّرك واحفظه بالاسم `combinePdfs.py`.

15.2.1 الخطوة الأولى: البحث عن جميع ملفات PDF

أولًا، يجب أن يحصل برنامجك على قائمةٍ بجميع الملفات التي لها الامتداد `.pdf` الموجودة في مجلد العمل الحالي وفرزها، لذا يجب أن تكون شيفرتك البرمجية تبدو كما يلي:

```
#! python3
# combinePdfs.py - دمج جميع ملفات PDF الموجودة في مجلد العمل الحالي في ملف PDF واحد

❶ import PyPDF2, os

# الحصول على جميع أسماء ملفات PDF
pdfFiles = []
for filename in os.listdir('.'):
  if filename.endswith('.pdf'):
    ❷ pdfFiles.append(filename)
❸ pdfFiles.sort(key = str.lower)
❹ pdfWriter = PyPDF2.PdfFileWriter()
```

```
    # التكرار ضمن حلقة على جميع ملفات PDF

    # التكرار ضمن حلقة على جميع الصفحات (باستثناء الصفحة الأولى) وإضافتها

    # حفظ ملف PDF الناتج في ملف
```

السطر الأول في الشيفرة البرمجية السابقة هو سطر Shebang (سطر يبدأ بالسلسلة النصية "`#!`")، والسطر الثاني هو التعليق الوصفي لما يفعله البرنامج، ثم تستورد الشيفرة البرمجية وحدات `os` و `PyPDF2` ❶. يعيد استدعاء التابع `os.listdir('.')` قائمةً بالملفات الموجودة في مجلد العمل الحالي، حيث تتكرر الشيفرة البرمجية ضمن حلقة على هذه القائمة وتضيف الملفات التي لها الامتداد `.pdf` فقط إلى القائمة `pdfFiles` ❷.

تُفرَز بعد ذلك هذه القائمة وفق الترتيب الأبجدي باستخدام وسيط الكلمة المفتاحية Keyword Argument الذي هو `key = str.lower` الخاص بالتابع `sort()` ❸، ويُنشَأ كائن `PdfFileWriter` للاحتفاظ بصفحات PDF المدموجة ❹.

أخيرًا، توجد بعض التعليقات التي توضّح ما تبقى من البرنامج.

15.2.2 الخطوة الثانية: فتح ملفات PDF

يجب الآن أن يقرأ البرنامج كلّ ملف PDF موجودٍ في القائمة `pdfFiles`، لذا أضِف ما يلي إلى برنامجك:

```
#! python3
# combinePdfs.py - دمج جميع ملفات PDF الموجودة في مجلد العمل الحالي في ملف PDF واحد
import PyPDF2, os

# الحصول على جميع أسماء ملفات PDF
pdfFiles = []
--snip--
# التكرار ضمن حلقة على جميع ملفات PDF
for filename in pdfFiles:
  pdfFileObj = open(filename, 'rb')
  pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
  # التكرار ضمن حلقة على جميع الصفحات (باستثناء الصفحة الأولى) وإضافتها

# حفظ ملف PDF الناتج في ملف
```

تفتح الحلقة اسم ملف لكل ملف PDF في وضع القراءة الثنائي من خلال استدعاء الدالة `open()` مع الوسيط الثاني `'rb'`، حيث يعيد استدعاء الدالة `open()` كائن `File` المُمرَّر إلى الدالة `PyPDF2.PdfFileReader()` لإنشاء كائن `PdfFileReader` لملف PDF.

15.2.3 الخطوة الثالثة: إضافة الصفحات

يجب التكرار ضمن حلقة على كل صفحة من كل ملف PDF باستثناء الصفحة الأولى. إذًا أضِف الشيفرة البرمجية التالية إلى برنامجك:

```
#! python3
# combinePdfs.py - دمج جميع ملفات PDF الموجودة في مجلد العمل الحالي في ملف PDF واحد
import PyPDF2, os

--snip--

# التكرار ضمن حلقة على جميع ملفات PDF
for filename in pdfFiles:
--snip--
  # التكرار ضمن حلقة على جميع الصفحات (باستثناء الصفحة الأولى) وإضافتها
❶ for pageNum in range(1, pdfReader.numPages):
    pageObj = pdfReader.getPage(pageNum)
    pdfWriter.addPage(pageObj)
# حفظ ملف PDF الناتج في ملف
```

تنسخ الشيفرة البرمجية الموجودة داخل حلقة `for` كل كائن `Page` إلى كائن `PdfFileWriter`، ولكن تذكّر أنك تريد تخطي الصفحة الأولى. يجب أن تبدأ حلقتك من القيمة `1` ❶ لأن وحدة `PyPDF2` تَعُدّ القيمة `0` هي الصفحة الأولى، ثم تصل إلى العدد الصحيح الموجود في `pdfReader.numPages` دون تضمينه في الحلقة.

15.2.4 الخطوة الرابعة: حفظ النتائج

سيحتوي المتغير `pdfWriter` على كائن `PdfFileWriter` مع الصفحات الخاصة بجميع ملفات PDF المدموجة بعد الانتهاء من حلقات `for` المتداخلة، والخطوة الأخيرة هي كتابة هذا المحتوى في ملفٍ على القرص الصلب، لذا أضِف الشيفرة البرمجية التالية إلى برنامجك:

```
#! python3
# combinePdfs.py -  دمج جميع ملفات PDF الموجودة في مجلد العمل الحالي في ملف PDF واحد
import PyPDF2, os

--snip--

# التكرار ضمن حلقة على جميع الملفات PDF
for filename in pdfFiles:
--snip--
  # التكرار ضمن حلقة على جميع الصفحات (باستثناء الصفحة الأولى) وإضافتها
  for pageNum in range(1, pdfReader.numPages):
  --snip--


# حفظ ملف PDF الناتج في ملف
pdfOutput = open('allminutes.pdf', 'wb')
pdfWriter.write(pdfOutput)
pdfOutput.close()
```

يؤدي تمرير `'wb'` إلى الدالة `open()` إلى فتح ملف PDF الناتج `allminutes.pdf` في وضع الكتابة الثنائي، وبالتالي يؤدي تمرير كائن `File` الناتج إلى التابع `write()` إلى إنشاء ملف PDF الفعلي، ويؤدي استدعاء التابع `close()` إلى إنهاء البرنامج.

15.2.5 أفكار لبرامج مماثلة

بإمكانك إنشاءَ برامج يمكنها تطبيق ما يلي، فقط من خلال القدرة على إنشاء ملفات PDF من صفحات ملفات PDF الأخرى:

- قص صفحات محددة من ملفات PDF.
- إعادة ترتيب الصفحات في ملف PDF.
- إنشاء ملف PDF من الصفحات التي تحتوي على بعض النصوص التي يحدّدها التابع `extractText()`.

15.3 مستندات وورد Word

يمكن للغة بايثون إنشاء وتعديل مستندات وورد التي لها امتداد الملفات `.docx` باستخدام الوحدة `docx` التي يمكنك تثبيتها من خلال تشغيل الأمر `pip install --user -U python-docx==0.8.10`. اتبع الإرشادات الخاصة بتثبيت الوحدات الخارجية التي سنوضّحها في الملحق 1.

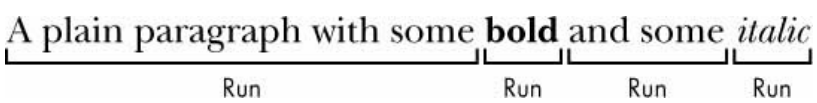
> **ملاحظة**: إذا استخدمتَ الأداة pip لتثبيت وحدة Python-Docx لأول مرة، فتأكد من تثبيت `python-docx` وليس `docx`، فاسم الحزمة `docx` مُخصَّص لوحدةٍ مختلفة لن نتحدّث عنها في هذا الفصل، ولكن ستحتاج إلى تشغيل الأمر `import docx` وليس `import python-docx` عندما تريد استيراد الوحدة من حزمة `python-docx`.

إن لم يكن لديك تطبيق وورد، فيمكنك استخدام ليبر أوفيس رايتر LibreOffice Writer وأوبن أوفيس رايتر OpenOffice Writer، وهما تطبيقان بديلان مجانيان لأنظمة ويندوز Windows وماك macOS ولينكس Linux، ويُستخدمان لفتح ملفات `.docx`، حيث يمكنك تنزيلهما من موقعهما الرسمي، ويوجد التوثيق الكامل لوحدة `Python-Docx` على موقعها الرسمي.

سنركّز في هذا الفصل على وورد في نظام تشغيل ويندوز بالرغم من وجود إصدار من وورد على نظام تشغيل ماك macOS.

تحتوي ملفات `.docx` على هياكل متعددة مقارنةً بالنصوص العادية، ويُمثَّل كلّ هيكلٍ منها بثلاثة أنواع مختلفة من البيانات في الوحدة `Python-Docx`، حيث يمثِّل كائن `Document` المستند بأكمله، ويحتوي كائن `Document` على قائمةٍ من كائنات `Paragraph` للفقرات الموجودة في المستند، إذ تبدأ فقرة جديدة عندما يضغط المستخدم على زر `ENTER` أو `RETURN` أثناء الكتابة في مستند وورد، ويحتوي كل كائن من كائنات `Paragraph` على قائمة تضم كائن `Run` واحدًا أو أكثر.

تحتوي الفقرة المكونة من جملة واحدة في الشكل التالي على أربعة كائنات `Run`:



الشكل 79: كائنات Run الموجودة ضمن كائن Paragraph

يُعَد النص الموجود ضمن مستند وورد أكثرَ من مجرد سلسلة نصية، ولهذا النص نوع خطٍ وحجم ولون ومعلومات تنسيقٍ أخرى مرتبطةٌ به، إذ يمثل النمط Style في وورد مجموعةً من هذه السمات.

يمثّل الكائن `Run` التشغيل المتجاور للنصوص التي لها النمط نفسه، إذ يجب أن يكون هناك كائن `Run` جديد كلما تغيّر نمط النص.

15.3.1 قراءة مستندات وورد

لنختبر الآن وحدة docx، لذا نزّل الملف demo.docx واحفظه في مجلد العمل، ثم أدخِل ما يلي في الصدفة التفاعلية:

```
>>> import docx
❶ >>> doc = docx.Document('demo.docx')
❷ >>> len(doc.paragraphs)
7
❸ >>> doc.paragraphs[0].text
'Document Title'
❹ >>> doc.paragraphs[1].text
'A plain paragraph with some bold and some italic'
❺ >>> len(doc.paragraphs[1].runs)
4
❻ >>> doc.paragraphs[1].runs[0].text
'A plain paragraph with some '
❼ >>> doc.paragraphs[1].runs[1].text
'bold'
❽ >>> doc.paragraphs[1].runs[2].text
' and some '
❾ >>> doc.paragraphs[1].runs[3].text
'italic'
```

نفتح ملف .docx في بايثون ونستدعي الدالة docx.Document() ونمرّر إليها اسم الملف demo.docx ❶، مما يؤدي إلى إعادة كائن Document الذي يحتوي على السمة paragraphs التي تمثل قائمةً من كائنات Paragraph. إذا استدعينا الدالة len() للسمة doc.paragraphs، فستعيد القيمة 7، مما يخبرنا بوجود سبعة كائنات Paragraph في هذا المستند ❷. يمتلك كل كائن من كائنات Paragraph

السمةَ `text` التي تحتوي على سلسلة نصية من النص الموجود في تلك الفقرة (بدون معلومات النمط). تحتوي السمة `text` الأولى في مثالنا على النص `'DocumentTitle'` ❸، وتحتوي السمة `text` الثانية على النص `'A plain paragraph with some bold and some italic'` ❹.

يمتلك كل كائن `Paragraph` أيضًا على السمة `runs` التي تمثل قائمةً من كائنات `Run` التي تمتلك أيضًا السمة `text` التي تحتوي على نص كائن `Run` الخاص بها. لنلقِ نظرةً على سمات `text` في كائن `Paragraph` الثاني، والتي تمثّل النص `'A plain paragraph with some bold and some italic'`، حيث يعطي استدعاء الدالة `len()` لهذا الكائن القيمةَ 4، والتي تمثل وجود أربعة كائنات `Run` ❺. يحتوي كائن `Run` الأول على النص `'A plain paragraph with some '` ❻، ثم يتغير النص إلى نمط خط عريض، وبالتالي يبدأ النص `'bold'` كائن `Run` جديد ❼، بعد ذلك يعود النص إلى نمط خطٍ غير عريض، مما يعطي كائن `Run` ثالث، وهو النص `' and some '` ❽. أخيرًا، يحتوي كائن `Run` الرابع والأخير على النص `'italic'` بنمط خطٍ مائل ❾.

ستتمكن برامج پایثون الآن باستخدام الوحدة `Python-Docx` من قراءة النص من ملف `.docx` واستخدامه مثل أيّ قيمة سلسلة نصية أخرى.

15.3.2 الحصول على النص الكامل من ملف امتداده .docx

إذا كان اهتمامك بالنص فقط دون الاهتمام بمعلومات التنسيق في مستند وورد، فيمكنك استخدام الدالة `getText()` التي تأخذ اسم ملف `.docx` وتعيد قيمة سلسلة نصية واحدة تمثّل النص الخاص بهذا الملف.

افتح تبويبًا جديدًا لإنشاء ملف جديد في محرّرك، وأدخِل الشيفرة البرمجية التالية، واحفظ الملف بالاسم `readDocx.py`:

```
#! python3
import docx

def getText(filename):
  doc = docx.Document(filename)
  fullText = []
  for para in doc.paragraphs:
    fullText.append(para.text)
  return '\n'.join(fullText)
```

تفتح الدالة `getText()` مستند وورد، وتتكرر ضمن حلقـة على كافـة كائنـات `Paragraph` الموجـودة في القائمة `paragraphs`، ثم تلحِق النص الخاص بها بالقائمة الموجودة في المتغير `fullText`. تُضَـم السلاسـل النصية الموجودة في المتغير `fullText` بعد انتهاء الحلقة مع محارف السطر الجديد.

يمكن اسـتيراد برنـامج `readDocx.py` مثـل أي وحـدة أخـرى، وإذا أردتَ النص فقـط من مسـتند وورد، فيمكنك إدخال ما يلي:

```
>>> import readDocx
>>> print(readDocx.getText('demo.docx'))
Document Title
A plain paragraph with some bold and some italic
Heading, level 1
Intense quote
first item in unordered list
first item in ordered list
```

يمكنك أيضًا ضبط الدالة `getText()` لتعديل السلسلة النصية قبل إعادتها، فمثلًا يمكننا وضع مسافة بادئة لكل فقرة من خلال تعديل استدعاء التابع `append()` في الملف `readDocx.py` كما يلي:

```
fullText.append('  ' + para.text)
```

يمكننا إضافة مسافة مزدوجة بين الفقرات من خلال تغيير شيفرة استدعاء التابع `join()` إلى ما يلي:

```
return '\n\n'.join(fullText)
```

لاحظ أنك لا تحتاج سوى بضعة أسطر من الشيفرة البرمجية لكتابة الدوال التي تقرأ ملف `.docx` وتعيد سلسلة نصية من محتوى هذا الملف حسب رغبتك.

15.3.3 تنسيق كائنات Paragraph وكائنات Run

يمكنك رؤية الأنماط في وورد ضمن نظـام ويـندوز بالضـغط على المفـاتيح `Ctrl-Alt-Shift-S` لعـرض لوحة الأنماط Styles التي تشبه الشكل التالي، بينما يمكنك عرض لوحة الأنماط في نظام تشغيل ماك macOS بالنقر على عنصر قائمة العرض View ثم الأنماط Styles.

الشكل 80: عرض لوحة الأنماط بالضغط على المفاتيح CTRL-ALT-SHIFT-S في نظام ويندوز

يستخدم برنامج وورد ومعالجات النصوص الأخرى أنماطًا للحفاظ على تناسق العرض المرئي لأنواع النصوص المتشابهة وسهولة تغييره، فمثلًا قد ترغب في ضبط فقرات النص لتكون بخطٍ من النوع Times New Roman وحجمه 11 نقطة ومتحاذٍ من جهة اليسار وغير مضبوط من جهة اليمين، حيث يمكنك إنشاء نمط باستخدام هذه الإعدادات وإسناده لجميع فقرات النص، وإذا أردتَ لاحقًا تغيير طريقة عرض جميع فقرات النص في المستند، فيمكنك تغيير النمط فقط، وستُحدَّث جميع تلك الفقرات تلقائيًا.

هناك ثلاثة أنواع من الأنماط بالنسبة لمستندات وورد وهي: أنماط الفقرة Paragraph Styles التي يمكن تطبيقها على كائنات `Paragraph`، وأنماط المحارف Character Styles التي يمكن تطبيقها على كائنات `Run`، والأنماط المرتبطة Linked Styles التي يمكن تطبيقها على كلا النوعين من الكائنات. يمكنك إعطاء كائنات `Paragraph` وكائنات `Run` أنماطٍ من خلال ضبط السمة `style` الخاصة بها على سلسلةٍ نصية تمثّل اسم النمط، وإذا كانت هذه السمة مضبوطةً على القيمة `None`، فلن يكون هناك نمط مرتبط بكائن `Paragraph` أو كائن `Run`.

إليك قيم السلاسل النصية لأنماط وورد الافتراضية:

```
'Normal'         'Heading 5'      'List Bullet'     'List Paragraph'
'Body Text'      'Heading 6'      'List Bullet 2'   'MacroText'
'Body Text 2'    'Heading 7'      'List Bullet 3'   'No Spacing'
'Body Text 3'    'Heading 8'      'List Continue'   'Quote'
'Caption'        'Heading 9'      'List Continue 2' 'Subtitle'
'Heading 1'      'Intense Quote'  'List Continue 3' 'TOC Heading'
'Heading 2'      'List'           'List Number '    'Title'
'Heading 3'      'List 2'         'List Number 2'
'Heading 4'      'List 3'         'List Number 3'
```

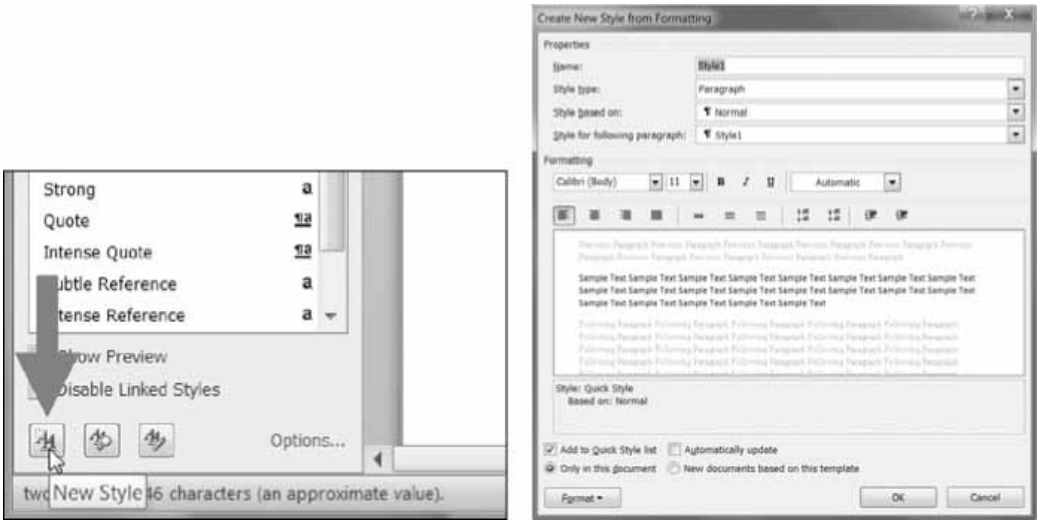
الشكل 81: قيم السلاسل النصية لأنماط وورد الافتراضية

يجب إضافة `' Char'` إلى نهاية اسم النمط عند استخدام نمط مرتبط بكائن Run، فمثلًا يمكنك ضبط النمط المرتبط Quote لكائن Paragraph من خلال استخدام `paragraphObj.style = 'Quote'`، ولكنك ستستخدم `runObj.style = 'Quote Char'` بالنسبة لكائن Run.

الأنماط الوحيدة التي يمكن استخدامها في الإصدار 0.8.10 من وحدة Python-Docx هي أنماط وورد الافتراضية والأنماط الموجودة في ملف `.docx` المفتوح، ولا يمكن إنشاء أنماط جديدة، بالرغم من أن ذلك قد تغيّر في الإصدارات اللاحقة من وحدة Python-Docx.

15.3.4 إنشاء مستندات وورد مع أنماط غير افتراضية

إذا أردتَ إنشاء مستندات وورد تستخدم أنماطٍ مختلفة عن الأنماط الافتراضية، فيجب فتح وورد على مستند فارغ وإنشاء الأنماط بنفسك من خلال النقر على زر "نمط جديد New Style" الموجود أسفل لوحة الأنماط كما هو موضح في الشكل التالي على نظام ويندوز:



الشكل 82: كيفية إنشاء مستندات وورد

سيؤدي الضغط على زر نمط جديد إلى فتح نافذة "إنشاء نمط جديد من التنسيق Create New Style from Formatting" حيث يمكنك إدخال النمط الجديد. ارجع بعد ذلك إلى الصدفة التفاعلية وافتح هذا المستند الفارغ باستخدام الدالة `docx.Document()`، واستخدمه كأساسٍ لمستند وورد الخاص بك. سيكون الاسم الذي أعطيته لهذا النمط متاحًا الآن للاستخدام مع وحدة `Python-Docx`.

15.3.5 سمات الكائن Run

يمكن تنسيق كائنات `Run` باستخدام سمات `text`، حيث يمكن ضبط كل سمة على قيمة من ثلاث قيم هي: القيمة `True` (تكون السمة مُفعَّلةً دائمًا بغض النظر عن الأنماط الأخرى المُطبَّقة على الكائن `Run`)، أو القيمة `False` (تكون السمة مُعطَّلة دائمًا)، أو القيمة `None` (الإعداد الافتراضي لأيّ نمطٍ مضبوط للكائن `Run`).

يوضّح الجدول الآتي السمات `text` التي يمكن ضبطها لكائنات `Run`:

السمة	وصفها
السمة `bold`	يظهر النص بخط عريض
السمة `italic`	يظهر النص بخط مائل
السمة `underline`	يوضَع خط تحت النص
السمة `strike`	يظهر النص مع خطٍ في وسطه
السمة `double_strike`	يظهر النص مع خط مزدوج في وسطه
السمة `all_caps`	يظهر النص بحروف كبيرة
السمة `small_caps`	يظهر النص بحروف كبيرة، وتكون الحروف الصغيرة أصغر بنقطتين
السمة `shadow`	يظهر النص مع ظل

السمة `outline`	يظهر النص مُحدَّدًا وليس ممتلئًا
السمة `rtl`	النص مكتوب من اليمين إلى اليسار
السمة `imprint`	يظهر النص مضغوطًا إلى داخل الصفحة
السمة `emboss`	يبدو النص مرتفعًا عن الصفحة ارتفاعًا بارزًا

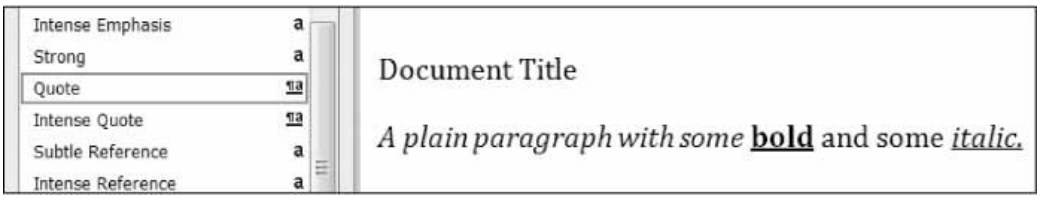
الجدول 21: السمات text التي يمكن ضبطها لكائنات Run

أدخِل مثلًا ما يلي في الصدفة التفاعلية لتغيير أنماط الملف `demo.docx`:

```
>>> import docx
>>> doc = docx.Document('demo.docx')
>>> doc.paragraphs[0].text
'Document Title'
>>> doc.paragraphs[0].style # قد يكون المعرف id مختلف:
_ParagraphStyle('Title') id: 3095631007984
>>> doc.paragraphs[0].style = 'Normal'
>>> doc.paragraphs[1].text
'A plain paragraph with some bold and some italic'
>>> (doc.paragraphs[1].runs[0].text, doc.paragraphs[1].runs[1].text,
doc.
paragraphs[1].runs[2].text, doc.paragraphs[1].runs[3].text)
('A plain paragraph with some ', 'bold', ' and some ', 'italic')
>>> doc.paragraphs[1].runs[0].style = 'QuoteChar'
>>> doc.paragraphs[1].runs[1].underline = True
>>> doc.paragraphs[1].runs[3].underline = True
>>> doc.save('restyled.docx')
```

استخدمنا في المثال السابق سمات `text` و `style` لرؤية ما هو موجود في الفقرات ضمن المستند بسهولة، إذ يمكننا أن نرى أنه من السهل تقسيم الفقرة إلى كائنات Run والوصول إلى كلٍّ منها على حدة، لذلك يمكننا الحصول على كائنات Run الأولى والثانية والرابعة في الفقرة الثانية، وتنسيق كلٍّ منها، وحفظ النتائج في مستند جديد.

ستكون للكلمات `Document Title` الموجودة في أعلى المستند `restyled.docx` النمط العادي Normal بدلًا من نمط العنوان Title، وسيكون لكائن Run الخاص بالنص `A plain paragraph with some` النمط QuoteChar، وسيكون لكائني Run الخاصين بالكلمتين `bold` و `italic` سمات `underline` المضبوطة على القيمة `True`. يوضح الشكل التالي كيف تبدو أنماط الفقرات وكائنات Run في المستند `restyled.docx`:



الشكل 83: ملف restyled.docx

15.3.6 كتابة مستندات وورد

لندخِل ما يلي في الصدفة التفاعلية:

```
>>> import docx
>>> doc = docx.Document()
>>> doc.add_paragraph('Hello, world!')
<docx.text.Paragraph object at 0x0000000003B56F60>
>>> doc.save('helloworld.docx')
```

يمكننا إنشاء ملف `.docx` من خلال استدعاء الدالة `docx.Document()` لإعادة كائن مستند `Document` وورد جديد وفارغ، ويضيف التابع `add_paragraph()` الخاص بالمستند فقرةً نصية جديدة إلى المستند ويعيد مرجعًا إلى كائن `Paragraph` المُضاف. نمرّر سلسلةً نصية تمثّل اسم الملف إلى التابع `save()` الخاص بالمستند عند الانتهاء من إضافة النص لحفظ الكائن `Document` في ملف. تؤدي الشيفرة البرمجية السابقة إلى إنشاء ملف بالاسم `helloworld.docx` في مجلد العمل الحالي والذي يبدو عند فتحه كما يلي:

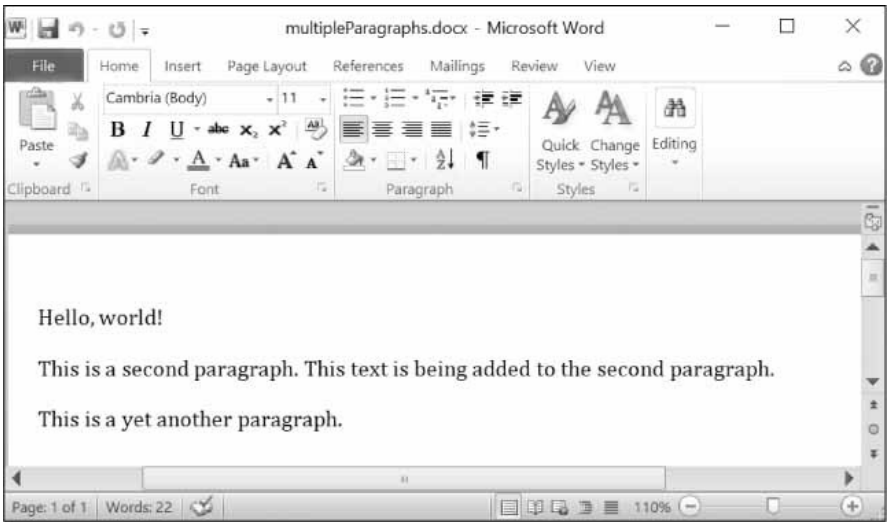
الشكل 84: مستند وورد الذي أنشأناه باستخدام الاستدعاء add_paragraph('Hello, world!')

يمكنك إضافة فقرات من خلال استدعاء التابع add_paragraph() مرةً أخرى مع نص الفقرة الجديدة، أو يمكنك استدعاء التابع add_run() الخاص بالفقرة وتمرير سلسلة نصية إليه لإضافة نص إلى نهاية فقرة موجودة مسبقًا. إذًا لندخِل ما يلي في الصدفة التفاعلية:

```
>>> import docx
>>> doc = docx.Document()
>>> doc.add_paragraph('Hello world!')
<docx.text.Paragraph object at 0x0000000000366AD30>
>>> paraObj1 = doc.add_paragraph('This is a second paragraph.')
>>> paraObj2 = doc.add_paragraph('This is a yet another paragraph.')
>>> paraObj1.add_run(' This text is being added to the second
paragraph.')
<docx.text.Run object at 0x0000000003A2C860>
>>> doc.save('multipleParagraphs.docx')
```

لاحظ أن النص "This text is being added to the second paragraph." أضيف إلى كائن Paragraph في المتغير paraObj1، وهو الفقرة الثانية المُضافة إلى المتغير doc. تعيد الدالتان add_paragraph() و add_run() كائنات Paragraph و Run على التوالي بحيث توفّر عليك عناء استخراجها في خطوة منفصلة. ضع في بالك أنه يمكن إضافة كائنات Paragraph الجديدة إلى نهاية المستند فقط، ويمكن إضافة كائنات Run الجديدة إلى نهاية كائن Paragraph فقط، وذلك اعتبارًا من الإصدار 0.8.10 من وحدة Python-Docx.

أخيرًا، يمكن استدعاء التابع save() مرة أخرى لحفظ التغييرات الإضافية التي أجريتها. سيبدو المستند الناتج مثل المستند الموضّح في الشكل التالي:



الشكل 85: المستند الذي يحتوي على كائنات Paragraph و Run المتعددة المُضافة

تأخذ كلّ من الدالتين `add_paragraph()` و `add_run()` وسيطًا ثانيًا اختياريًا، وهـو سلسـلة نصـية من نمط الكائن `Paragraph` أو `Run` كما في المثال التالي:

```
>>> doc.add_paragraph('Hello, world!', 'Title')
```

يضيف السطر السابق فقرةً تحتوي على النص "`Hello, world!`" من نمط العنوان Title Style.

15.3.7 إضافة العناوين Headings

يؤدي استدعاء الدالة `add_heading()` إلى إضافة فقرة تحتوي على أحد أنماط العناوين.

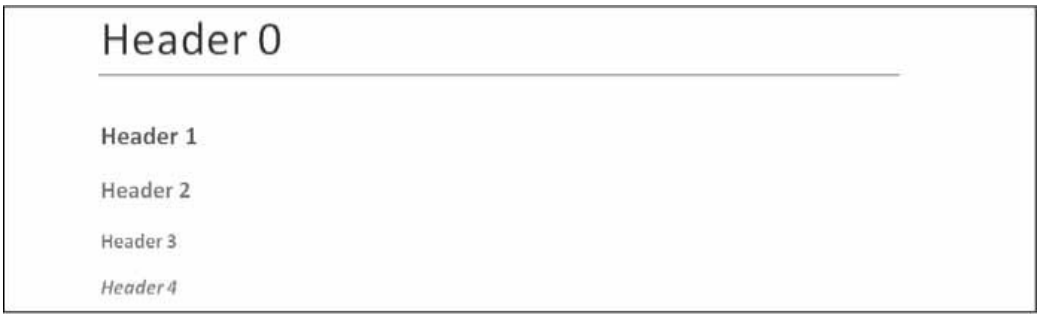
لندخِل ما يلي في الصدفة التفاعلية:

```
>>> doc = docx.Document()
>>> doc.add_heading('Header 0', 0)
<docx.text.Paragraph object at 0x00000000036CB3C8>
>>> doc.add_heading('Header 1', 1)
<docx.text.Paragraph object at 0x00000000036CB630>
>>> doc.add_heading('Header 2', 2)
<docx.text.Paragraph object at 0x00000000036CB828>
>>> doc.add_heading('Header 3', 3)
<docx.text.Paragraph object at 0x00000000036CB2E8>
>>> doc.add_heading('Header 4', 4)
<docx.text.Paragraph object at 0x00000000036CB3C8>
>>> doc.save('headings.docx')
```

وسطاء الدالة `add_heading()` هي سلسلة نصية تمثّل نص العنـوان وعـدد صـحيح قيمتـه من 0 إلى 4، حيث يجعل العدد الصحيح 0 العنوان من النمط Title style، والذي يُسـتخدَم في الجـزء العلـوي من المسـتند، والأعـداد الصـحيحة من 1 إلى 4 مُخصَّصـة لمسـتويات العنـاوين المختلفـة، حيث يكـون العـدد 1 هـو العنـوان الرئيسي والعدد 4 هو العنوان الفرعي الأدنى.

تعيـد الدالـة `add_heading()` كـائن `Paragraph` لتـوفّر عليـك إجـراء خطـوة اسـتخراجه من كـائن `Document` في خطوة منفصلة.

سيبدو ملف `headings.docx` الناتج مثل المستند الموضّح في الشكل التالي:



الشكل 86: مستند headings.docx الذي يحتوي على العناوين من 0 إلى 4

15.3.8 إضافة فواصل الأسطر والصفحات

يمكن إضافة فاصل أسطر بدلًا من بدء فقرة جديدة بالكامل من خلال استدعاء التابع `add_break()` لكائن Run الـذي تريـد ظهـور الفاصـل بعـده، وإذا أردتَ إضـافة فاصـل صـفحات، فيجب تمريـر القيمـة `docx.enum.text.WD_BREAK.PAGE` بوصفها وسيطًا وحيدًا للتابع `add_break()` كمـا في السـطر ❶ من المثال التالي:

```
>>> doc = docx.Document()
>>> doc.add_paragraph('This is on the first page!')
<docx.text.Paragraph object at 0x0000000003785518>
❶ >>> doc.paragraphs[0].runs[0].add_break(docx.enum.text.WD_BREAK.PAGE)
>>> doc.add_paragraph('This is on the second page!')
<docx.text.Paragraph object at 0x00000000037855F8>
>>> doc.save('twoPage.docx')
```

تؤدي الشيفرة البرمجية السابقة إلى إنشاء مستند وورد مؤلف من صفحتين مع وجـود النص "This is on the first page!" في الصفحة الأولى والنص "This is on the second page!" في الصفحة الثانية. لا يزال هناك مساحة كبيرة في الصفحة الأولى بعد النص "This is on the first page!"، ولكننا أجبرنا الفقرة التالية على البدء في صفحة جديدة من خلال إدراج فاصل صفحات بعد كائن Run الأول للفقرة الأولى ❶.

15.3.9 إضافة الصور

تحتوي كائنات Document على التـابع `add_picture()` الـذي يـتيح إضـافة صـورةٍ إلى نهايـة المسـتند. لنفـترض أن لـديك ملفًـا اسـمه `zophie.png` مثلًا في مجلـد العمـل الحـالي، حيث يمكنـك إضـافة الصـورة

zophie.png إلى نهاية مستندك بعرض 1 بوصة وارتفاع 4 سنتيمترات من خلال إدخال ما يلي (يستخدم برنامج وورد الوحدات الإنجليزية والمترية):

```
>>> doc.add_picture('zophie.png', width=docx.shared.Inches(1),
height=docx.shared.Cm(4))
<docx.shape.InlineShape object at 0x00000000036C7D30>
```

الوسيط الأول للتابع add_picture() هو سلسلة نصية تمثّل اسم ملف الصورة، وتضبط وسطاء الكلمات المفتاحية width و height الاختيارية عرض الصورة وارتفاعها في المستند، وإذا تُرِكا دون ضبط، فسيكون العرض والارتفاع الافتراضي هو الحجم الطبيعي للصورة.

قد تفضّل تحديد ارتفاع الصورة وعرضها بوحدات مألوفة مثل وحدات البوصة والسنتيمتر، لذا يمكنك استخدام الدالتين docx.shared.Inches() و docx.shared.Cm() عندما تحدّد وسطاء الكلمات المفتاحية width و height.

15.4 إنشاء ملفات PDF من مستندات وورد

لا تسمح لك وحدة PyPDF2 بإنشاء مستندات PDF مباشرةً، ولكن توجد طريقة لإنشاء ملفات PDF باستخدام بايثون إذا كنت تستخدم نظام ويندوز مع وجود مايكروسوفت وورد مثبّتًا عليه، لذا ستحتاج إلى تثبيت الحزمة Pywin32 من خلال تشغيل الأمر pip install --user -U pywin32==224. إذا استخدمتَ هذه الحزمة مع وحدة docx، فيمكنك إنشاء مستندات وورد ثم تحويلها إلى ملفات PDF باستخدام السكربت التالي، لذا افتح تبويبًا جديدًا لإنشاء ملف جديد في محرّرك، وأدخِل الشيفرة البرمجية التالية، واحفظها بالاسم convertWordToPDF.py:

```
# يعمل هذا السكربت على نظام ويندوز فقط، ويجب أن يكون وورد مثبتًا عليه

import win32com.client # التثبيت عبر "pip install pywin32==224"

import docx
wordFilename = 'your_word_document.docx'
pdfFilename = 'your_pdf_filename.pdf'

doc = docx.Document()

# ضع شيفرة إنشاء مستند وورد هنا

doc.save(wordFilename)


wdFormatPDF = 17 # رمز مستندات وورد الرقمي لملفات بي دي اف.
```

```
wordObj = win32com.client.Dispatch('Word.Application')

docObj = wordObj.Documents.Open(wordFilename)
docObj.SaveAs(pdfFilename, FileFormat=wdFormatPDF)
docObj.Close()
wordObj.Quit()
```

يمكنك كتابة برنامج ينتج عنه ملفات PDF مع المحتوى الخاص بك من خلال استخدام وحدة `docx` لإنشاء مستند وورد، ثم استخدام وحدة `win32com.client` الخاصة بحزمة `Pywin32` لتحويله إلى ملف PDF. ضع استدعاءات دوال الوحدة `docx` مكان التعليق `# ضع شيفرة إنشاء مستند وورد هنا` لإنشاء المحتوى الخاص بك لملف PDF في مستند وورد.

قد تبدو هذه الطريقة لإنتاج ملفات PDF معقدة، ولكن هذا طبيعي، إذ تكون الحلول البرمجية الاحترافية معقدةً في أغلب الأحيان.

15.5 أسئلة للتدريب

1. لا تُمرَّر قيمة السلسلة النصية التي تمثّل اسم ملف PDF إلى الدالة `PyPDF2.PdfFileReader()`، إذًا ما الذي يمكنك تمريره إلى هذه الدالة بدلًا من ذلك؟
2. ما هي الأوضاع التي يجب فتح كائنات `File` الخاصة بالدالتين `PdfFileReader()` و`PdfFileWriter()` بها؟
3. كيف يمكنك الحصول على كائن `Page` للصفحة 5 من كائن `PdfFileReader`؟
4. ما هو متغير الكائن `PdfFileReader` الذي يخزن عدد الصفحات الموجودة في مستند PDF؟
5. إذا شفّرنا ملف PDF الخاص بكائن `PdfFileReader` باستخدام كلمة المرور `swordfish`، فماذا يجب عليك فعله قبل أن تتمكّن من الحصول على كائنات `Page` منه؟
6. ما هي التوابع التي تستخدمها لتدوير الصفحة؟
7. ما هو التابع الذي يعيد كائن `Document` لملفٍ اسمه `demo.docx`؟
8. ما الفرق بين كائن `Paragraph` وكائن `Run`؟
9. كيف تحصل على قائمة بكائنات `Paragraph` لكائن `Document` المُخزّن في متغير اسمه `doc`؟
10. ما هو نوع الكائن الذي يحتوي على المتغيرات `bold` و `underline` و `italic` و `strike`؟
11. ما هو الفرق بين ضبط المتغير `bold` على القيم `True` أو `False` أو `None`؟

12. كيف يمكنك إنشاء كائن `Document` لمستند وورد جديد؟

13. كيف يمكنك إضافة فقرة تحتـوي على النص `'Hello, there!'` إلى كـائن `Document` مُخـزَّن في متغير اسمه `doc`؟

14. ما هي الأعداد الصحيحة التي تمثل مستويات العناوين المتوفرة في مستندات وورد؟

15.6 مشاريع للتدريب

حاول كتابة البرامج التي تؤدي المهام التي سنوضّحها فيما يلي لكسب خبرة عملية أكبر.

15.6.1 برنامج للتأكد من تشفير ملفات PDF

استخدم الدالة `os.walk()` لكتابة سكربتٍ يمر على كل ملف PDF في المجلـد ومجلداتـه الفرعيـة، وشـفّر ملفات PDF باستخدام كلمة المرور المتوفرة في سطر الأوامر، واحفظ كل ملف PDF مشفّر مـع إضـافة اللاحقـة `_encrypted.pdf` إلى اسم الملف الأصلي، ثم اطلب من البرنامج محاولة قراءة الملف وفـك تشـفيره للتأكـد من تشفيره بصورة صحيحة قبل حذف الملف الأصلي.

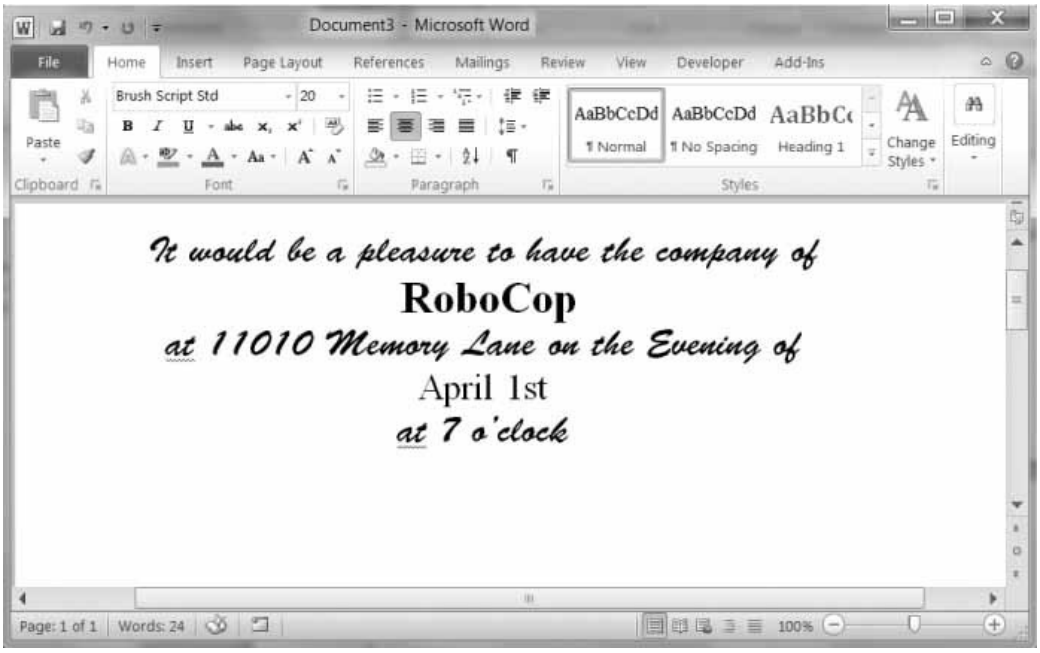
اكتب بعد ذلك برنامجًا يبحث عن جميع ملفات PDF المشفرة في المجلد ومجلداته الفرعية، وينشئ نسـخة مشفَّرة من ملف PDF باستخدام كلمة المرور المتوفرة. إذا كانت كلمة المرور غير صـحيحة، فيجب على البرنـامج طباعة رسالة للمستخدم والانتقال إلى ملف PDF التالي.

15.6.2 برنامج لإنشاء دعوات مخصصة في مستندات وورد

لنفترض أن لديك ملفًا نصيًا بأسماء الضيوف، حيث يحتوي الملف `guests.txt` على اسم شـخص واحـد في كل سطر كما يلي:

```
Prof. Plum
Miss Scarlet
Col. Mustard
Al Sweigart
RoboCop
```

اكتب برنامجًا ينشئ مستند وورد يحتوي على دعوات مخصصة كما يلي:



الشكل 87: مستند وورد الذي أنشأناه باستخدام سكربت الدعوات المخصصة

يمكن للوحدة `Python-Docx` استخدام الأنماط الموجودة مسبقًا في مستند وورد فقط، لذا يجب أولًا إضافة هذه الأنماط إلى ملف وورد فارغ ثم فتح هذا الملف باستخدام وحدة `Python-Docx`. يجب أن تكون هناك دعوة واحدة لكل صفحة في مستند وورد الناتج، لذا استدعِ التابع `add_break()` لإضافة فاصل صفحة بعد الفقرة الأخيرة من كل دعوة، وبالتالي يجب فتح مستند وورد واحد فقط لطباعة كافة الدعوات دفعةً واحدة.

> **ملاحظة**: يمكنك أيضًا تنزيل نموذج الملف guests.txt.

15.6.3 برنامج لاستخدام هجوم القوة الغاشمة لكسر كلمة مرور ملفات PDF

لنفترض أن لديك ملف PDF مشفَّرًا نسيت كلمة مروره، ولكنك تتذكر أنه كان كلمة إنجليزية واحدة، وتُعَد محاولة تخمين كلمة المرور التي نسيتها مهمةً مملة جدًا، لذا يمكنك كتابة برنامج يفك تشفير ملف PDF من خلال تجربة جميع الكلمات الإنجليزية الممكنة حتى يجد الكلمة الصحيحة، ويسمى ذلك هجوم القوة الغاشمة لإيجاد كلمة المرور. نزّل الملف النصي dictionary.txt الذي يحتوي على أكثر من 44000 كلمة إنجليزية بحيث توجد كلمة واحدة في كل سطر.

استخدم مهارات قراءة الملفات التي تعلمتها سابقًا لإنشاء قائمةٍ بالسلاسل النصية التي تمثّل الكلمات من خلال قراءة الملف `dictionary.txt`، ثم المرور على كل كلمة في هذه القائمة، وتمريرها إلى التابع `decrypt()`. إذا أعاد هذا التابع العدد الصحيح 0، فستكون كلمة المرور خاطئة ويجب أن ينتقل برنامجك إلى كلمة المرور التالية، وإذا أعاد التابع `decrypt()` القيمة 1، فيجب أن يخرج برنامجك من الحلقة ويطبع كلمة المرور المُخترقة، ويجب عليك أيضًا تجربة كلٍّ من الحروف الكبيرة والصغيرة لكل كلمة. يستغرق استعراض

جميع الكلمات الكبيرة والصغيرة البالغ عددها 88000 كلمة من ملف القاموس بضع دقائق، ولذلك يجب عدم استخدام كلمة إنجليزية بسيطة لكلمات المرور الخاصة بك.

15.7 الخلاصة

لا تُعد المعلومات النصية مُخصَّصة للملفات النصية العادية فقط، إذ يُحتمَل أن تتعامل مع ملفات PDF ومستندات وورد في كثير من الأحيان، حيث يمكنك استخدام وحدة `PyPDF2` لقراءة وكتابة مستندات PDF، ولكن قد لا تؤدي قراءة النص من مستندات PDF دائمًا إلى ترجمة مثالية للسلاسل النصية بسبب تنسيق ملف PDF المعقد، وقد لا تكون بعض ملفات PDF قابلة للقراءة على الإطلاق، وبالتالي لن يحالفك الحظ في هذه الحالات إن لم تدعم التحديثات المستقبلية لوحدة `PyPDF2` ميزات إضافية لملفات PDF.

تُعَد مستندات وورد أكثر موثوقية، ويمكنك قراءتها باستخدام وحدة `docx` الخاصة بحزمة `python-docx`. يمكنك معالجة النصوص في مستندات وورد باستخدام كائنات `Paragraph` و `Run`، ويمكن أيضًا إعطاء هذه الكائنات أنماطًا، بالرغم من أن هذه الأنماط يجب أن تكون من مجموعة الأنماط الافتراضية أو الأنماط الموجودة في المستند مسبقًا.

يمكنك إضافة فقرات وعناوين وفواصل وصور جديدة إلى المستند في نهايته فقط.

ترجِع العديد من قيود التعامل مع ملفات PDF ومستندات وورد إلى أن هذه التنسيقات تهدف إلى عرضها بصورة جيدة للقرّاء، عوضًا عن سهولة تحليلها من طرف البرمجيات، لذا سنوضّح في الفصل التالي تنسيقين شائعين آخرين لتخزين المعلومات هما: ملفات JSON و CSV المُصمَّمة لتستخدمها الحواسيب، وسترى أن لغة بايثون يمكنها العمل مع هذه التنسيقات بسهولة أكبر.

16. تعديل ملفات CSV و JSON ببايثون

تعلّمنا في الفصل السابق كيفية استخراج النص من مستندات PDF وورد التي تُعَدّ ملفاتٍ بتنسيق ثنائي، إذ تتطلب هذه الملفات استخدام وحدات بايثون Python خاصة للوصول إلى بياناتها، بينما تُعَد ملفات CSV وJSON مجرد ملفاتٍ نصية عادية، حيث يمكنك عرضها في محرر نصوص مثل محرر النصوص Mu.

تحتوي لغة بايثون مسبقًا على وحدتين خاصتين هما `csv` و `json`، حيث توفّر كلٌّ منهما دوالًا لمساعدتك في العمل مع تنسيقات هذه الملفات.

يرمز الاختصار CSV إلى "القيم المفصولة بفواصل Comma-separated Values"، وملفات CSV هي جداول بيانات بسيطة مُخزَّنة بوصفها ملفات نصية عادية، وتسهّل وحدة `csv` في بايثون تحليل ملفات CSV. يُنطَق الاختصار JSON بالطريقة "JAY-sawn" أو "Jason"، ولكن لا يهم كيف تنطقها لأن الناس سيقولون أنك تنطقها بطريقة خاطئة في كلتا الحالتين، وهو تنسيق يخزن المعلومات بوصفها شيفرة جافاسكربت مصدرية في ملفات نصية عادية.

JSON هو اختصار لترميز الكائنات باستعمال جافاسكربت JavaScript Object Notation، ولكن لا تحتاج إلى معرفة لغة البرمجة جافاسكربت لاستخدام ملفات JSON، ولكن من المفيد معرفة تنسيق ملفات JSON لأنه يُستخدَم في العديد من تطبيقات الويب.

16.1.1 وحدة CSV

يمثّل كل سطر في ملف CSV صفًا في جدول البيانات، حيث تفصل الفواصل بين الخلايا الموجودة في الصف، فمثلًا سيبدو جدول البيانات example.xlsx في ملف CSV كما يلي:

```
4/5/2015 13:34,Apples,73
4/5/2015 3:41,Cherries,85
4/6/2015 12:46,Pears,14
4/8/2015 8:59,Oranges,52
4/10/2015 2:07,Apples,152
4/10/2015 18:10,Bananas,23
4/10/2015 2:40,Strawberries,98
```

سنستخدم هـذا الملـف لأمثلـة الصـدفة التفاعليـة Interactive Shell الموجـودة في هـذا الفصـل، حيث يمكنك تنزيله أو إدخال النص في محرّر النصوص وحفظه بالاسم `example.csv`.

تُعَد ملفات CSV بسيطة ولا تحتوي على العديد من ميزات جدول بيانات إكسل Excel مثل الميزات التالية:

- ليس لديها أنواع لقيمها، فكل شيء فيها هو سلسلة نصية String.
- ليس لديها إعدادات لحجم الخط أو لونه.
- ليس لديها أوراق عمل متعددة.
- لا يمكنها تحديد عرض الخلية وارتفاعها.
- لا يمكنها أن تحتوي على خلايا مدموجة.
- لا يمكنها تضمين الصور أو المخططات.

ميزة ملفات CSV الأساسية هي البسـاطة، إذ تـدعمها العديـد من أنـواع الـبرامج على نطـاقٍ واسـع، ويمكن عرضها في برامج تحرير النصوص بما في ذلك محرّر النصوص Mu، وتُعَد ملفـات CSV طريقـةً مباشـرة لتمثيـل بيانات جداول البيانات. تنسيق ملف CSV هو مجرد ملف نصي يحتوي على قيم يُفصَل بينها بفواصل.

تُعَد ملفات CSV مجرد ملفات نصية، لذا قد تقرأها بوصفها سلسلة نصية ثم تعالج تلك السلسلة باسـتخدام التقنيات التي تعلمتها سابقًا في الفصل التاسع، فمثلًا يمكنك استدعاء التابع `split(',')` في كـل سـطر من النص للحصول على القيم المفصولة بفواصل بوصفها قائمةً من السلاسل النصية، لأن كل خلية في ملــف CSV مفصولة عن غيرها من الخلايا بفاصلة، ولكن لا تمثّل جميع الفواصل في ملف CSV هذه الحـدود بين الخلايـا، إذ تحتوي ملفات CSV أيضًا على مجموعة خاصة بها من محارف الهروب Escape Characters للسماح بتضمين الفواصل والمحارف الأخرى كجزء من القيم، حيث لا يعالج التابع `split()` محارف الهروب. يجب عليـك دائمًـا استخدام وحدة `csv` لقراءة ملفات CSV وكتابتها بسبب هذه المخاطر المحتملة.

16.1.2 كائنات reader

يمكنك قراءة البيانات من ملف CSV باستخدام وحدة `csv` من خلال إنشاء كائن `reader` الذي يتيح لك التكرار على الأسطر الموجودة في ملف CSV. أدخِل ما يلي في الصدفة التفاعلية مع وضع الملف `example.csv` في مجلد العمل الحالي:

```
❶ >>> import csv
❷ >>> exampleFile = open('example.csv')
❸ >>> exampleReader = csv.reader(exampleFile)
❹ >>> exampleData = list(exampleReader)
❺ >>> exampleData
 [['4/5/2015 13:34', 'Apples', '73'], ['4/5/2015 3:41', 'Cherries',
'85'],
 ['4/6/2015 12:46', 'Pears', '14'], ['4/8/2015 8:59', 'Oranges',
'52'],
 ['4/10/2015 2:07', 'Apples', '152'], ['4/10/2015 18:10', 'Bananas',
'23'],
 ['4/10/2015 2:40', 'Strawberries', '98']]
```

تحتوي لغة بايثون على وحدة `csv`، لذا يمكننا استيرادها ❶ دون الحاجة إلى تثبيتها أولًا. يمكنك قراءة ملف CSV باستخدام وحدة `csv` من خلال فتحه أولًا باستخدام الدالة `open()` ❷ كما تفعل مع أيّ ملف نصي آخر، ولكننا لا نستدعي التابع `read()` أو `readlines()` لكائن `File` الذي تعيده الدالة `open()`، بل نمرّره إلى الدالة `csv.reader()` ❸، مما يؤدي إلى إعادة كائن `reader` لتستخدمه.

لاحظ أنك لا تمرّر السلسلة النصية التي تمثّل اسم الملف مباشرةً إلى الدالة `csv.reader()`.

أكثر الطرق مباشرةً للوصول إلى القيم الموجودة في كائن `reader` هي تحويله إلى قائمة بايثون عادية عبر تمريره إلى التابع `list()` ❹، إذ يعيد استخدام التابع `list()` لكائن `reader` قائمةً من القوائم، والتي يمكنك تخزينها في متغير مثل المتغير `exampleData` الذي يؤدي إدخاله في الصدفة إلى عرض قائمةٍ القوائم ❺.

أصبح لديك ملف CSV بوصفه قائمةً من القوائم، ويمكنك الآن الوصول إلى القيمة الموجودة في صف وعمود محدّد باستخدام التعبير `exampleData[row][col]`، حيث يكون `row` هو فهرس إحدى القوائم الموجودة في المتغير `exampleData`، ويكون `col` هو فهرس العنصر الذي تريده من تلك القائمة

لندخِل الآن ما يلي في الصدفة التفاعلية:

```
>>> exampleData[0][0]
'4/5/2015 13:34'
>>> exampleData[0][1]
'Apples'
>>> exampleData[0][2]
'73'
>>> exampleData[1][1]
'Cherries'
>>> exampleData[6][1]
'Strawberries'
```

لاحظ أن `exampleData[0][0]` ينتقل إلى القائمة الأولى ويعطي السلسلة النصية الأولى، وينتقل `exampleData[0][2]` إلى القائمة الأولى ويعطينا السلسلة النصية الثالثة وإلخ.

16.1.3 قراءة البيانات من كائنات reader في حلقة for

يجب استخدام كائن `reader` في حلقة `for` بالنسبة لملفات CSV الكبيرة، مما يؤدي إلى تجنّب تحميل الملف بأكمله إلى الذاكرة دفعة واحدة، إذًا لندخِل ما يلي مثلًا في الصدفة التفاعلية:

```
>>> import csv
>>> exampleFile = open('example.csv')
>>> exampleReader = csv.reader(exampleFile)
>>> for row in exampleReader:
    print('Row #' + str(exampleReader.line_num) + ' ' + str(row))


Row #1 ['4/5/2015 13:34', 'Apples', '73']
Row #2 ['4/5/2015 3:41', 'Cherries', '85']
Row #3 ['4/6/2015 12:46', 'Pears', '14']
Row #4 ['4/8/2015 8:59', 'Oranges', '52']
Row #5 ['4/10/2015 2:07', 'Apples', '152']
Row #6 ['4/10/2015 18:10', 'Bananas', '23']
Row #7 ['4/10/2015 2:40', 'Strawberries', '98']
```

استوردنا وحدة `csv` وأنشأنا كائن `reader` من ملف CSV، ويمكنك بعد ذلك التكرار ضمن حلقة على الصفوف الموجودة في كائن `reader`، حيث يمثل كلّ صف قائمةً من القيم، وتمثّل كل قيمة خلية. يطبع استدعاء الدالة `print()` رقم الصف الحالي ومحتوياته، ويمكنك الحصول على رقم الصف من خلال استخدام

المتغير `line_num` الخاص بكائن `reader`، والذي يحتوي على رقم السطر الحالي. يمكن تكرار كائن `reader` مرة واحدة فقط، لذا يجب استدعاء الدالة `csv.reader` لإنشاء كائن `reader` وإعادة قراءة ملف CSV.

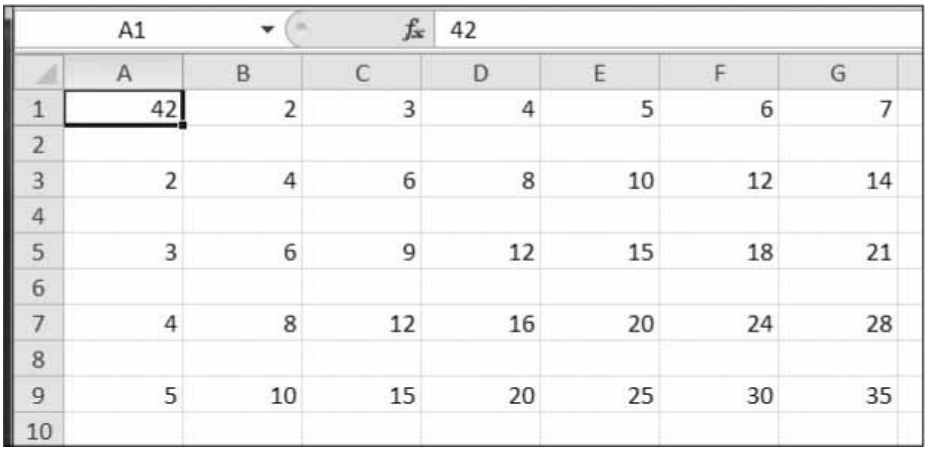
16.1.4 كائنات writer

يتيح كائن `writer` كتابة البيانات في ملف CSV، حيث يمكنك استخدام الدالة `csv.writer()` لإنشاء هذا الكائن. لندخِل الآن ما يلي في الصدفة التفاعلية:

```
>>> import csv
❶ >>> outputFile = open('output.csv', 'w', newline='')
❷ >>> outputWriter = csv.writer(outputFile)
>>> outputWriter.writerow(['spam', 'eggs', 'meat', 'beef'])
21
>>> outputWriter.writerow(['Hello, world!', 'eggs', 'meat', 'beef'])
32
>>> outputWriter.writerow([1, 2, 3.141592, 4])
16
>>> outputFile.close()
```

استدعِ أولًا الدالة `open()` ومرّر لها القيمة `'w'` لفتح الملف في وضع الكتابة ❶، مما يؤدي إلى إنشاء الكائن الذي يمكنك بعد ذلك تمريره إلى الدالة `csv.writer()` ❷ لإنشاء كائن `writer`.

يجب أيضًا في نظام ويندوز تمرير سلسلة نصية فارغة لوسيط الكلمات المفتاحية Keyword Argument الذي هو `newline` للدالة `open()`، وإذا نسيت ضبط الوسيط `newline`، فستكون الصفوف في الملف `output.csv` مزدوجة المسافات كما هو موضّح في الشكل التالي:



	A	B	C	D	E	F	G
1	42	2	3	4	5	6	7
2							
3	2	4	6	8	10	12	14
4							
5	3	6	9	12	15	18	21
6							
7	4	8	12	16	20	24	28
8							
9	5	10	15	20	25	30	35
10							

الشكل 88: إزدواجية مسافات الصفوف في الملف output.csv

في حال ما إذا نسيت وسيط الكلمات المفتاحية `newline=''` في الدالة `open()`، فسيكون ملف CSV مزدوج المسافة

يأخذ التابع `writerow()` الخاص بكائنات `writer` وسيطًا من نوع قائمة، حيث تُوضَع كل قيمة من هذه القائمة في خليتها الخاصة في ملف CSV الناتج، ويعيد هذا التابع عدد المحارف المكتوبة في الملف لهذا الصف بما في ذلك محارف السطر الجديد.

ينتج عن الشيفرة البرمجية السابقة ملف `output.csv` الذي يبدو كما يلي:

```
spam,eggs,meat,beef
"Hello, world!",eggs,meat,beef
1,2,3.141592,4
```

لاحظ كيف يهرّب الكائن `writer` تلقائيًا الفاصلة الموجودة في القيمة `'Hello, world!'` مع علامات الاقتباس المزدوجة في ملف CSV، إذ توفّر وحدة `csv` عليك الاضطرار إلى معالجة هذه الحالات الخاصة بنفسك.

16.1.5 وسطاء الكلمات المفتاحية delimiter و lineterminator

لنفترض أنك تريد الفصل بين الخلايا باستخدام محرف جدولة Tab بدلًا من الفاصلة وتريد أن تكون الصفوف ذات مسافات مزدوجة، لندخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import csv

>>> csvFile = open('example.tsv', 'w', newline='')

❶ >>> csvWriter = csv.writer(csvFile, delimiter='\t',
lineterminator='\n\n')
```

```
>>> csvWriter.writerow(['apples', 'oranges', 'grapes'])
  24
>>> csvWriter.writerow(['eggs', 'meat', 'beef'])
  17
>>> csvWriter.writerow(['spam', 'spam', 'spam', 'spam', 'spam',
'spam'])
  32
>>> csvFile.close()
```

يؤدي ذلك إلى تغيير محارف المحدِّد Delimiter والفاصل بين السطور Line Terminator في ملفك، فالمحدِّد هو المحرف الذي يظهر بين الخلايا في الصف، والمحدِّد الافتراضي لملف CSV هو الفاصلة، بينما يكون الفاصل بين السطور هو المحرف الذي يأتي في نهاية الصف، فالفاصل بين السطور الافتراضي هو محرف السطر الجديد. يمكنك تغيير هذه المحارف إلى قيم مختلفة باستخدام وسطاء الكلمات المفتاحية Keyword Arguments التي هي `delimiter` و `lineterminator` باستخدام الدالة `csv.writer()`.

يؤدي تمرير الوسطاء `delimiter='\t'` و `lineterminator='\n\n'` ❶ إلى تغيير المحرف بين الخلايا إلى محرف الجدولة والمحرف بين الصفوف إلى محرفي سطر جديد. نستدعي بعد ذلك الدالة `writerow()` ثلاث مرات لتعطينا ثلاثة صفوف.

ينتج عن ذلك ملف بالاسم `example.tsv` يحتوي ما يلي:

```
apples  oranges grapes

eggs    meat    beef

spam    spam    spam    spam    spam    spam
```

بهذا نكون قد فصلنا بين الخلايا بمحارف جدولة، وبالتالي سنستخدم امتداد الملف `.tsv` للقيم المفصول بينها بمحارف جدولة.

16.1.6 كائنات DictReader و DictWriter الخاصة بملفات CSV

من الأسهل العمل مع كائنات `DictReader` و `DictWriter` بدلًا من كائنات `reader` و `writer` بالنسبة لملفات CSV التي تحتوي على صفوف الترويسات، إذ تقرأ وتكتب كائنات `reader` و `writer` صفوف ملف CSV باستخدام القوائم، وتطبّق كائنات `DictReader` و `DictWriter` الخاصة بملفات CSV الوظائف

نفسها ولكن باستخدام القواميس Dictionaries، وتستخدم الصف الأول من ملف CSV بوصفها مفاتيحًا لهـذه القواميس.

نزّل الملف exampleWithHeader.csv الذي هو الملف `example.csv` نفسه باستثناء أنـه يحتـوي على ترويسات الأعمدة Timestamp و Fruit و Quantity في الصف الأول.

أدخِل ما يلي في الصدفة التفاعلية لقراءة هذا الملف:

```
>>> import csv
>>> exampleFile = open('exampleWithHeader.csv')
>>> exampleDictReader = csv.DictReader(exampleFile)
>>> for row in exampleDictReader:
...     print(row['Timestamp'], row['Fruit'], row['Quantity'])
...
4/5/2015 13:34 Apples 73
4/5/2015 3:41 Cherries 85
4/6/2015 12:46 Pears 14
4/8/2015 8:59 Oranges 52
4/10/2015 2:07 Apples 152
4/10/2015 18:10 Bananas 23
4/10/2015 2:40 Strawberries 98
```

يضبط الكائنُ `DictReader` ضمن الحلقـة الصـفَّ `row` على كـائن القـاموس مـع المفـاتيح المشـتقة من الترويسات الموجودة في الصف الأول، ولكنـه يَضـبط الصـف `row` على الكـائن `OrderedDict` الـذي يمكنـك استخدامه بالطريقة نفسها لاستخدام القاموس، ولكننا لن نشرح الفرق بين هـاتين الطريقـتين في هـذا الفصـل. يعني استخدام الكائن `DictReader` أنك لا تحتاج إلى شيفرة برمجية إضافية لتخطي معلومات ترويسة الصـف الأول، لأن الكائن `DictReader` يفعل ذلك نيابةً عنك.

إذا حاولتَ اسـتخدام كائنـات `DictReader` مـع الملـف `example.csv` الـذي لا يحتـوي على ترويسـات أعمــدة في الصــف الأول، فسيســتخدم كــائن `DictReader` مفــاتيح القــاموس `'4/5/2015 13:34'` و`'Apples'` و `'73'`، حيث يمكننا تجنب ذلك من خلال تزويـد الدالـة `DictReader()` بوسـيطٍ ثـانٍ يحتـوي على أسماء الترويسات التي تريدها:

```
>>> import csv
>>> exampleFile = open('example.csv')
>>> exampleDictReader = csv.DictReader(exampleFile, ['time', 'name',
'amount'])
```

```
>>> for row in exampleDictReader:
...     print(row['time'], row['name'], row['amount'])
...
4/5/2015 13:34 Apples 73
4/5/2015 3:41 Cherries 85
4/6/2015 12:46 Pears 14
4/8/2015 8:59 Oranges 52
4/10/2015 2:07 Apples 152
4/10/2015 18:10 Bananas 23
4/10/2015 2:40 Strawberries 98
```

لا يحتوي الصف الأول من الملف `example.csv` على أيّ نص لعناوين الأعمدة، لذا أنشأنا عناوينا الخاصة `'time'` و `'name'` و `'amount'`.

تستخدم كائنات `DictWriter` أيضًا القواميس لإنشاء ملفات CSV. إذا أردتَ أن يحتوي ملفك على صف الترويسات، فاكتب هذا الصف من خلال استدعاء الدالة `writeheader()`، وإلّا فتخطى استدعاء هذه الدالة لحذف صف الترويسات من الملف. اكتب بعد ذلك كل صف من صفوف الملف CSV باستخدام استدعاء التابع `writerow()` مع تمرير قاموسٍ يستخدم الترويسات بوصفها مفاتيحًا ويحتوي على البيانات المراد كتابتها ضمن الملف.

```
>>> import csv
>>> outputFile = open('output.csv', 'w', newline='')
>>> outputDictWriter = csv.DictWriter(outputFile, ['Name', 'Pet',
'Phone'])
>>> outputDictWriter.writeheader()
>>> outputDictWriter.writerow({'Name': 'Alice', 'Pet': 'cat', 'Phone':
'555-
1234'})
20
>>> outputDictWriter.writerow({'Name': 'Bob', 'Phone': '555-9999'})
15
>>> outputDictWriter.writerow({'Phone': '555-5555', 'Name': 'Carol',
'Pet':
'dog'})
20
>>> outputFile.close()
```

يبدو الملف `output.csv` الذي تنشئه الشيفرة البرمجية السابقة كما يلي:

```
Name,Pet,Phone
Alice,cat,555-1234
Bob,,555-9999
Carol,dog,555-5555
```

لاحظ أن ترتيب أزواج القيمة-المفتاح key-value في القواميس التي مرّرتها إلى الدالة `writerow()` غير مهم، فهي مكتوبة بترتيب المفاتيح المعطاة إلى الدالة `DictWriter()`، فمثلًا لا يزال رقم الهاتف يظهر في آخر الخرج بالرغم من أنك مرّرتَ المفتاح `Phone` وقيمته قبل مفاتيح وقيم `Name` و `Pet` في الصف الرابع. لاحظ أيضًا أن أيّ مفاتيح مفقودة مثل `'Pet'` في `{'Name': 'Bob', 'Phone': '555-9999'}` ستكون ببساطة فارغة في ملف CSV.

16.2 تطبيق عملي: إزالة الترويسة من ملفات CSV

لنفترض أن لديك مهمة مملة تتمثل في إزالة السطر الأول من عدة مئات من ملفات CSV، إذ قد تدخل هذه الملفات في عملية آلية تتطلب البيانات فقط وليس الترويسات الموجودة في أعلى الأعمدة، حيث يمكنك فتح كل ملف في إكسل وحذف الصف الأول، ثم تعيد حفظ الملف، ولكن قد يستغرق ذلك ساعات، إذًا لنكتب برنامجًا ينفّذ هذه المهمة بدلًا من ذلك.

يجب أن يفتح البرنامج كل ملفٍ امتداده `.csv` في مجلد العمل الحالي، ويقرأ محتويات ملف CSV، ثم يعيد كتابة المحتويات بدون الصف الأول في ملف يحمل الاسم نفسه، مما يؤدي إلى وضع المحتويات الجديدة لملف CSV بدون الترويسات مكان المحتويات القديمة.

> **ملاحظة**: تأكد من إنشاء نسخة احتياطية للملفات أولًا عندما تكتب برنامجًا يعدّل الملفات، فأنت لا تريد مسح ملفاتك الأصلية عن طريق الخطأ في حالة عدم عمل البرنامج بالطريقة التي تتوقعها.

إليك الخطوات العامة التي سيطبّقها برنامجك:

1. البحث عن جميع ملفات CSV في مجلد العمل الحالي.
2. قراءة المحتويات الكاملة لكل ملف.
3. كتابة المحتويات مع تخطي السطر الأول في ملف CSV جديد.

ولكن سيحتاج برنامجك إلى تطبيق الخطوات التالية من ناحية الشيفرة البرمجية:

1. المرور على قائمة الملفات باستخدام التابع `os.listdir()` مع تخطي الملفات التي هي ليست ملفات CSV.

2. إنشاء كائن `reader` الخاص بوحدة CSV وقراءة محتويات الملف باستخدام السمة Attribute التي هي `line_num` لمعرفة السطر الذي يجب تخطيه.
3. إنشاء كائن `writer` الخاص بوحدة CSV وكتابة بيانات القراءة في الملف الجديد.

افتح نافذة محرّر جديدة لإنشاء ملف جديد واحفظه بالاسم `removeCsvHeader.py` لهذا المشروع.

16.2.1 الخطوة الأولى: المرور على جميع ملفات CSV

أول شيء يجب على برنامجك فعله هو المرور على قائمة جميع أسماء ملفات CSV الموجودة في مجلد العمل الحالي، لذا اجعل الملف `removeCsvHeader.py` يبدو كما يلي:

```
#! python3
# removeCsvHeader.py - إزالة الترويسات من جميع ملفات CSV الموجودة في مجلد العمل الحالي

import csv, os
os.makedirs('headerRemoved', exist_ok=True)

# المرور ضمن حلقة على جميع الملفات الموجودة في مجلد العمل الحالي
for csvFilename in os.listdir('.'):
    if not csvFilename.endswith('.csv'):
        ❶ continue    # تخطي الملفات التي ليست ملفات CSV

    print('Removing header from ' + csvFilename + '...')

    # قراءة ملف CSV مع تخطي الصف الأول

    # كتابة ملف CSV
```

يؤدي استدعاء التابع `os.makedirs()` إلى إنشاء المجلد `headerRemoved` الذي سنكتب فيه جميع ملفات CSV التي ليس لها ترويسات. ستقودك حلقة `for` التي نكررها على التابع `os.listdir('.')` إلى هذه النتيجة جزئيًا، ولكنها ستتكرر على جميع الملفات الموجودة في مجلد العمل، لذلك يجب إضافة بعض الشيفرة البرمجية في بداية الحلقة التي تتخطى أسماء الملفات التي لا تنتهي بالامتداد `.csv`. تجعل التعليمة `continue` ❶ حلقة `for` تنتقل إلى اسم الملف التالي عندما تصل إلى ملف ليس ملف CSV.

اطبع رسالةً توضّح ملف CSV الذي يعمل عليه البرنامج حتى يكون هناك بعض المخرجات أثناء تنفيذ البرنامج، ثم أضف بعض التعليقات لما يجب أن يفعله باقي البرنامج.

16.2.2 الخطوة الثانية: قراءة ملف CSV

لا يزيل البرنامج السطر الأول من ملف CSV، بل ينشئ نسخةً جديدة من ملف CSV بدون السطر الأول، وستحل النسخة محل الملف الأصلي لأن اسم ملف النسخة هو اسم الملف الأصلي نفسه.

سيحتاج برنامجك إلى طريقة لتعقّب المرور على الصف الأول حاليًا، لذا أضِف ما يلي إلى الملف `removeCsvHeader.py`:

```
#! python3
# removeCsvHeader.py - إزالة الترويسات من جميع ملفات CSV الموجودة في مجلد العمل الحالي
--snip--

  # قراءة ملف CSV مع تخطي الصف الأول
  csvRows = []
  csvFileObj = open(csvFilename)
  readerObj = csv.reader(csvFileObj)
  for row in readerObj:
    if readerObj.line_num == 1:
      continue     # تخطي الصف الأول
    csvRows.append(row)
  csvFileObj.close()
  # كتابة ملف CSV
```

يمكن استخدام السمة `line_num` الخاصة بكائن `reader` لتحديد السطر الموجود في ملف CSV الذي يقرأه حاليًا، حيث توجد حلقة `for` أخرى للمرور على الصفوف التي يعيدها كائن `reader` الخاص بوحدة `CSV`، وستُلحَق جميع الصفوف باستثناء الصف الأول بالقائمة `csvRows`.

تتحقق الشيفرة البرمجية السابقة مما إذا كان `readerObj.line_num` مضبوطًا على القيمة `1` أثناء تكرار حلقة `for` على كل صف. إذا كان ذلك صحيحًا، فستُنفَّذ تعليمة `continue` للانتقال إلى الصف التالي دون إلحاقه بالقائمة `csvRows`، وستكون قيمة الشرط `False` دائمًا بالنسبة لكل صفٍ لاحق، وسيُلحَق هذا الصف بالقائمة `csvRows`.

16.2.3 الخطوة الثالثة: كتابة ملف CSV بدون الصف الأول

أصبحت القائمة `csvRows` تحتوي على كافة الصفوف باستثناء الصف الأول، ويجب الآن كتابة هذه القائمة في ملف CSV الموجود في المجلد `headerRemoved`. أضِف ما يلي إلى الملف `removeCsvHeader.py`:

```
#! python3
# removeCsvHeader.py - إزالة الترويسات من جميع ملفات CSV الموجودة في مجلد العمل الحالي
--snip--


# المرور ضمن حلقة على جميع الملفات الموجودة في مجلد العمل الحالي
❶ for csvFilename in os.listdir('.'):
    if not csvFilename.endswith('.csv'):
        continue    # تخطي الملفات التي ليست ملفات CSV
    --snip--

    # كتابة ملف CSV
    csvFileObj = open(os.path.join('headerRemoved', csvFilename), 'w',
                      newline='')
    csvWriter = csv.writer(csvFileObj)
    for row in csvRows:
        csvWriter.writerow(row)
    csvFileObj.close()
```

يكتب الكائن `writer` الخاص بوحدة `CSV` القائمة الناتجة في ملف CSV موجود ضمن المجلد `headerRemoved` باستخدام المتغير `csvFilename` الذي استخدمناه أيضًا في كائن `reader` الخاص بوحدة `CSV`، مما يؤدي إلى الكتابة فوق الملف الأصلي. نمر بعد إنشاء الكائن `writer` على القوائم الفرعية المُخزَّنة في القائمة `csvRows` ونكتب كل قائمة فرعية في الملف. تنتقل حلقة `for` الخارجية ❶ إلى اسم الملف التالي من `os.listdir('.')` بعد تنفيذ الشيفرة البرمجية، وسيكتمل البرنامج عند الانتهاء من تلك الحلقة.

اختبر برنامجك من خلال تنزيل الملف المضغوط removeCsvHeader.zip وفك ضغطه في مجلدٍ ما، ثم شغّل البرنامج `removeCsvHeader.py` في هذا المجلد، وسيكون الخرج كما يلي:

```
Removing header from NAICS_data_1048.csv...
Removing header from NAICS_data_1218.csv...
--snip--
Removing header from NAICS_data_9834.csv...
Removing header from NAICS_data_9986.csv…
```

يجب أن يطبع هذا البرنامج اسم ملفٍ في كل مرة يزيل فيها السطر الأول من ملف CSV.

16.2.4 أفكار لبرامج مماثلة

تشبه البرامجُ التي يمكنك كتابتها لملفات CSV أنواعَ البرامج التي يمكنك كتابتها لملفات إكسل، لأنهما ملفات جداول بيانات، حيث يمكنك كتابة برامج تنفّذ ما يلي:

- مقارنة البيانات بين صفوف مختلفة في ملف CSV واحد أو بين ملفات CSV متعددة.
- نسخ بيانات محددة من ملف CSV إلى ملف إكسل أو العكس.
- التحقق من وجود بيانات غير صالحة أو أخطاء في تنسيق ملفات CSV وتنبيه المستخدم بهذه الأخطاء.
- قراءة البيانات من ملف CSV بوصفها دخلًا لبرامج بايثون الخاصة بك.

16.3 JSON وواجهات برمجة التطبيقات API

يُعَد ترميز الكائنات باستعمال جافاسكربت JavaScript Object Notation -أو JSON اختصارًا- طريقةً شائعة لتنسيق البيانات بوصفها سلسلة نصية واحدة يمكن أن يقرأها البشر، وهو الطريقة الأصيلة التي تكتب بها برامج جافاسكربت هياكلَ البيانات الخاصة بها وتشبه ما ستنتجه دالة `pprint()` في بايثون. لست بحاجةٍ لمعرفة لغة جافاسكربت لتتمكّن من التعامل مع البيانات المكتوبة بتنسيق JSON.

إليك مثال للبيانات المكتوبة بتنسيق JSON:

```
{"name": "Zophie", "isCat": true,
 "miceCaught": 0, "napsTaken": 37.5,
 "felineIQ": null}
```

تُعَد معرفة JSON أمرًا مفيدًا، لأن العديد من مواقع الويب تقدم محتوى JSON بوصفه وسيلةً للبرامج للتفاعل مع موقع الويب، ويُعرَف ذلك بتوفير واجهة برمجة تطبيقات Application Programming

Interface -أو API اختصارًا. يشبه الوصول إلى واجهة برمجة التطبيقات الوصولَ إلى أيّ صفحة ويب أخرى باستخدام عنوان URL، ولكن الفرق بينهما هو أن البيانات التي تعيدها واجهة برمجة التطبيقات تكون منسَّقة لتفهمها الأجهزة باستخدام JSON مثلًا، إذ ليس من السهل أن يقرأ البشر واجهات برمجة التطبيقات.

تتيح العديد من مواقع الويب بياناتها بتنسيق JSON، حيث توفر فيسبوك Facebook وتويتر Twitter وياهو Yahoo وجوجل Google وتمبلر Tumblr وويكيبيديا Wikipedia وفليكر Flickr و Data.gov وريديت Reddit و IMDb وروتن توميتوز Rotten Tomatoes ولينكد إن LinkedIn والعديد من المواقع الشهيرة الأخرى واجهات برمجة تطبيقات لتستخدمها البرامج. تتطلب بعض هذه المواقع التسجيل الذي يكون مجانيًا دائمًا، ويجب أن تعثر على توثيق عناوين URL التي يحتاج برنامجك إلى طلبها للحصول على البيانات التي تريدها، بالإضافة إلى التنسيق العام لهياكل بيانات JSON المُعادة. يجب توفير هذا التوثيق من خلال أيّ موقع يقدم واجهة برمجة التطبيقات، حيث إذا توافرت صفحة للمطورين "Developers"، فابحث عن التوثيق هناك.

يمكنك كتابة البرامج التي تنفّذ ما يلي باستخدام واجهات برمجة التطبيقات:

- استخلاص البيانات الخام من المواقع، حيث يكون الوصول إلى واجهات برمجة التطبيقات أسهل من تنزيل صفحات الويب وتحليل شيفرة HTML باستخدام مكتبة `Beautiful Soup`.
- تنزيل المنشورات الجديدة تلقائيًا من أحد حساباتك على شبكة التواصل الاجتماعي ونشرها على حسابٍ آخر، فمثلًا يمكنك أخذ منشوراتك على تمبلر ونشرها على فيسبوك.
- إنشاء "موسوعة أفلام" لمجموعة أفلامك الشخصية من خلال سحب البيانات من IMDb و Rotten Tomatoes وويكيبيديا ووضعها في ملف نصي واحد على حاسوبك.

> **ملاحظة**: يمكنك رؤية بعض الأمثلة على واجهات برمجة تطبيقات JSON في الموارد الموجودة على موقع nostarch.

لا تعد JSON الطريقة الوحيدة لتنسيق البيانات في سلسلة نصية يمكن أن يقرأها البشر، إذ توجد العديد من اللغات الأخرى بما في ذلك لغات XML (لغة التوصيف الموسَّعة eXtensible Markup Language) و TOML (أو Tom's Obvious, Minimal Language) و YML (أو Yet another Markup Language) و INI (أو Initialization) وتنسيقات ASN.1 القديمة (Abstract Syntax Notation One)، حيث توفر جميع هذه اللغات هيكلًا لتمثيل البيانات بوصفها نصًا يمكن أن يقرأه البشر. لن نغطّي هذه اللغات في هذا الفصل، لأن تنسيق JSON أصبح التنسيق البديل الأكثر استخدامًا على نطاق واسع، ولكن توجد وحدات بايثون خارجية يمكنها التعامل معها بسهولة.

16.4 وحدة json

تتعامل وحدة `json` في بايثون مع جميع تفاصيل الترجمة بين سلسلة نصية تحتوي على بيانات JSON وقيم بايثون الخاصة بالدوال `json.loads()` و `json.dumps()`. لا يمكن لتنسيق JSON تخزين كل أنواع قيم بايثون، إذ يمكن أن يحتوي على قيم لأنواع بياناتٍ مُحدَّدة فقط وهي: السلاسل النصية Strings والأعداد الصحيحة Integers والأعداد العشرية Floats والقيم المنطقية Booleans والقوائم Lists والقواميس Dictionaries والنوع `NoneType`. كما لا يمكن لتنسيق JSON أن يمثل كائنات خاصة بلغة بايثون مثل كائنات `File` أو كائنات `reader` أو `writer` الخاصة بوحدة `CSV` أو كائنات `Regex` أو كائنات `WebElement` الخاصة بالوحدة `Selenium`.

16.4.1 قراءة بيانات JSON باستخدام الدالة ()loads

يمكنك ترجمة سلسلة نصية تحتوي على بيانات JSON إلى قيمة في لغة بايثون من خلال تمريرها إلى الدالة `json.loads()`، حيث يعني اسم هذه الدالة loads تحميل سلسلة نصية "load string" ولا يعني مجموعة التحميلات "loads". لندخِل الآن ما يلي في الصدفة التفاعلية Interactive Shell:

```
>>> stringOfJsonData = '{"name": "Zophie", "isCat": true,
"miceCaught": 0,
"felineIQ": null}'
>>> import json
>>> jsonDataAsPythonValue = json.loads(stringOfJsonData)
>>> jsonDataAsPythonValue
{'isCat': True, 'miceCaught': 0, 'name': 'Zophie', 'felineIQ': None}
```

نستورد أولًا الوحدة `json`، ثم يمكننا استدعاء الدالة `loads()` وتمرير سلسلة نصية من بيانات JSON إليها، حيث تستخدم سلاسل JSON النصية دائمًا علامات اقتباس مزدوجة، وتعيد هذه الدالة البيانات بوصفها قاموس بايثون. تُعَد قواميس بايثون غير مرتبة، لذا يمكن أن تظهر أزواج مفتاح-قيمة بترتيب مختلف عند طباعة `jsonDataAsPythonValue`.

16.4.2 كتابة بيانات JSON باستخدام الدالة ()dumps

تترجم الدالة `json.dumps()` قيمة بايثون إلى سلسلة نصية من البيانات بتنسيق JSON، حيث يعني اسم هذه الدالة dumps تفريغ سلسلة نصية "dump string" ولا يعني الجمع "dumps").

لندخِل الآن ما يلي في الصدفة التفاعلية:

```
>>> pythonValue = {'isCat': True, 'miceCaught': 0, 'name': 'Zophie',
'felineIQ': None}
>>> import json
>>> stringOfJsonData = json.dumps(pythonValue)
>>> stringOfJsonData
'{"isCat": true, "felineIQ": null, "miceCaught": 0, "name":
"Zophie" }'
```

يمكن أن يكون نوع القيمة أحد أنواع بيانات بايثون الأساسية فقط وهي: قاموس أو قائمة أو عدد صحيح أو عدد عشري أو سلسلة نصية أو قيمة منطقية أو None.

16.5 تطبيق عملي: جلب بيانات الطقس الحالية

يبدو التحقق من الطقس أمرًا بسيطًا إلى حد ما، حيث يمكنك فتح متصفح الويب الخاص بك والنقر على شريط العناوين، ثم كتابة عنوان URL لموقع ويب خاص بالطقس أو البحث عن موقع ثم النقر على الرابط، وانتظار تحميل الصفحة، والاطلاع على جميع الإعلانات وإلخ. هناك الكثير من الخطوات المملة التي يمكنك تخطيها إذا كان لديك برنامج ينزّل توقعات الطقس للأيام القليلة القادمة ويطبعها ضمن نصٍ عادي، حيث يستخدم هذا البرنامج الوحدة requests لتنزيل البيانات من الويب، فالخطوات العامة التي يطبّقها هذا البرنامج هي ما يلي:

1. قراءة الموقع المطلوب من سطر الأوامر.
2. تنزيل بيانات JSON الخاصة بالطقس من الموقع OpenWeatherMap.org.
3. تحويل سلسلة بيانات JSON إلى هيكل بيانات بايثون.
4. طباعة حالة الطقس لهذا اليوم واليومين القادمين.

لذا ستطبّق الشيفرة البرمجية الخطوات التالية:

1. ضم السلاسل النصية إلى القائمة sys.argv للحصول على الموقع.
2. استدعاء الدالة requests.get() لتنزيل بيانات الطقس.
3. استدعاء الدالة json.loads() لتحويل بيانات JSON إلى هيكل بيانات بايثون.
4. طباعة توقعات الطقس.

افتح نافذة محرّر جديدة لإنشاء ملف جديد لهذا المشروع واحفظه بالاسم `getOpenWeather.py`، ثم انتقل إلى الموقع OpenWeatherMap في متصفحك وسجّل فيه على حساب مجاني للحصول على مفتاح API، ويُسمَّى أيضًا معرّف التطبيق app ID، والذي يمثل رمز سلسلة نصية يبدو مثل الرمز `'30144aba38018987d84710d0e319281e'` بالنسبة لخدمة OpenWeatherMap. لا حاجة للدفع مقابل هذه الخدمة إلّا إذا أردتَ إجراء أكثر من 60 استدعاء لواجهة برمجة التطبيقات في الدقيقة. حافظ على سرية مفتاح API، إذ يمكن لأيّ شخص يعرفه كتابة سكربتات تأخذ من حصة الاستخدام الخاصة بحسابك.

16.5.1 الخطوة الأولى: الحصول على الموقع من وسيط سطر الأوامر

يأتي دخل هذا البرنامج من سطر الأوامر، لذا اجعل برنامج `getOpenWeather.py` كما يلي:

```
#! python3
# getOpenWeather.py - طباعة الطقس لموقعٍ ما من سطر الأوامر
APPID = 'YOUR_APPID_HERE'
import json, requests, sys

# حساب الموقع من وسطاء سطر الأوامر
if len(sys.argv) < 2:
  print('Usage: getOpenWeather.py city_name, 2-letter_country_code')
  sys.exit()
location = ' '.join(sys.argv[1:])
# تنزيل بيانات JSON من واجهة برمجة تطبيقات OpenWeatherMap.org
# تحميل بيانات JSON في متغير بايثون
```

تُخزَّن وسطاء سطر الأوامر ضمن القائمة `sys.argv` في لغة بايثون، ويجب ضبط المتغير `APPID` على قيمة مفتاح API الخاص بحسابك، إذ ستفشل طلباتك لخدمة الطقس بدون هذا المفتاح، ثم سيتحقق البرنامج من وجود أكثر من وسيط سطر أوامر بعد سطر Shebang (الذي يبدأ بالرمز `#!`) والتعليمة `import`. تذكّر أن القائمة `sys.argv` تحتوي دائمًا على عنصر واحد على الأقل `sys.argv[0]`، والذي يحتوي على اسم ملف سكربت بايثون. إذا كان هناك عنصر واحد فقط في القائمة، فهذا يعني أن المستخدم لم يقدّم موقعًا في سطر الأوامر، وستُعرَض رسالة الاستخدام "usage" للمستخدم قبل انتهاء البرنامج.

تتطلب خدمة OpenWeatherMap تنسيق الاستعلام بالشكل: اسم المدينة ثم فاصلة ثم رمز البلد المكون من حرفين مثل الرمز "US" للولايات المتحدة الأمريكية، حيث يمكنك العثور على قائمة بهذه الرموز على ويكيبيديا. يعرض هذا السكربت الطقس للمدينة الأولى المُدرَجة في نص JSON المُعاد، ولكن ستُضمَّن جميع

المدن التي لها الاسم نفسه مثل مدينة بورتلاند Portland في ولاية أوريجون ومدينة بورتلاند بولاية ماين، بالرغم من أن نص JSON سيتضمّن معلومات خطوط الطول والعرض للتمييز بين هذه المدن.

تُقسَم وسطاء سطر الأوامر بناءً على الفراغات، حيث سيجعل وسيط سطر الأوامر `San Francisco, US` القائمة `sys.argv` تحتوي على `['getOpenWeather.py', 'San', 'Francisco,', 'US']`، لذلك استدعِ التابع `join()` لضم جميع السلاسل النصية باستثناء السلسلة النصية الأولى في القائمة `sys.argv`، وخزّن هذه السلسلة النصية الناتجة عن الضم في متغير اسمه `location`.

16.5.2 الخطوة الثانية: تنزيل بيانات JSON

يوفّر موقع OpenWeatherMap.org معلومات الطقس بتنسيق JSON في الزمن الحقيقي، ولكن يجب عليك أولًا التسجيل في الموقع للحصول على مفتاح API مجاني، حيث يُستخدَم هذا المفتاح لتقييد عدد مرات تقديم الطلبات على الخادم، مما يؤدي إلى إبقاء تكاليف حيز النطاق التراسلي منخفضة.

يجب أن ينزّل برنامجك الصفحة الموجودة على الرابط الآتي:

```
https://api.openweathermap.org/data/2.5/forecast/daily?
q=<Location>&cnt=3&APPID=<APIkey>
```

حيث `<Location>` هو اسم المدينة التي تريد الطقس فيها و `<API key>` هو مفتاح API الشخصي الخاص بك. أضِف ما يلي إلى برنامج `getOpenWeather.py`:

```
#! python3

# getOpenWeather.py - طباعة الطقس لموقعٍ ما من سطر الأوامر

--snip--


# تنزيل بيانات JSON من واجهة برمجة تطبيقات OpenWeatherMap.org

url ='https://api.openweathermap.org/data/2.5/forecast/daily?
q=%s&cnt=3&APPID=%s ' % (location,

APPID)

response = requests.get(url)

response.raise_for_status()

# ألغِ التعليق لرؤية نص JSON الخام:

#print(response.text)

# تحميل بيانات JSON في متغير بايثون
```

يأتي الموقع `location` من وسطاء سطر الأوامر، ويمكننا إنشاء عنوان URL الـذي نريـد الوصـول إليـه من خلال استخدام العنصر البديل `%s` وإدراج أيّ سلسلة نصية مُخزَّنة في `location` في ذلك المكان من السلسـلة النصية التي تمثّل عنوان URL.

نخزّن النتيجـة في المتغـير `url` الـذي نمـرّره إلى الدالـة `requests.get()`، حيث يعيـد اسـتدعاء الدالـة `requests.get()` الكائن `Response`، والذي يمكنك التحقق من وجود أخطاء فيـه من خلال اسـتدعاء الدالـة `raise_for_status()`. إن لم يظهر أيّ استثناء، فسيكون النص المُنزَّل موجودًا في `response.text`.

16.5.3 الخطوة الثالثة: تحميل بيانات JSON وطباعة الطقس

يحتوي المتغير العضو `response.text` على سلسلة نصية كبيرة من البيانات المُنسَّـقة بتنسـيق JSON، حيث يمكنك تحويلها إلى قيمة بايثون من خلال استدعاء الدالة `json.loads()`، وسـتبدو بيانـات JSON كمـا الحال في ما يلي:

```
{'city': {'coord': {'lat': 37.7771, 'lon': -122.42},
    'country': 'United States of America',
    'id': '5391959',
    'name': 'San Francisco',
    'population': 0},
 'cnt': 3,
 'cod': '200',
 'list': [{'clouds': 0,
    'deg': 233,
    'dt': 1402344000,
    'humidity': 58,
    'pressure': 1012.23,
    'speed': 1.96,
    'temp': {'day': 302.29,
       'eve': 296.46,
       'max': 302.29,
```

```
        'min': 289.77,
        'morn': 294.59,
        'night': 289.77},
    'weather': [{'description': 'sky is clear',
      'icon': '01d',
--snip--
```

يمكنك رؤية هذه البيانات من خلال تمرير المتغير `weatherData` إلى الدالة `pprint.pprint()`، وقد ترغب في الاطلاع على موقع openweathermap.org لمزيد من التوثيق الذي يوضّح معنى هذه الحقول، فمثلًا سيخبرك التوثيق عبر الإنترنت أن القيمة `302.29` بعد `'day'` هي درجة الحرارة أثناء النهار بواحدة الكلفن وليست بواحدة فهرنهايت أو الدرجة المئوية.

يوجد وصف الطقس الذي تريده بعد `'main'` و `'description'`، لذا أضِف ما يلي إلى برنامج `getOpenWeather.py` لطباعته بدقة:

```
! python3
# getOpenWeather.py - طباعة الطقس لموقعٍ ما من سطر الأوامر
--snip--


# تحميل بيانات JSON في متغير بايثون
weatherData = json.loads(response.text)


# طباعة وصف الطقس
❶ w = weatherData['list']
print('Current weather in %s:' % (location))
print(w[0]['weather'][0]['main'], '-', w[0]['weather'][0]
['description'])
print()
print('Tomorrow:')
```

```
 print(w[1]['weather'][0]['main'], '-', w[1]['weather'][0]
['description'])
 print()
 print('Day after tomorrow:')
 print(w[2]['weather'][0]['main'], '-', w[2]['weather'][0]
['description'])
```

لاحظ كيف تخزّن الشيفرة البرمجية السابقة المتغير `weatherData['list']` في المتغير `w` ليوفر عليك بعض الجهد عند الكتابة ❶، حيث يمكنك استخدام `w[0]` و `w[1]` و `w[2]` لاسترداد القواميس الخاصة بطقس اليوم والغد وبعد الغد على التوالي، ويحتوي كلّ قاموس من هذه القواميس على مفتاح `'weather'` الذي يحتوي على قيمة من النوع قائمة، ويهمنا منها عنصر القائمة الأول عند الفهرس `0`، وهو قاموس متداخل يحتوي على عدة مفاتيح أخرى. نطبع القيم المخزنة في المفتاحين `'main'` و `'description'`، حيث نفصل بينها بشرطة واصلة.

سيبدو الخرج كما يلي عند تشغيل هذا البرنامج باستخدام وسيط سطر الأوامر `getOpenWeather.py San Francisco, CA`:

```
Current weather in San Francisco, CA:
Clear - sky is clear

Tomorrow:
Clouds - few clouds

Day after tomorrow:
Clear - sky is clear
```

16.5.4 أفكار لبرامج مماثلة

يمكن أن يشكّل الوصول إلى بيانات الطقس الأساس الذي نبني عليه العديد من أنواع البرامج الأخرى، حيث يمكنك إنشاء برامج مماثلة لتنفيذ ما يلي:

- جمع تنبؤات الطقس للعديد من مواقع التخييم أو مسارات المشي لمسافات طويلة لمعرفة أيّ منها سيكون فيها الطقس الأفضل.
- جدولة برنامج للتحقق من الطقس بانتظام وإرسال تنبيه عند حدوث الصقيع إذا أردتَ نقل نباتاتك إلى الداخل، حيث سنوضّح لاحقًا الجدولة وكيفية إرسال البريد الإلكتروني.

- سحب بيانات الطقس من مواقع متعددة لإظهارها دفعة واحدة، أو حساب وإظهار متوسط توقعات الطقس المتعددة.

16.6 أسئلة للتدريب

1. ما هي بعض الميزات التي تحتوي عليها جداول بيانات إكسل ولا تتوفر في جداول بيانات CSV؟
2. ما الذي يمكنك تمريره إلى الدالتين `csv.reader()` و `csv.writer()` من أجل إنشاء كائنات `reader` و `writer`؟
3. ما هي الأوضاع التي يجب فيها فتح كائنات `File` لكائنات `reader` و `writer`؟
4. ما هو التابع الذي يأخذ وسيط القائمة ويكتبها في ملف CSV؟
5. ما الذي تفعله وسطاء الكلمات المفتاحية `delimiter` و `lineterminator`؟
6. ما هي الدالة التي تأخذ سلسلة نصية من بيانات JSON وتعيد هيكل بيانات بايثون؟
7. ما هي الدالة التي تأخذ هيكل بيانات بايثون وتعيد سلسلة نصية من بيانات JSON؟

16.7 مشاريع للتدريب

سنعمل في هذا الجزء من الفصل على مشروع يحول ملف Exel إلى ملف CSV

16.7.1 محوّل ملف Exel إلى ملف CSV

يمكن لإكسل حفظ جدول بيانات في ملف CSV باستخدام بضع نقرات بالماوس، ولكن إذا أردتَ تحويل مئاتٍ من ملفات إكسل إلى ملفات CSV، فسيستغرق الأمر ساعات من النقر، لذا استخدم وحدة `openpyxl` لكتابة برنامجٍ يقرأ جميع ملفات إكسل الموجودة في مجلد العمل الحالي ويخرجها بوصفها ملفات CSV.

قد يحتوي ملف إكسل واحد على أوراق متعددة، لذا يجب إنشاء ملف CSV واحد لكل ورقة، ويجب أن تكون أسماء ملفات CSV هي `<excel filename>_<sheet title>.csv`، حيث يكون `<excel filename>` هو اسم ملف إكسل بدون امتداد الملف مثل `'spam_data'` وليس `'spam_data.xlsx'` ويكون `<sheet title>` هو السلسلة النصية التي تأتي من المتغير `title` الخاص بكائن `Worksheet`.

سيتضمن هذا البرنامج العديد من حلقات `for` المتداخلة، وسيبدو هيكل هذا البرنامج كما يلي:

```
for excelFile in os.listdir('.'):
    # تخطي الملفات التي ليست ملفات xlsx، وتحميل كائن المصنف workbook
    for sheetName in wb.get_sheet_names():
```

```
    # المرور ضمن حلقة على كل ورقة في المصنف
    sheet = wb.get_sheet_by_name(sheetName)
    # إنشاء اسم ملف CSV من اسم ملف إكسل وعنوان الورقة
    # إنشاء كائن csv.writer لملف CSV
    # المرور ضمن حلقة على كل صف في الورقة
    for rowNum in range(1, sheet.max_row + 1):
      rowData = []     # إلحاق كل خلية بهذه القائمة
      # المرور ضمن حلقة على كل خلية في الصف
      for colNum in range(1, sheet.max_column + 1):
        # إلحاق بيانات كل خلية بالقائمة rowData
      # كتابة القائمة rowData في ملف CSV
    csvFile.close()
```

نزّل الملف المضغوط excelSpreadsheets.zip وفك ضغط جداول البيانات في المجلد نفسه الموجود فيه برنامجك، حيث يمكنك استخدام جداول البيانات هذه كملفات لاختبار البرنامج عليها.

16.8 الخلاصة

تُعد CSV و JSON من تنسيقات النصوص العادية الشائعة لتخزين البيانات، حيث يسهُل على البرامج تحليلها مع بقاء قابليتها للقراءة، لذا تُستخدَم لجداول البيانات أو بيانات تطبيقات الويب البسيطة. تبسّط وحدتا `csv` و `json` عملية القراءة والكتابة في ملفات CSV و JSON بصورة كبيرة.

تعلّمنا سابقًا كيفية استخدام بايثون لتحليل معلومات مجموعة واسعة من تنسيقات الملفات، فإحدى المهام الشائعة هي أخذ البيانات من مجموعة متنوعة من التنسيقات وتحليلها للحصول على المعلومات المُحدَّدة التي تحتاجها، وتكون هذه المهام مُحدَّدة لدرجة أن البرمجيات التجارية لن تفيدك في إنجازها على النحو الأمثل، لذا يمكنك جعل حاسوبك يتعامل مع كميات كبيرة من البيانات المُقدَّمة بهذه التنسيقات من خلال كتابة السكربتات الخاصة بك.

سنبتعد في الفصول اللاحقة عن تنسيقات البيانات وسنتعلّم كيفية جعل برامجك يتواصل معك من خلال إرسال رسائل البريد الإلكتروني والرسائل النصية.

دورة إدارة تطوير المنتجات

تعلم تحويل أفكارك لمنتجات ومشاريع حقيقية بدءًا من دراسة السوق
وتحليل المنافسين وحتى إطلاق منتج مميز وناجح

التحق بالدورة الآن

17. الوقت وجدولة المهام بلغة بايثون

قد تشغّل برامجك أثناء جلوسك أمام الحاسوب، ولكنك ستفضّل تشغيلها دون إشرافك المباشر، إذ يمكن لساعة حاسوبك جدولة البرامج لتشغيل الشيفرة البرمجية في وقت وتاريخ محدَّد أو على فترات زمنية منتظمة، فمثلًا يمكن أن يستخلص Scrape برنامجك البيانات من موقع ويب كل ساعة للتحقق من التغييرات أو يجري مهمةً تستخدم وحدة المعالجة المركزية بصورة كبيرة في الساعة 4 صباحًا أثناء نومك، حيث توفر وحدتا `time` و`datetime` في لغة بايثون هذه الوظائف.

يمكنك أيضًا كتابة البرامج التي تشغّل Launch برامجًا أخرى وفقًا لجدولٍ زمني باستخدام وحدات `subprocess` و `threading`، فأسرع طريقة لكتابة البرامج في أغلب الأحيان هي الاستفادة من التطبيقات التي كتبها أشخاص آخرون مسبقًا.

17.1 الوحدة time

تُضبَط ساعة نظام حاسوبك على تاريخٍ ووقت ومنطقة زمنية محددة، حيث تسمح الوحدة `time` المُدمَجة لبرامج بايثون الخاصة بك بقراءة ساعة النظام للوقت الحالي، وتعد الدالتان `time.time()` و`time.sleep()` الأكثر فائدة في هذه الوحدة.

17.1.1 الدالة time.time()

توقيت يونكس Unix Epoch هو مرجع زمني شائع الاستخدام في البرمجة، وهو الساعة 12 صباحًا في 1 من الشهر الأول من عام 1970 بالتوقيت العالمي المنسق Coordinated Universal Time -أو UTC اختصارًا، حيث تعيد الدالة `time.time()` عدد الثواني منذ تلك اللحظة التي تمثّل توقيت يونكس بوصفها قيمةً عشرية Float، والتي تُعَد مجرد عددٍ مع فاصلة عشرية، ويسمى هذا العدد الذي تعيده الدالة `time.time()` بعلامة يونكس الزمنية Epoch Timestamp أو البصمة الزمنية.

أدخِل ما يلي مثلًا في الصدفة التفاعلية Interactive Shell:

```
>>> import time
>>> time.time()
1712718940.0904896
```

استدعينا الدالة `time.time()` صبيحة عيد الفطر 10 من الشهر 4 من عام 2022 في الساعة 06:15 مساءً بتوقيت مكة المكرمة، وتكون القيمة المُعادة هي عدد الثواني التي مرّت بين توقيت يونكس ولحظة استدعاء الدالة `time.time()`.

يمكن استخدام علامات يونكس الزمنية لفحص أداء Profile الشيفرة البرمجية، أي قياس المدة التي يستغرقها تشغيل جزء من هذه الشيفرة البرمجية. إذا استدعيتَ الدالة `time.time()` في بداية مقطع الشيفرة البرمجية الذي تريد قياس المدة التي يستغرقها تشغيله وفي نهايته مرةً أخرى، فيمكنك طرح العلامة الزمنية الأولى من الثانية لإيجاد الوقت المنقضي بين هذين الاستدعاءين. افتح تبويبًا جديدًا في محرّرك لإنشاء ملفٍ جديد وأدخِل البرنامج التالي مثلًا:

```
  import time
❶ def calcProd():
    # حساب حاصل ضرب أول 100,000 عدد
    product = 1
    for i in range(1, 100000):
      product = product * i
    return product
❷ startTime = time.time()
 prod = calcProd()
❸ endTime = time.time()
❹ print('The result is %s digits long.' % (len(str(prod))))
❺ print('Took %s seconds to calculate.' % (endTime - startTime))
```

نعرّف الدالة `calcProd()` ❶ للتكرار ضمن حلقة على الأعداد الصحيحة من 1 إلى 99,999 وإعادة ناتج ضربها.

نستدعي الدالة `time.time()` ونخزّنها في المتغير `startTime` ❷، ثم نستدعي الدالة `time.time()` مرةً أخرى بعد استدعاء الدالة `calcProd()` مباشرةً ونخزّنها في المتغير `endTime` ❸، ثم نطبع عدد أرقام حاصل الضرب الذي تعيده الدالة `calcProd()` ❹ والمدة التي استغرقها تشغيل هذه الدالة ❺.

احفظ البرنامج السابق بالاسم `calcProd.py` وشغّله، حيث سيبدو خرجه كما يلي:

```
The result is 456569 digits long.
Took 2.844162940979004 seconds to calculate.
```

> **ملاحظة**: هناك طريقة أخرى لفحص أداء شيفرتك البرمجية باستخدام الدالة `cProfile.run()` التي توفّر مستوًى أعلى من التفاصيل بدلًا من استخدام تقنية الدالة `time.time()` البسيطة. اطلّع على فصل قياس أداء وسرعة تنفيذ شيفرة بايثون لمزيدٍ من التفاصيل عن الدالة `cProfile.run()`.

تُعَد القيمة المُعادة من الدالة `time.time()` مفيدةً للحصول على عدد الثواني منذ توقيت يونكس إلى لحظة استدعائها بوصفها قيمةً عشرية ، ولكن لا يستطيع البشر قراءتها، لذا توجد دالة أخرى هي الدالة `time.ctime()` التي تعيد سلسلةً نصية تمثّل وصفًا للوقت الحالي. يمكنك أيضًا اختياريًا تمرير عدد الثواني منذ توقيت يونكس الذي تعيده الدالة `time.time()` إلى الدالة `time.ctime()` للحصول على قيمة السلسلة النصية التي تمثّل ذلك الوقت. لندخِل الآن ما يلي في الصدفة التفاعلية:

```
>>> import time
>>> time.ctime()
'Mon Jun 15 14:00:38 2023'
>>> thisMoment = time.time()
>>> time.ctime(thisMoment)
'Mon Jun 15 14:00:45 2023'
```

17.1.2 الدالة time.sleep()

إذا أردتَ إيقاف برنامجك مؤقتًا لفترة من الوقت، فاستدعِ الدالة `time.sleep()` ومرّر إليها عدد الثواني التي تريد أن يبقى فيها برنامجك متوقفًا مؤقتًا. إذًا لندخِل ما يلي في الصدفة التفاعلية:

```
>>> import time
>>> for i in range(3):
    ❶ print('Tick')
    ❷ time.sleep(1)
```

```
    ❸ print('Tock')
    ❹ time.sleep(1)
Tick
Tock
Tick
Tock
Tick
Tock
❺ >>> time.sleep(5)
```

تطبع حلقة `for` الكلمة `Tick` ❶، وتتوقف مؤقتًا لمدة ثانية واحدة ❷، ثم تطبع الكلمة `Tock` ❸، وتتوقف مؤقتًا لمدة ثانية واحدة ❹، ثم تطبع الكلمة `Tick`، وتتوقف مؤقتًا، وهكذا حتى طباعة كلٍّ من الكلمتين `Tick` و`Tock` ثلاث مرات.

تُعَد الدالة `time.sleep()` مُعطِّلة، أي أنها لن تعيد شيئًا ولن تحرّر برنامجك لتنفيذ شيفرة برمجية أخرى إلّا بعد انقضاء عدد الثواني التي مررتها إلى الدالة `time.sleep()`، فمثلًا إذا أدخلتَ `time.sleep(5)` ❺، فلن تظهر تعليمة المطالبة التالية (`>>>`) إلّا بعد مرور 5 ثوانٍ.

17.2 تقريب الأعداد

سترى في أغلب الأحيان عند التعامل مع الأوقات قيمًا عشرية تحتوي على العديد من الأعداد بعد الفاصلة العشرية، حيث يمكن تسهيل التعامل مع هذه القيم من خلال تقصيرها باستخدام الدالة `round()` المُدمَجة مع لغة بايثون، والتي تقرّب الأعداد العشرية إلى الدقة التي تحدّدها، حيث تدخِل العدد الذي تريد تقريبه، بالإضافة إلى وسيط ثانٍ اختياري يمثّل عدد الأعداد بعد الفاصلة العشرية التي تريد تقريبه إليها. إذا حذفتَ الوسيط الثاني، فستقرّب الدالة `round()` العدد إلى أقرب عدد صحيح كامل بدون فاصلة عشرية. لندخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import time
>>> now = time.time()
>>> now
1543814036.6147408
>>> round(now, 2)
```

```
1543814036.61
>>> round(now, 4)
1543814036.6147
>>> round(now)
1543814037
```

استوردنا الوحدة `time` وخزّنا الدالة `time.time()` في المتغير `now`، ثم استدعينا الدالة `round(now, 2)` لتقريب `now` إلى عددين بعد الفاصلة العشرية، واستدعينا الدالة `round(now, 4)` للتقريب إلى أربعة أعداد بعد الفاصلة العشرية، واستدعينا الدالة `round(now)` للتقريب إلى أقرب عدد صحيح.

17.3 تطبيق عملي: برنامج المؤقت الزمني الفائق Super Stopwatch

لنفترض أنك تريد تعقّب مقدار الوقت الذي تقضيه لإنجاز المهام المملة التي لم تؤتمتها بعد، فليس لديك مؤقت زمني فعلي ومن الصعب العثور على تطبيق مؤقت زمني مجاني لحاسوبك المحمول أو هاتفك الذكي غير مملوءٍ الإعلانات ولا يرسل نسخة من سجل متصفحك إلى المسوّقين، فمذكور أن هذا التطبيق يمكنه ذلك في اتفاقية ترخيصه التي وافقت عليها ولم تقرأها على الأغلب. إذًا لنكتب برنامج مؤقت زمني بسيط باستخدام لغة بايثون.

إليك الخطوات العامة التي سيطبقها برنامجك:

1. تعقّب مقدار الوقت المنقضي بين الضغطات على مفتاح `ENTER`، حيث تبدأ كل ضغطة "دورةً Lap" جديدة في المؤقت.
2. طباعة رقم الدورة والوقت الإجمالي ووقت الدورة.

يجب أن تطبق شيفرتك البرمجية الخطوات التالية:

1. إيجاد الوقت الحالي من خلال استدعاء الدالة `time.time()` وتخزينه بوصفه علامة زمنية في بداية البرنامج، وفي بداية كل دورة أيضًا.
2. الاحتفاظ بعدّاد دورات وزيادته في كل مرة يضغط فيها المستخدم على مفتاح `ENTER`.
3. حساب الوقت المنقضي من خلال طرح العلامات الزمنية.
4. معالجة الاستثناء `KeyboardInterrupt` حتى يتمكّن المستخدم من الضغط على الاختصار `CTRL-C` للإنهاء.

افتح تبويبًا جديدًا في محرّرك لإنشاء ملف جديد واحفظه بالاسم `stopwatch.py`.

17.3.1 الخطوة الأولى: إعداد البرنامج لتعقّب الأوقات

يحتاج برنامج المؤقت الزمني إلى استخدام الوقت الحالي، لذا يجب استيراد الوحدة `time`، ويجب أن يطبع برنامجك أيضًا بعض التعليمات المختصَرة للمستخدم قبل استدعاء الدالة `input()`، إذ يمكن أن يبدأ المـؤقت بعد أن يضغط المستخدم على مفتاح `ENTER`، ثم ستبدأ الشيفرة البرمجية في تعقّب أوقات الدورات.

أدخِل الآن الشيفرة البرمجية التالية في محرّر ملفاتك، واكتب تعليقاتٍ في النهاية، والتي تمثّل عناصر بديلة لما تبقى من شيفرتك البرمجية:

```
#! python3

# stopwatch.py - برنامج مؤقت زمني بسيط

import time


# عرض تعليمات البرنامج

print('Press ENTER to begin. Afterward, press ENTER to "click" the
stopwatch.
Press Ctrl-C to quit.')

input()                       # الضغط على مفتاح Enter للبدء

print('Started.')

startTime = time.time()       # الحصول على وقت بدء الدورة الأولى

lastTime = startTime

lapNum = 1

# البدء بتعقّب أوقات الدورات
```

كتبنا الشيفرة البرمجيـة لعـرض التعليمـات للمسـتخدم وبـدء الـدورة الأولى وتسـجيل الـوقت وضـبط عـدد الدورات على القيمة 1.

17.3.2 الخطوة الثانية: تعقّب أوقات الدورات وطباعتها

لنكتب الآن الشيفرة البرمجية لبدء دورات جديدة، وحساب المدة التي استغرقتها الدورة السـابقة، وحسـاب إجمالي الوقت المنقضي منذ بدء المؤقت الزمني، حيث سـنعرض وقت الـدورة والـوقت الإجمـالي ونزيـد عـدد الدورات بمقدار 1 عند كل دورة جديدة. إذًا أضِف الشيفرة البرمجية التالية إلى برنامجك:

```
#! python3

# stopwatch.py - برنامج مؤقت زمني بسيط
```

```
import time
--snip--

# البدء بتعقّب أوقات الدورات
❶ try:
  ❷ while True:
    input()
    ❸ lapTime = round(time.time() - lastTime, 2)
    ❹ totalTime = round(time.time() - startTime, 2)
    ❺ print('Lap #%s: %s (%s)' % (lapNum, totalTime, lapTime), end='')
      lapNum += 1
      lastTime = time.time() # إعادة ضبط وقت الدورة الأخيرة
❻ except KeyboardInterrupt:
    # معالجة الاستثناء Ctrl-C لمنع عرض رسالة الخطأ
    print('\nDone.')
```

إذا ضغط المستخدم على الاختصار CTRL-C لإيقاف المؤقت الزمني، فسيظهر الاستثناء KeyboardInterrupt، وسيتعطل البرنامج إذا لم يكن تنفيذه موجودًا ضمن تعليمة try، لذا غلّفنا هذا الجزء من البرنامج ضمن تعليمة try ❶، وسنعالج الاستثناء ضمن التعليمة except ❻، لذا ينتقل تنفيذ البرنامج إلى التعليمة except لطباعة الكلمة Done بدلًا من رسالة الخطأ KeyboardInterrupt عند الضغط على الاختصار CTRL-C ورفع الاستثناء. يكون التنفيذ ضمن حلقة لا نهائية ❷ حتى حدوث الاستثناء، حيث تستدعي هذه الحلقة الدالة input() وتنتظر حتى يضغط المستخدم على مفتاح ENTER لإنهاء الدورة. إذا انتهت الدورة، فسنحسب المدة التي استغرقتها الدورة من خلال طرح وقت بدء الدورة lastTime من الوقت الحالي time.time() ❸، ويمكننا حساب إجمالي الوقت المنقضي من خلال طرح وقت البدء الإجمالي للمؤقت الزمني startTime من الوقت الحالي ❹.

ستحتوي نتائج حسابات الوقت على العديد من الأعداد بعد الفاصلة العشرية مثل العدد 4.766272783279419، ولهذا استخدمنا الدالة round() من أجل تقريب القيمة العشرية إلى عددين في التعليمتين ❸ و ❹.

نطبع بعد ذلك رقم الدورة والوقت الإجمالي المنقضي ووقت الدورة ❺. يطبع المستخدم، الذي يضغط على مفتاح ENTER لاستدعاء الدالة `input()`، سطرًا جديدًا على الشاشة، لهذا مرّر الوسيط `end=''` إلى الدالة `print()` من أجل تجنب مضاعفة المسافة بين المخرجات. نتجهّز للدورة التالية بعد طباعة معلومات الدورة من خلال إضافة القيمة `1` إلى العدّاد `lapNum` ونضبط المتغير `lastTime` على الوقت الحالي الذي يمثّل وقت بدء الدورة التالية.

17.3.3 أفكار لبرامج مماثلة

يفتح تعقّب الوقت العديدَ من الاحتمالات أمام برامجك، حيث يمكنك كتابة هذه البرامج بنفسك بالرغم من أنه يمكنك تنزيل تطبيقاتٍ تنفّذ بعضًا منها، ولكن ستكون البرامج التي تكتبها بنفسك مجانية وغير مملوءة بالإعلانات والميزات عديمة الفائدة، لذلك جرّب كتابة برامج مماثلة تطبّق ما يلي:

- إنشاء تطبيق جدول حضور زمني Timesheet بسيط يسجّل متى كتبتَ اسم شخصٍ ما ويستخدم الوقت الحالي لتسجيل زمن دخوله أو خروجه.
- إضافة ميزة إلى برنامجك لعرض الوقت المنقضي منذ بدء عمليةٍ ما مثل عملية التنزيل التي تستخدم الوحدة `requests`.
- التحقّق خلال فترات زمنية متقطعة من مدة تشغيل البرنامج ومنح المستخدم فرصة لإلغاء المهام التي تستغرق وقتًا طويلًا.

17.4 الوحدة datetime

تُعَد الوحدة `time` مفيدةً للحصول على علامة يونكس الزمنية للعمل معها، ولكن إذا أردتَ عرض التاريخ بتنسيق أسهل أو إجراء العمليات الحسابية باستخدام التواريخ مثل معرفة التاريخ الذي كان قبل 205 يومًا أو التاريخ الذي سيكون بعد 123 يومًا من الآن، فيجب أن تستخدم الوحدة `datetime`.

تمتلك الوحدة `datetime` نوع البيانات `datetime` الخاص بها، حيث تمثل القيم التي نوعها `datetime` لحظةً محددة من الزمن. لندخِل الآن ما يلي في الصدفة التفاعلية:

```
>>> import datetime
❶ >>> datetime.datetime.now()
❷ datetime.datetime(2024, 2, 27, 11, 10, 49, 55, 53)
❸ >>> dt = datetime.datetime(2023, 10, 21, 16, 29, 0)
❹ >>> dt.year, dt.month, dt.day
```

```
  (2023, 10, 21)
❺ >>> dt.hour, dt.minute, dt.second
  (16, 29, 0)
```

يؤدي استدعاء الدالة `datetime.datetime.now()` ❶ إلى إعادة الكائن `datetime` ❷ للتاريخ والوقت الحاليين وفقًا لساعة حاسوبك، حيث يتضمن هذا الكائن السنة والشهر واليوم والساعة والدقيقة والثانية والميكروثانية للحظة الحالية. يمكنك أيضًا استرداد الكائن `datetime` للحظة معينة باستخدام الدالة `datetime.datetime()` ❸ وتمرير أعداد صحيحة إليها، والتي تمثّل السنة والشهر واليوم والساعة والدقيقة والثانية للحظة التي تريدها، وتُخزَّن هذه الأعداد الصحيحة في سمات Attributes خاصة بالكائن `datetime`، وهذه السمات هي `year` و `month` و `day` ❹ و `hour` و `minute` و `second` ❺.

يمكن تحويل علامة يونكس الزمنية إلى كائن `datetime` ، وذلك عبر استخدام الدالة `datetime.datetime.fromtimestamp()`، ويُحوَّل التاريخ والوقت الخاصين بكائن `datetime` إلى المنطقة الزمنية المحلية. إذًا أدخِل ما يلي في الصدفة التفاعلية:

```
>>> import datetime, time
>>> datetime.datetime.fromtimestamp(1000000)
datetime.datetime(1970, 1, 12, 5, 46, 40)
>>> datetime.datetime.fromtimestamp(time.time())
datetime.datetime(2023, 10, 21, 16, 30, 0, 604980)
```

يؤدي استدعاء الدالة `datetime.datetime.fromtimestamp()` وتمرير القيمة `1000000` إليها إلى إعادة كائن `datetime` للحظة التي تكون بعد 1,000,000 ثانية من توقيت يونكس، بينما يؤدي تمرير الدالة `time.time()`، التي هي بالأساس تمثّل علامة يونكس الزمنية خلال اللحظة الحالية، إلى الدالة `datetime.datetime.fromtimestamp()` إلى إعادة كائن `datetime` للحظة الحالية، إذ يفعل التعبيران `datetime.datetime.now()` و `datetime.datetime.fromtimestamp(time.time())` الشيء نفسه، حيث يعطيان كائن `datetime` للوقت الحالي.

يمكنك مقارنة كائنات `datetime` مع بعضها البعض باستخدام عوامل المقارنة لمعرفة أيّ منها يسبق الآخر، حيث يكون لكائن `datetime` الأحدث القيمة الأكبر. لندخِل الآن ما يلي في الصدفة التفاعلية:

```
❶ >>> endYear2023 = datetime.datetime(2023, 12, 31, 23, 59, 59)
❷ >>> newyears2024 = datetime.datetime(2024, 1, 1, 0, 0, 0)
```

```
>>> dec31_2023 = datetime.datetime(2023, 12, 31, 23, 59, 59)
❸ >>> endYear2023 == dec31_2023
True
❹ >>> endYear2023 > newyears2024
False
❺ >>> newyears2024 > endYear2023
True
>>> newyears2024 != dec31_2023
True
```

أنشـأنا كـائن `datetime` للحظـة الأخـيرة من 31 من الشـهر 12 من عـام 2023، وخزّنـاه في المتغـير `endYear2023` ❶، ثم أنشأنا كائن `datetime` للحظة الأولى من 1 من الشهر الأول من عام 2024، وخزّناه في المتغير `newyears2024` ❷، ثم أنشأنا كائنًا آخر لمنتصف ليل يوم 31 من الشهر 12 من عام 2023، وخزّناه في المتغير `dec31_2023`. تُظهِـر المقارنـة بين المتغـيرين `endYear2023` و `dec31_2023` أنهمـا متسـاويان ❸، وتُظهِـر مقارنـة المتغـيرين `newyears2024` و `endYear2023` أن `newyears2024` أكـبر (أو أحـدث) من `endYear2023` ❹ ❺.

17.4.1 نوع البيانات timedelta

توفر الوحدة `datetime` أيضًا نوع البيانات `timedelta` الذي يمثل مدة زمنية بدلًا من توفير لحظة زمنيـة. لندخِل الآن ما يلي في الصدفة التفاعلية:

```
❶ >>> delta = datetime.timedelta(days=11, hours=10, minutes=9,
seconds=8)
❷ >>> delta.days, delta.seconds, delta.microseconds
(11, 36548, 0)
>>> delta.total_seconds()
  986948.0
>>> str(delta)
'11 days, 10:09:08'
```

نستخدم الدالة `datetime.timedelta()` لإنشاء كائن `timedelta`، حيث تأخذ هذه الدالة وسطاء الكلمات المفتاحية Keyword Arguments التي هي `weeks` و `days` و `hours` و `minutes` و `seconds` و `milliseconds` و `microseconds`، ولا توجد وسطاء الكلمات المفتاحية `month` و `year`، إذ يُعَد الشهر أو السنة مقدارًا متغيرًا من الزمن اعتمادًا على شهرٍ أو سنة معينة. يحتوي الكائن `timedelta` على المدة الإجمالية المُمثَّلة بالأيام والثواني والميكروثانية، حيث تُخزَّن هذه الأعداد في السمات `days` و `seconds` و`microseconds`. يعيد التابع `total_seconds()` المدة بعدد الثواني فقط، بينما يؤدي تمرير كائن `timedelta` إلى الدالة `str()` إلى إعادة تمثيل سلسلة نصية مُنسَّقة جيدًا وقابلة للقراءة البشرية لهذا الكائن.

مرّرنا في المثال السابق وسطاء الكلمات المفتاحية إلى الدالة `datetime.timedelta()` لتحديد مدة 11 يومًا و10 ساعات و9 دقائق و8 ثوانٍ، وخزّنا كائن `timedelta` المُعاد في المتغير `delta` ❶. تخزِّن السمة `days` الخاصة بكائن `timedelta` القيمة `11`، وتخزن السمة `seconds` القيمة `36548` (أي 10 ساعات و9 دقائق و8 ثوانٍ من خلال التعبير عنها بالثواني) ❷. يخبرنا استدعاء الدالة `total_seconds()` أن 11 يومًا و10 ساعات و9 دقائق و8 ثوانٍ هي 986,948 ثانية، ويعيد تمرير كائن `timedelta` إلى الدالة `str()` سلسلةً نصية تشرح المدة بوضوح.

يمكن استخدام المعاملات الحسابية لإجراء عملية حسابية للتاريخ على قيم `datetime`، فمثلًا أدخِل ما يلي في الصدفة التفاعلية لحساب التاريخ بعد 1000 يوم من الآن:

```
>>> dt = datetime.datetime.now()
>>> dt
datetime.datetime(2018, 12, 2, 18, 38, 50, 636181)
>>> thousandDays = datetime.timedelta(days=1000)
>>> dt + thousandDays
datetime.datetime(2021, 8, 28, 18, 38, 50, 636181)
```

أنشأنا أولًا كائن `datetime` للحظة الحالية وخزّناه في المتغير `dt`، ثم أنشأنا كائن `timedelta` لمدة 1000 يوم وخزّناه في المتغير `thousandDays`، ثم جمعنا `dt` و `thousandDays` للحصول على كائن `datetime` للتاريخ بعد 1000 يوم من الآن. تجري شيفرة بايثون عملية حسابية للتاريخ لمعرفة أن 1000 يوم بعد 2 من الشهر 12 من عام 2018 ستكون في 18 من الشهر الثامن من عام 2021. يُعَد ذلك مفيدًا لأنه يجب أن تتذكّر عدد الأيام في كل شهر والعامل المشترك للسنوات الكبيسة وغيرها من التفاصيل الصعبة عندما تحسب 1000 يوم بعد تاريخ معين، لذا تعالج الوحدة `datetime` جميع تلك الأمور نيابةً عنك.

يمكن جمع كائنات `timedelta` أو طرحها من كائنات `datetime` أو كائنات `timedelta` الأخرى باستخدام المعاملَين `+` و `-`، ويمكن ضرب كائن `timedelta` أو قسمته على عدد صحيح أو قيم عشرية باستخدام المعاملَين `*` و `/`. لندخِل مثلًا ما يلي في الصدفة التفاعلية:

```
❶ >>> oct21st = datetime.datetime(2019, 10, 21, 16, 29, 0)
❷ >>> aboutThirtyYears = datetime.timedelta(days=365 * 30)
 >>> oct21st
 datetime.datetime(2019, 10, 21, 16, 29)
 >>> oct21st - aboutThirtyYears
 datetime.datetime(1989, 10, 28, 16, 29)
 >>> oct21st - (2 * aboutThirtyYears)
 datetime.datetime(1959, 11, 5, 16, 29)
```

أنشأنا كائن `datetime` ليوم 21 من الشهر العاشر من عام 2019 ❶، وأنشأنا كائن `timedelta` لمدة 30 عامًا تقريبًا بافتراض أن السنة تتكون من 365 يومًا ❷. يؤدي طرح `aboutThirtyYears` من `oct21st` إلى الحصول على كائن `datetime` للتاريخ قبل 30 عامًا من تاريخ 21 من الشهر العاشر من عام 2019، ويؤدي طرح `2 * aboutThirtyYears` من `oct21st` إلى إعادة كائن `datetime` للتاريخ قبل 60 عامًا من تاريخ 21 من الشهر العاشر من عام 2019.

17.4.2 الإيقاف المؤقت حتى تاريخ محدد

يتيح التابع `time.sleep()` إيقافَ برنامجٍ ما مؤقتًا لعددٍ محدّدٍ من الثواني، حيث يمكنك إيقاف برامجك مؤقتًا حتى تاريخ محدّد باستخدام حلقة `while`، فمثلًا ستستمر الشيفرة البرمجية التالية في التكرار حتى يوم نهاية عام 2024:

```
import datetime
import time
end2024 = datetime.datetime(2024, 12, 31, 0, 0, 0)
while datetime.datetime.now() < end2024:
  time.sleep(1)
```

سيؤدي استدعاء `time.sleep(1)` إلى إيقاف برنامج بايثون الخاص بك مؤقتًا حتى لا يضيع الحاسوب دورات معالجة لوحدة المعالجة المركزية بعد التحقق من الوقت مرارًا وتكرارًا، ولذا تتحقق حلقة `while` من الشرط مرة واحدة فقط في الثانية ومع ذلك تستمر إل غاية باقي البرنامج بعد اليوم المحدد من عام 2024 (أو أيّ تاريخ تبرمجه للتوقف).

17.4.3 تحويل كائنات datetime إلى سلاسل نصية

لا تُعَد علامات يونكس الزمنية وكائنات `datetime` سهلة القراءة، لذا نستخدم التابع `strftime()` لعرض كائن `datetime` بوصفها سلسلة نصية، حيث يشير الحرف f الموجود في اسم الدالة `strftime()` إلى التنسيق format. يستخدم التابع `strftime()` موجّهات Directives مشابهة لتنسيق سلاسل بايثون النصية، حيث يحتوي الجدول الآتي على قائمة كاملة بموجّهات التابع `strftime()`:

موجّه التابع strftime()	معناه
`%Y`	العام مع القرن مثل `'2024'`
`%y`	العام بدون القرن من `'00'` إلى `'99'` (من عام 1970 إلى 2069 مثلًا)
`%m`	الشهر كعدد عشري من `'01'` إلى `'12'`
`%B`	اسم الشهر كاملًا مثل `'November'`
`%b`	اسم الشهر المختصر مثل `'Nov'`
`%d`	يوم من الشهر من `'01'` إلى `'31'`
`%j`	يوم من السنة من `'001'` إلى `'366'`
`%w`	يوم من الأسبوع من `'0'` (الأحد) إلى `'6'` (السبت)
`%A`	الاسم الكامل ليوم من الأسبوع مثل `'Monday'`
`%a`	الاسم المختصر ليوم من الأسبوع مثل `'Mon'`
`%H`	الساعة (نظام 24 ساعة) من `'00'` إلى `'23'`
`%I`	الساعة (نظام 12 ساعة) من `'01'` إلى `'12'`
`%M`	الدقيقة من `'00'` إلى `'59'`
`%S`	الثانية من `'00'` إلى `'59'`
`%p`	صباحًا `'AM'` أو مساءً `'PM'`
`%%`	المحرف `'%'` حرفيًا

الجدول 22: موجّهات التابع strftime()

مرّر سلسلة نصية ذات تنسيقٍ مخصَّص تحتوي على موجّهات التنسيق (مع أيّ خطوط مائلة ونقطتين وغير ذلك) إلى التابع `strftime()`، وسيعيد هذا التابع معلومات كائن `datetime` بوصفها سلسلة نصية مُنسَّقة.

لندخِل الآن ما يلي في الصدفة التفاعلية:

```
>>> oct21st = datetime.datetime(2023, 10, 21, 16, 29, 0)
>>> oct21st.strftime('%Y/%m/%d %H:%M:%S')
'2023/10/21 16:29:00'
```

```
>>> oct21st.strftime('%I:%M %p')
'04:29 PM'
>>> oct21st.strftime("%B of '%y")
"October of '23"
```

خزّنّا في المثال السابق كائن `datetime` ليوم 21 من الشهر العاشر من عام 2023 في الساعة 4:29 مساءً في المتغير `oct21st`. سوف يعيد تمرير سلسلة التنسيق المخصَّصة `'%Y/%m/%d %H:%M:%S'` إلى التابع `strftime()` سلسلةً نصية تحتوي على القيم 2023 و10 و21 والمفصولة بخطوط مائلة والقيم 16 و29 و00 المفصولة بنقطتين، ويؤدي تمرير السلسلة النصية `'%I:%M% p'` إلى إعادة `'04:29 PM'`، ويؤدي تمرير السلسلة النصية `"%B of '%y"` إلى إعادة `"October of '23"`.

لاحظ أننا لا نضع `datetime.datetime` قبل التابع `strftime()`.

17.4.4 تحويل السلاسل النصية إلى كائنات datetime

إذا كان لديك سلسلة نصية تمثّل معلومات التاريخ مثل `'2023/10/21 16:29:00'` أو `'October 21, 2023'` وتريد تحويلها إلى كائن `datetime`، فاستخدم الدالة `datetime.datetime.strptime()`، حيث تُعَد هذه الدالة عكس التابع `strftime()`. يجب تمرير سلسلة تنسيق مُخصَّصة تستخدم الموجّهات نفسها التي يستخدمها التابع `strftime()` إلى الدالة `strptime()` حتى تعرف كيفية تحليل السلسلة النصية وفهمها، ويشير الحرف p الموجود في اسم الدالة `strptime()` إلى التحليل Parse. لندخِل الآن ما يلي في الصدفة التفاعلية:

```
❶ >>> datetime.datetime.strptime('October 21, 2023', '%B %d, %Y')
  datetime.datetime(2023, 10, 21, 0, 0)
  >>> datetime.datetime.strptime('2023/10/21 16:29:00', '%Y/%m/%d %H:
%M:%S')
  datetime.datetime(2023, 10, 21, 16, 29)
  >>> datetime.datetime.strptime("October of '23", "%B of '%y")
  datetime.datetime(2023, 10, 1, 0, 0)
  >>> datetime.datetime.strptime("November of '63", "%B of '%y")
  datetime.datetime(2063, 11, 1, 0, 0)
```

يمكن الحصول على كائن `datetime` من السلسلة النصية `'October 21, 2023'` من خلال تمرير هذه السلسلة كوسيطٍ أول إلى الدالة `strptime()` وتمرير سلسلة التنسيق المُخصصة المقابلة للسلسلة النصية

`'October 21, 2023'` كوسيطٍ ثـانٍ ❶. يجب أن تتطـابق السلسـلة النصـية الـتي تحتـوي على معلومـات التاريخ مع سلسلة التنسيق المخصصة تمامًا، وإلّا سيرفع بايثون استثناء `ValueError`.

17.5 مراجعة لدوال بايثون الخاصة بالوقت

يمكن أن تتضمن التواريخ والأوقات في بايثون عددًا من أنواع البيانات والدوال المختلفة، لذا سنوضح فيمـا يلي الأنواع الثلاثة المختلفة من القيم المُستخدَمة لتمثيل الوقت:

- علامة يونكس الزمنية التي تستخدمها الوحدة `time`، وهي قيمة عشرية أو صـحيحة لعـدد الثـواني منـذ الساعة 12 صباحًا في 1 من الشهر الأول من عام 1970 بالتوقيت العالمي المنسق.
- كائن `datetime` الخاص بالوحدة `datetime`، والذي يحتوي على أعداد صـحيحة مخزَّنـة في السـمات `year` و `month` و `day` و `hour` و `minute` و `second`.
- كائن `timedelta` الخاص بالوحدة `datetime`، والذي يمثّل مدة زمنية وليس لحظة مُحدَّدة.
- إليك مراجعة لدوال الوقت ومعاملاتها والقيم التي تعيدها:
- `time.time()`: تعيد هذه الدالة القيمة العشرية لعلامة يونكس الزمنية للحظة الحالية.
- `time.sleep(seconds)`: توقِـف هـذه الدالـة البرنـامج لعـددٍ من الثـواني الـتي يحـدّدها الوسـيط `seconds`.
- `datetime.datetime(year, month, day, hour, minute, second)`: تعيد هذه الدالة كـائن `datetime` للحظـة الـتي تحـدّدها وسـطاؤها. إذا لم تتـوفّر قيم للوسـطاء `hour` أو `minute` أو `second`، فستكون قيمها الافتراضية 0.
- `datetime.datetime.now()`: تعيد هذه الدالة كائن `datetime` للحظة الحالية.
- `datetime.datetime.fromtimestamp(epoch)`: تعيـد هـذه الدالـة كـائن `datetime` للحظـة التي يمثلها وسيط العلامة الزمنية `epoch`.
- `datetime.timedelta(weeks, days, hours, minutes, seconds, milliseconds, microseconds)`: تعيد هذه الدالة كائن `timedelta` الذي يمثـل مـدة زمنيـة، وتُعَد وسطاء الكلمات المفتاحية لهذه الدالة اختيارية ولا تتضمن `month` أو `year`.
- `total_seconds()`: يعيد هذا التابع الخاص بكائنات `timedelta` عـدد الثـواني الـتي يمثّلهـا كـائن `timedelta`.

- `strftime(format)`: يعيد هذا التابع سلسلة نصية للوقت الذي يمثّله كائن `datetime` بتنسيقٍ مخصَّص يعتمد على سلسلة التنسيق `format`. اطّلع على الجدول السابق من أجل الحصول على تفاصيل التنسيق.
- `datetime.datetime.strptime(time_string, format)`: تعيد هذه الدالة كائن `datetime` للحظة التي يحدّدها الوسيط `time_string`، وتُحلَّل باستخدام وسيط سلسلة التنسيق `format`. اطّلع على الجدول السابق للحصول على تفاصيل التنسيق.

17.6 تعدّد الخيوط Multithreading

لنفترض أنك تريد جدولة شيفرتك البرمجية لتشغيلها بعد تأخير محدّد أو في وقت معيَّن، حيث يمكنك إضافة شيفرة برمجية كما يلي في بداية برنامجك:

```
import time, datetime
startTime = datetime.datetime(2029, 01, 01, 0, 0, 0)
while datetime.datetime.now() < startTime:
  time.sleep(1)
print('Program now starting on new year 2029')
--snip--
```

تحدّد الشيفرة البرمجية السابقة وقت البدء في 01 من الشهر الأول من عام 2029، وتستمر في استدعاء الدالة `time.sleep(1)` حتى الوصول إلى وقت البدء، ولا يستطيع برنامجك فعل أيّ شيء أثناء انتظار انتهاء حلقة استدعاءات `time.sleep()`، ويبقى متوقفًا حتى يوم رأس السنة من عام 2029، لأن برامج بايثون افتراضيًا لها خيط Thread تنفيذ واحد.

يمكنك فهم ما هو خيط التنفيذ من خلال تذكّر ما ناقشناه في الفصل الثاني حول التحكم في التدفق عندما تخيّلت تنفيذ برنامجٍ ما على أنه وضع إصبعك على سطرٍ من الشيفرة البرمجية في برنامجك والانتقال إلى السطر التالي أو المكان التي ترسله تعليمة التحكم في التدفق، حيث يحتوي البرنامج ذو الخيط الواحد Single-threaded على إصبع واحد فقط، ولكن يحتوي البرنامج متعدد الخيوط Multithreaded على أصابع متعددة. يستمر كل إصبع في التحرك إلى السطر التالي من الشيفرة البرمجية كما تحدِّده تعليمات التحكم في التدفق، ولكن يمكن أن تكون الأصابع في أماكن مختلفة في البرنامج لتنفيذ أسطر مختلفة من الشيفرة البرمجية في الوقت ذاته. لاحظ أن جميع البرامج التي مرّت معنا حتى الآن ذات خيط واحد.

يمكنك تنفيذ الشيفرة البرمجية المؤجَّلة أو المجدولة في خيط منفصل باستخدام وحدة بايثون `threading` بدلًا من أن تنتظر شيفرتك البرمجية بأكملها انتهاء الدالة `time.sleep()`، حيث سيتوقف الخيط المنفصل مؤقتًا عند استدعاءات `time.sleep`، بينما يمكن لبرنامجك تنفيذ أشياء أخرى في الخيط الأصلي.

ننشئ خيطًا منفصلًا من خلال إنشاء كائن Thread أولًا، وذلك عبر استخدام استدعاء الدالة threading.Thread(). إذًا لندخِل الشيفرة البرمجية التالية في ملفٍ جديد ونحفظه بالاسم threadDemo.py:

```
import threading, time
print('Start of program.')
❶ def takeANap():
    time.sleep(5)
    print('Wake up!')
❷ threadObj = threading.Thread(target=takeANap)
❸ threadObj.start()
print('End of program.')
```

عرّفنا في الشيفرة البرمجية السابقة الدالة التي نريد استخدامها في خيطٍ جديد ❶، واستدعينا الدالة threading.Thread() ومرّرنا لها وسيط الكلمات المفتاحية target=takeANap ❷ لإنشاء كائن Thread، وهذا يعني أن الدالة التي نريد استدعاءها في الخيط الجديد هي takeANap(). لاحظ أن وسيط الكلمات المفتاحية هو target=takeANap وليس target=takeANap()، لأنك تريد تمرير الدالة takeANap() كوسيط، ولا تريد استدعاءها وتمرير قيمتها المُعادة.

خزّنا الكائن Thread الذي أنشأته الدالة threading.Thread() في المتغير threadObj، ثم استدعينا الدالة threadObj.start() ❸ لإنشاء الخيط الجديد والبدء في تنفيذ الدالة المستهدفة في الخيط الجديد. سيكون الناتج كما يلي عند تشغيل هذا البرنامج:

```
Start of program.
End of program.
Wake up!
```

قد يكون الخرج السابق مربكًا بعض الشيء، حيث إذا كانت التعليمة print('End of program.') هي السطر الأخير من البرنامج، فقد تعتقد أنه آخر شيء سيُطبَع، ولكن تُشغَّل الدالة المستهدفة للمتغير threadObj في خيط تنفيذ جديد عند استدعاء الدالة threadObj.start()، لذا تظهر العبارة Wake up! في النهاية. فكّر في الأمر كإصبعٍ ثانٍ يظهر في بداية الدالة takeANap()، حيث يستمر الخيط الرئيسي في تنفيذ التعليمة print('End of program.')، بينما يتوقف الخيط الجديد الذي كان ينفّذ استدعاء

`time.sleep(5)` مؤقتًا لمدة 5 ثوانٍ، ويطبع عبارة `'Wake up!'` بعد أن يستيقظ من غفوته لمدة 5 ثوان، ثم يعود من الدالة `takeANap()`، وبالتالي فإن عبارة `'Wake up!'` هي آخر ما يطبعه البرنامج زمنيًا.

ينتهي البرنامج عادةً عند تشغيل السطر الأخير من الشيفرة البرمجية في الملف أو عند استدعاء الدالة `sys.exit()`، ولكن يحتوي البرنامج `threadDemo.py` على خيطين هما: الأول هو الخيط الأصلي الذي بدأ في بداية البرنامج وينتهي بعد التعليمة `print('End of program.')`، والخيط الثاني الذي ينشأ عند استدعاء الدالة `threadObj.start()` ويبدأ عند بداية الدالة `takeANap()` وينتهي بعد العودة من الدالة `takeANap()`.

لن ينتهي برنامج بايثون حتى تنتهي جميع خيوطه. لاحظ أن الخيط الثاني لا يزال ينفّذ الاستدعاء `time.sleep(5)` عند تشغيل البرنامج `threadDemo.py` بالرغم من انتهاء الخيط الأصلي.

17.6.1 تمرير الوسطاء إلى الدالة المستهدفة للخيط

إذا أخذت الدالة المستهدفة التي تريد تشغيلها في الخيط الجديد وسطاءً، فيمكنك تمرير وسطائها إلى الدالة `threading.Thread()`. لنفترض مثلًا أنك تريد تشغيل استدعاء الدالة `print()` التالية في خيطها الخاص:

```
>>> print('Cats', 'Dogs', 'Frogs', sep=' & ')
Cats & Dogs & Frogs
```

يحتوي استدعاء الدالة `print()` السابق على ثلاث وسطاء عادية هي: `'Cats'` و `'Dogs'` و `'Frogs'` ووسيط كلمات مفتاحية واحد هو `sep=' & '`، حيث يمكن تمرير الوسطاء العادية كقائمة إلى وسيط الكلمات المفتاحية `args` في الدالة `threading.Thread()`، ويمكن تحديد وسيط الكلمات المفتاحية Keyword Argument كقاموس لوسيط الكلمات المفتاحية `kwargs` في الدالة `threading.Thread()`.

لندخِل الآن ما يلي في الصدفة التفاعلية:

```
>>> import threading
>>> threadObj = threading.Thread(target=print, args=['Cats', 'Dogs',
'Frogs'],
kwargs={'sep': ' & '})
>>> threadObj.start()
Cats & Dogs & Frogs
```

نتأكد من تمرير الوسطاء `'Cats'` و `'Dogs'` و `'Frogs'` إلى الدالة `print()` في الخيط الجديد وذلك من خلال تمرير الوسطاء `args=['Cats', 'Dogs', 'Frogs']` إلى الدالة `threading.Thread()`، ونتأكد أيضًا من تمرير وسيط الكلمات المفتاحية `sep=' & '` إلى الدالة `print()` في الخيط الجديد وذلك من خلال تمرير `kwargs={'sep': ' & '}` إلى الدالة `threading.Thread()`. يؤدي استدعاء الدالة

`threadObj.start()` إلى إنشاء خيط جديد لاستدعاء الدالة `print()`، وستمرّر إليها `'Cats'` و`'Dogs'` و`'Frogs'` كوسطاء والقيمة `' & '` لوسيط الكلمات المفتاحية `sep`.

تُعَد الطريقة التالية غير صحيحة لإنشاء الخيط الجديد الذي يستدعي الدالة `print()`:

```
threadObj = threading.Thread(target=print('Cats', 'Dogs', 'Frogs',
sep=' & '))
```

تؤدي الطريقة السابقة إلى استدعاء الدالة `print()` وتمرير القيمة المُعادة الخاصة بها كوسيط الكلمات المفتاحية `target`، حيث تكون القيمة المُعادة الخاصة بالدالة `print()` دائمًا `None`، ولا تؤدي إلى تمرير الدالة `print()` نفسها، لذا استخدم وسطاء الكلمات المفتاحية `args` و `kwargs` الخاصة بالدالة `threading.Thread()` عند تمرير الوسطاء إلى دالة في خيطٍ جديد.

17.6.2 مشاكل التزامن Concurrency

يمكنك بسهولة إنشاء عدة خيوط جديدة وتشغيلها جميعًا في الوقت نفسه، ولكن يمكن أن تسبّب الخيوط المتعددة أيضًا مشكلات في التزامن التي تحدث عندما تقرأ الخيوطُ المتغيرات وتكتبها في الوقت نفسه، مما يؤدّي إلى تصادم الخيوط مع بعضها البعض. قد يكون من الصعب إعادة إنتاج مشكلات التزامن بصورة متناسقة، مما يصعّب تنقيح أخطائها Debug.

تُعَد البرمجة متعددة الخيوط موضوعًا واسعًا ولن نناقشه في هذا الفصل، ولكن ما عليك أن تضعه في بالك هو أنه يجب ألّا تسمح أبدًا لخيوط متعددة بقراءة أو كتابة المتغيرات نفسها لتجنب مشكلات التزامن. تأكّد من أن الدالة المستهدفة لكائن `Thread` الجديد عند إنشائه تستخدم المتغيرات المحلية فقط في تلك الدالة، مما سيؤدّي إلى تجنب مشكلات التزامن التي يصعب تنقيح أخطائها في برامجك.

17.7 تطبيق عملي: برنامج متعدد الخيوط لتنزيل قصص XKCD الهزلية

كتبتَ في الفصل الثاني عشر برنامجًا ينزّل جميع قصص XKCD الهزلية من موقع XKCD، حيث كان برنامجًا له خيط واحد، لأنه ينزّل قصة هزلية واحدة في كل مرة. يقضي هذا البرنامج الكثير من وقت التشغيل في إنشاء اتصال بالشبكة لبدء التنزيل وكتابة الصور المُنزَّلة على القرص الصلب، وإذا كان لديك اتصال إنترنت له حيز نطاق تراسلي عريض، فلن يستخدم برنامجك ذو الخيط الواحد هذا الحيز المتوفر بأكمله.

يحتوي البرنامج متعدد الخيوط على بعض الخيوط التي تنزّل القصص الهزلية، وتنشئ بعض الخيوط الأخرى الاتصالات وتكتب ملفات الصور الهزلية على القرص الصلب، حيث يستخدم هذا البرنامج اتصال الإنترنت الخاص بك بكفاءة أكبر وينزّل مجموعة القصص الهزلية بسرعة أكبر. افتح تبويبًا جديدًا في محرّرك لإنشاء ملف جديد واحفظه بالاسم `threadedDownloadXkcd.py`، وستعدّل هذا البرنامج لإضافة خيوط متعددة، فالشيفرة البرمجية المُعدَّلة بالكامل متاحة للتنزيل على nostarch.

17.7.1 الخطوة الأولى: تعديل البرنامج لاستخدام دالة

سيكون هذا البرنامج في أغلبه مشابهًا لشيفرة التنزيل البرمجية التي كتبناها في الفصل الثاني عشر، لذا سنتخطّى شرح `requests` وشيفرة المكتبة Beautiful Soup. التغييرات الرئيسية التي يجب أن تجريها هي استيراد الوحدة `threading` وإنشاء الدالة `downloadXkcd()` التي تأخذ أرقام البداية والنهاية للقصة الهزلية كمعاملاتٍ لها.

سيؤدّي استدعاء الدالة `downloadXkcd(140, 280)` مثلًا إلى تكرار شيفرة التنزيل لتنزيل القصص الهزلية 140 و 141 و 142 وحتى القصة الهزلية 279. سيستدعي كلُّ خيط تنشئه الدالةَ `downloadXkcd()` ويمرّر مجالًا مختلفًا من القصص الهزلية لتنزيلها.

أضِف الشيفرة البرمجية التالية إلى برنامج `threadedDownloadXkcd.py` الخاص بك:

```
#! python3

# threadedDownloadXkcd.py - تنزيل قصص XKCD الهزلية باستخدام خيوط متعددة

import requests, os, bs4, threading

❶ os.makedirs('xkcd', exist_ok=True)     # تخزين القصص الهزلية في المجلد ./xkcd

❷ def downloadXkcd(startComic, endComic):

  ❸ for urlNumber in range(startComic, endComic):

      # تنزيل الصفحة

      print('Downloading page https://xkcd.com/%s...' % (urlNumber))

    ❹ res = requests.get('https://xkcd.com/%s' % (urlNumber))

      res.raise_for_status()

    ❺ soup = bs4.BeautifulSoup(res.text, 'html.parser')

      # البحث عن عنوان URL للصورة الهزلية

    ❻ comicElem = soup.select('#comic img')

      if comicElem == []:

        print('Could not find comic image.')

      else:
```

```
        ❼ comicUrl = comicElem[0].get('src')
          # تنزيل الصورة
          print('Downloading image %s...' % (comicUrl))
        ❽ res = requests.get('https:' + comicUrl)
          res.raise_for_status()
          # ./xkcd حفظ الصورة في المجلد
          imageFile = open(os.path.join('xkcd',
os.path.basename(comicUrl)),
'wb')
          for chunk in res.iter_content(100000):
            imageFile.write(chunk)
          imageFile.close()
# Thread إنشاء وبدء كائنات الخيط
# انتظار انتهاء جميع الخيوط
```

نستورد الوحدات التي نحتاجها، ثم ننشئ مجلدًا لتخزين القصص الهزلية ❶، ونبدأ بتعريف الدالة `downloadxkcd()` ❷، ثم نمر ضمن حلقة على جميع الأرقام الموجودة في المجال المحدَّد ❸ وننزّل كل صفحة❹. نستخدم المكتبة `Beautiful Soup` للبحث في شيفرة HTML لكل صفحة ❺ والعثور على الصورة الهزلية ❻، حيث إذا لم نعثر على صورة هزلية في الصفحة، فإننا نطبع رسالة، وإلّا سنحصل على عنوان URL للصورة ❼ وننزّلها ❽. أخيرًا، نحفظ الصورة في المجلد الذي أنشأناه.

17.7.2 الخطوة الثانية: إنشاء وبدء الخيوط

عرّفنا الدالة `downloadxkcd()` وسننشئ الآن الخيوط المتعددة التي يستدعي كل منها الدالة `downloadxkcd()` لتنزيل مجالات مختلفة من القصص الهزلية من موقع XKCD. أضِف الشيفرة البرمجية التالية إلى البرنامج `threadedDownloadXkcd.py` بعد تعريف الدالة `downloadxkcd()`:

```
#! python3
# threadedDownloadXkcd.py - تنزيل قصص XKCD الهزلية باستخدام خيوط متعددة
--snip--
```

```
# إنشاء وبدء كائنات الخيط Thread

downloadThreads = []                # قائمة بجميع كائنات الخيط Thread

for i in range(0, 140, 10):         # التكرار 14 مرة وإنشاء 14 خيطًا

  start = i

  end = i + 9

  if start == 0:

    start = 1 # لا توجد قصة هزلية رقمها 0، لذا اضبط المتغير على القيمة 1

  downloadThread = threading.Thread(target=downloadXkcd, args=(start,
end))

  downloadThreads.append(downloadThread)

  downloadThread.start()
```

ننشئ أولًا قائمة فارغة `downloadThreads`، والتي ستساعدنا على تعقّب العديد من كائنات `Thread` التي سننشئها، ثم نبدأ حلقة `for`، حيث ننشئ في كل تكرار من هذه الحلقة كائن `Thread` باستخدام الدالة `threading.Thread()`، ونلحِق هذا الكائن بالقائمة، ونستدعي التابع `start()` لبدء تشغيل الدالة `downloadXkcd()` في الخيط الجديد. تضبط حلقة `for` المتغير `i` على القيم من 0 إلى 140 بخطوة مقدارها 10، لذا يجب ضبط المتغير `i` على القيمة `0` في التكرار الأول، وعلى القيمة `10` في التكرار الثاني، وعلى القيمة `20` في التكرار الثالث، وإلخ.

نمرّر الوسيط `args=(start, end)` إلى الدالة `threading.Thread()`، لذا سيكون الوسيطان المُمرَّران إلى الدالة `downloadXkcd()` هما `1` و `9` في التكرار الأول، و `10` و `19` في التكرار الثاني، و `20` و `29` في التكرار الثالث ...إلخ.

سينتقل الخيط الرئيسي إلى التكرار التالي من حلقة `for` وينشئ الخيط التالي عند استدعاء التابع `start()` الخاص بالكائن `Thread` ويبدأ الخيط الجديد بتشغيل الشيفرة البرمجية الموجودة ضمن الدالة `downloadXkcd()`.

17.7.3 الخطوة الثالثة: انتظار انتهاء جميع الخيوط

يتحرك الخيط الرئيسي كالمعتاد بينما تنزّل الخيوطُ الأخرى التي أنشأناها القصصَ الهزلية، ولكن لنفترض أن هناك بعض التعليمات البرمجية التي لا تريد تشغيلها في الخيط الرئيسي حتى يكتمل تنفيذ جميع الخيوط الأخرى، حيث سيتوقّف استدعاء التابع `join()` الخاص بالكائن `Thread` حتى انتهاء هذا الخيط.

يمكن للخيط الرئيسي استدعاء التابع `join()` على كلٍّ من الخيوط الأخرى باستخدام حلقة `for` للتكرار على كافة كائنات `Thread` الموجودة في القائمة `downloadThreads`.

أضِف الآن ما يلي إلى نهاية برنامجك:

```
#! python3
# threadedDownloadXkcd.py - تنزيل قصص XKCD الهزلية باستخدام خيوط متعددة
--snip--
# الانتظار حتى انتهاء جميع الخيوط
for downloadThread in downloadThreads:
  downloadThread.join()
print('Done.')
```

لن تُطبَع السلسلة النصية `'Done.'` حتى إعادة جميع استدعاءات التابع `join()`، حيث إذا اكتمل كائن `Thread` عند استدعاء التابع `join()` الخاص به، فسيعود التابع مباشرةً ببساطة.

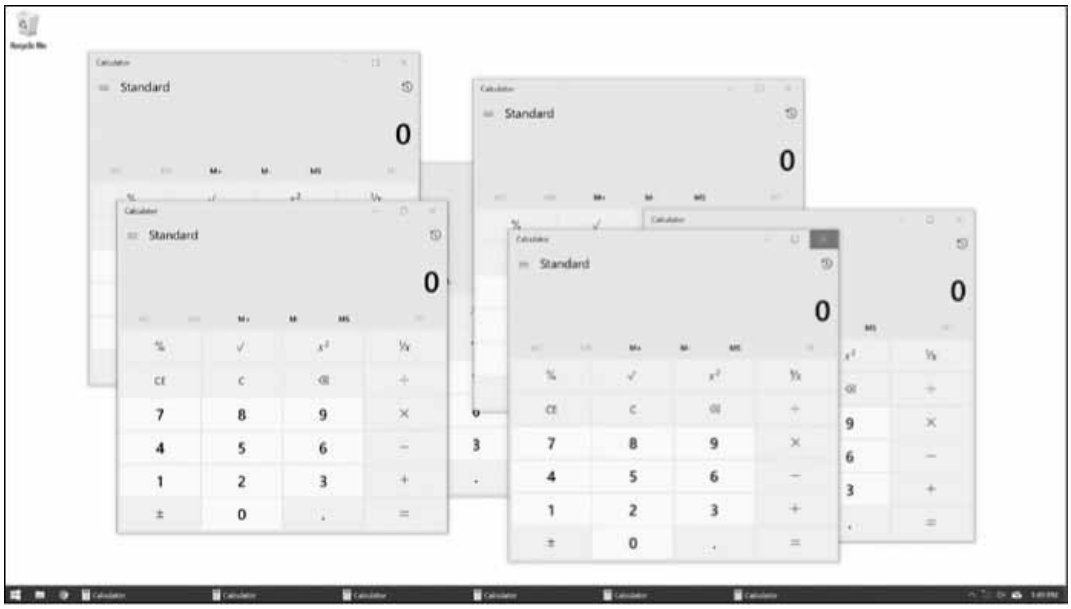
إذا أردتَ توسيع هذا البرنامج باستخدام شيفرة برمجية تُشغَّل فقط بعد تنزيل كافة القصص الهزلية، فيمكنك وضع شيفرتك البرمجية الجديدة مكان السطر `print('Done.')`.

17.8 تشغيل Launching برامج أخرى من برنامج بايثون

يمكن لبرنامج بايثون الخاص بك بدء برامج أخرى على حاسوبك باستخدام الدالة `Popen()` الموجودة في الوحدة المُدمَجة `subprocess`، حيث يرمز الحرف P في اسم هذه الدالة إلى العملية Process.

إذا كان لديك نسخ Instances متعددة من تطبيق مفتوح، فستُعَد كلّ نسخة من هذه النسخ عمليةً منفصلة للبرنامج نفسه، فمثلًا إذا فتحتَ نوافذ متعددة لمتصفح الويب في الوقت نفسه، فإن كلّ نافذة من تلك النوافذ هي عملية مختلفة لبرنامج متصفح الويب.

يوضّح الشكل التالي مثالًا لعمليات آلة حاسبة متعددة مفتوحة في وقتٍ واحد:



الشكل 89: ست عمليات مُشغَّلة لبرنامج الآلة الحاسبة نفسه

يمكن أن يكون لكل عملية خيوط متعددة، ولا يمكن للعملية قراءة وكتابة متغيرات عملية أخرى مباشرةً على عكس الخيوط. إذا افترضنا أن برنامجًا متعدد الخيوط يحتوي على أصابع متعددة تتبع الشيفرة البرمجية، فإن فتح عمليات متعددة للبرنامج نفسه يشبه وجود صديق لديه نسخة منفصلة من شيفرة البرنامج البرمجية، وكلاكما تنفّذان البرنامج نفسه بصورة مستقلة.

إذا أردتَ بدء برنامج خارجي من سكربت بايثون الخاص بك، فمرّر اسم ملف البرنامج إلى الدالة `subprocess.Popen()`. انقر بزر الفأرة الأيمن على عنصر القائمة "ابدأ Start" الخاص بالتطبيق وحدّد الخيار "خصائص Properties" لعرض اسم ملف التطبيق في نظام ويندوز، وانقر مع الضغط على مفتاح `CTRL` على التطبيق وحدّد الخيار "إظهار محتويات الحزمة Show Package Contents" للعثور على مسار الملف القابل للتنفيذ في نظام ماك macOS.

ستعيد بعد ذلك الدالة `Popen()` مباشرةً، وضع في بالك أن البرنامج الذي شغّلناه لا يعمل في الخيط نفسه لبرنامج بايثون الخاص بك.

أدخِل ما يلي في الصدفة التفاعلية على حاسوبٍ يعمل بنظام ويندوز:

```
>>> import subprocess
>>> subprocess.Popen('C:\\Windows\\System32\\calc.exe')
<subprocess.Popen object at 0x0000000003055A58>
```

أدخِل ما يلي على نظام يعمل بنظام أوبنتو لينكس Ubuntu Linux:

```
>>> import subprocess
>>> subprocess.Popen('/snap/bin/gnome-calculator')
<subprocess.Popen object at 0x7f2bcf93b20>
```

تختلف العملية قليلًا على نظام ماك macOS، لذا اطّلع على قسم "فتح الملفات باستخدام التطبيقات الافتراضية" من هذا الفصل لمزيد من التفاصيل.

تكون القيمة المُعادة هي كائن `Popen` الذي له تابعان مفيدان هما: `poll()` و `wait()`، حيث يمكنك التفكير في التابع `poll()` بأنك تسأل سائقك "هل وصلنا؟" مرارًا وتكرارًا حتى الوصول إلى وِجهتك، ويعيد هذا التابع القيمة `None` إذا كانت العملية لا تزال قيد التشغيل في وقت استدعاء هذا التابع.

إذا أُنهي البرنامج، فسيعيد العدد الصحيح لرمز الخروج exit code الخاص بالعملية، حيث يُستخدَم رمز الخروج للإشارة إلى أن العملية انتهت بدون أخطاء (رمز الخروج 0) أو ما إذا قد تسبّب خطأٌ ما في إنهاء العملية (رمز خروج غير صفري قيمته 1 عادةً، ولكنه قد يختلف اعتمادًا على البرنامج).

يشبه التابع `wait()` الانتظار حتى يصل السائق إلى وِجهتك، حيث يوقِف هذا التابع التنفيذ حتى تنتهي العملية التي جرى تشغيلها، ويُعَد ذلك مفيدًا إذا أردتَ أن يتوقف برنامجك مؤقتًا حتى ينتهي المستخدم من البرنامج الآخر. القيمة المُعادة من التابع `wait()` هي العدد الصحيح لرمز الخروج الخاص بالعملية.

أدخِل ما يلي في الصدفة التفاعلية على نظام ويندوز، ولاحظ أن استدعاء التابع `wait()` سيوقِف التنفيذ حتى إنهاء برنامج الرسام MS Paint الذي جرى تشغيله:

```
 >>> import subprocess

❶ >>> paintProc = subprocess.Popen('c:\\Windows\\System32\
\mspaint.exe')
❷ >>> paintProc.poll() == None
 True

❸ >>> paintProc.wait() # لن يعود حتى إغلاق برنامج الرسام MS Paint
   0
 >>> paintProc.poll()
   0
```

فتحنا في المثال السابق عملية برنامج الرسام MS Paint ❶، وتحقّقنا مما إذا كان التابع `poll()` يعيد القيمة `None` ❷ بينما لا تزال العملية قيد التشغيل، حيث ينبغي أن يكون ذلك صحيحًا لأنها لا تزال قيد التشغيل. أغلقنا بعد ذلك برنامج الرسام MS Paint واستدعينا التابع `wait()` للعملية المنتهية ❸.

سيعيد الآن التابعان `wait()` و `poll()` القيمة 0، مما يشير إلى أن العملية قد انتهت بدون أخطاء.

> **ملاحظة**: إذا شغّلت برنامج الآلة الحاسبة calc.exe على نظام ويندوز 10 باستخدام الدالة `subprocess.Popen()`، فستلاحظ أن التابع `wait()` يعيد مباشرةً على عكس برنامج الرسام mspaint.exe بالرغم من أن تطبيق الآلة الحاسبة لا يزال قيد التشغيل، والسبب أن برنامج الآلة الحاسبة calc.exe يشغّل تطبيق الآلة الحاسبة ثم يغلق نفسه مباشرةً. يُعَد برنامج الآلة الحاسبة الخاص بنظام ويندوز "تطبيق متجر مايكروسوفت موثوق به"، ولن ندخل في تفاصيله في هذا الفصل، ولكن يكفي أن نقول أنه يمكن تشغيل البرامج بعدة طرقٍ خاصة بالتطبيق وبنظام التشغيل.

17.8.1 تمرير وسطاء سطر الأوامر إلى الدالة ()Popen

يمكنك تمرير وسطاء سطر الأوامر إلى العمليات التي تنشئها باستخدام الدالة `Popen()` من خلال تمرير قائمة تمثّل الوسيط الوحيد لهذه الدالة. ستكون السلسلة النصية الأولى في هذه القائمة هي اسم الملف التنفيذي للبرنامج الذي تريد تشغيله، وتكون جميع السلاسل النصية اللاحقة وسطاء سطر الأوامر التي تمرّرها إلى البرنامج عندما يبدأ. تمثّل هذه القائمة قيمة `sys.argv` للبرنامج الذي جرى تشغيله.

لا تستخدم معظم التطبيقات ذات واجهة المستخدم الرسومية Graphical User Interface -أو GUI اختصارًا- وسطاء سطر الأوامر على نطاق واسع كما تفعل البرامج المستندة إلى سطر الأوامر أو البرامج المستندة إلى الطرفية Terminal، ولكن تقبل معظم تطبيقات واجهة المستخدم الرسومية وسيطًا واحدًا للملف الذي ستفتحه التطبيقات مباشرةً عندما تبدأ. إذا استخدمتَ نظام ويندوز، فأنشئ مثلًا ملفًا نصيًا بسيطًا بالاسم `C:\Users\Al\hello.txt`، ثم أدخِل ما يلي في الصدفة التفاعلية:

```
>>> subprocess.Popen(['C:\\Windows\\notepad.exe', 'C:\\Users\Al\
\hello.txt'])
<subprocess.Popen object at 0x00000000032DCEB8>
```

لن يؤدي ذلك إلى تشغيل تطبيق المفكرة Notepad فحسب، بل سيؤدي أيضًا إلى فتح الملف `C:\Users\Al\hello.txt` مباشرةً.

17.8.2 أدوات مجدول المهام Task Scheduler و launchd و cron

إذا كنت خبيرًا في استخدام الحاسوب، فقد تكون على دراية بأداة مجدول المهام Task Scheduler على ويندوز أو أداة launchd على نظام ماك macOS أو أداة الجدولة cron على نظام لينكس، حيث تتيح لك هذه الأدوات المُوثَّقة جيدًا والموثوقة جدولةَ تشغيل التطبيقات في أوقات محددة.

يوفّر عليك استخدام المجدول المُدمَج مع نظام تشغيلك كتابةَ الشيفرة البرمجية الخاصة بالتحقق من ساعتك لجدولة برامجك، ولكن يمكنك استخدام الدالة `time.sleep()` إذا أردتَ فقط إيقاف البرنامج مؤقتًا

لفترة وجيزة، أو يمكنك تكرار شيفرتك البرمجية حتى تاريخ ووقت محدَّدين واستدعاء `time.sleep(1)` في كل مرة خلال الحلقة بدلًا من استخدام المجدول الخاص بنظام التشغيل.

17.8.3 فتح المواقع باستخدام شيفرة بايثون

يمكن للدالة `webbrowser.open()` تشغيل متصفح ويب من برنامجك إلى موقع ويب محدّد بدلًا من فتح تطبيق المتصفح باستخدام الدالة `subprocess.Popen()`.

17.8.4 تشغيل سكربتات بايثون الأخرى

يمكنك تشغيل سكربت بايثون من شيفرة بايثون أخرى مثل أيّ تطبيق آخر، فما عليك فعله سوى تمرير ملف بايثون `python.exe` القابل للتنفيذ إلى الدالة `Popen()` واسم ملف سكربت `.py` الذي تريد تشغيله بوصفه وسيطًا لهذه الدالة، فمثلًا سيشغّل ما يلي السكربت `hello.py` الذي كتبناه في الفصل الأول:

```
>>> subprocess.Popen(['C:\\Users\\<YOUR USERNAME>\\AppData\\Local\
\Programs\\
Python\\Python38\\python.exe', 'hello.py'])
<subprocess.Popen object at 0x000000000331CF28>
```

نمرّر إلى الدالة `Popen()` قائمةً تحتوي على سلسلة نصية تمثل مسار ملف بايثون القابل للتنفيذ وسلسلة نصية تمثّل اسم ملف السكربت، وإذا احتاج السكربت الذي تشغّله إلى وسطاء سطر الأوامر، فيمكنك إضافتها إلى القائمة بعد اسم ملف السكربت. موقع ملف بايثون القابل للتنفيذ على نظام ويندوز هو :

```
C:\Users\<YOUR
USERNAME>\AppData\Local\Programs\Python\Python38\python.exe
```

وعلى نظام ماك macOS هو:

```
/Library/Frameworks/Python.framework/Versions/3.8/bin/python3
```

أما على نظام لينكس فهو:

```
/usr/bin/python3.8
```

إذا شغّل برنامج بايثون الخاص بك برنامجَ بايثون آخر، فسيُشغَّل كلا البرنامجين ضمن عمليات منفصلة ولن يتمكن كلّ منهما من مشاركة متغيرات الآخر على عكس استيراد برنامج بايثون بوصفه وحدة Module.

17.8.5 فتح الملفات باستخدام التطبيقات الافتراضية

سيؤدّي النقر المزدوج على ملف `.txt` على حاسوبك إلى تشغيل التطبيق المرتبط بلاحقة الملف `.txt` تلقائيًا، وسيكون لحاسوبك ارتباطات متعددة بامتدادات الملفات المُعَدّة مسبقًا، ويمكن لبايثون أيضًا فتح الملفات بهذه الطريقة باستخدام الدالة `Popen()`.

يحتوي كل نظام تشغيل على برنامج يطبّق شيئًا يعادل النقر المزدوج على ملف مستند لفتحه، وهو البرنامج `start` على نظام ويندوز، والبرنامج `open` على نظام ماك macOS، والبرنامج `see` على نظام أوبنتو لينكس. أدخِل مثلًا ما يلي في الصدفة التفاعلية مع تمرير `'start'` أو `'open'` أو `'see'` إلى الدالة `Popen()` اعتمادًا على نظامك:

```
>>> fileObj = open('hello.txt', 'w')
>>> fileObj.write('Hello, world!')
12
>>> fileObj.close()
>>> import subprocess
>>> subprocess.Popen(['start', 'hello.txt'], shell=True)
```

كتبنا في مثالنا السابق عبارة `Hello, world!` في ملف hello.txt جديد، ثم استدعينا الدالة `Popen()`، ومرّرنا إليها قائمة تحتوي على اسم البرنامج (وهو `'start'` في مثالنا لنظام ويندوز) واسم الملف.

مرّرنا أيضًا وسيط الكلمات المفتاحية `shell=True`، ويُعَد هذا الوسيط مطلوبًا فقط على نظام ويندوز.

يعرِف نظام التشغيل جميع ارتباطات الملفات ويمكنه معرفة أنه يجب عليه تشغيل برنامج المفكرة `Notepad.exe` مثلًا للتعامل مع الملف `hello.txt`.

يُستخدَم البرنامج `open` لفتح ملفات المستندات والبرامج على نظام ماك macOS. إذًا لندخِل ما يلي في الصدفة التفاعلية إذا كان حاسوبك عليه نظام ماك، ويجب أن يفتح تطبيق الآلة الحاسبة:

```
>>> subprocess.Popen(['open', '/Applications/Calculator.app/'])
<subprocess.Popen object at 0x10202ff98>
```

17.9 تطبيق عملي: برنامج بسيط للعد التنازلي

يصعب العثور على تطبيق بسيط للمؤقت الزمني، وبالمثل قد يكون من الصعب العثور على تطبيق بسيط للعد التنازلي، إذًا لنكتب برنامجًا للعد التنازلي يشغّل المنبه في نهاية العد التنازلي.

إليك الخطوات العامة التي سيفعلها برنامجك:

1. العد التنازلي من العدد 60.
2. تشغيل ملف صوتي `alarm.wav` عندما يصل العد التنازلي إلى الصفر.

وبالتالي يجب أن تطبّق شيفرتك البرمجية الخطوات التالية:

1. التوقف مؤقتًا لمدة ثانية واحدة بين عرض كل عدد في العد التنازلي من خلال استدعاء الدالة `time.sleep()`.
2. استدعاء الدالة `subprocess.Popen()` لفتح الملف الصوتي باستخدام التطبيق الافتراضي.

افتح تبويبًا جديدًا في محرّرك لإنشاء ملف جديد واحفظه بالاسم `countdown.py`.

17.9.1 الخطوة الأولى: العد التنازلي

يتطلب هذا البرنامج الوحدة `time` لاستخدام الدالة `time.sleep()` ووحدة `subprocess` لاستخدام الدالة `subprocess.Popen()`.

أدخِل الآن الشيفرة البرمجية التالية واحفظ الملف بالاسم `countdown.py`:

```
#! python3
# countdown.py - سكربت بسيط للعد التنازلي

import time, subprocess

❶ timeLeft = 60
while timeLeft > 0:
    ❷ print(timeLeft, end='')
    ❸ time.sleep(1)
    ❹ timeLeft = timeLeft - 1

# تشغيل الملف الصوتي في نهاية العد التنازلي
```

استوردنا الوحدتين `time` و `subprocess`، ثم أنشأنا متغيرًا بالاسم `timeLeft` ليحتفظ بعدد الثواني المتبقية في العد التنازلي ❶. يمكن أن تبدأ عند القيمة 60، أو يمكنك تغيير القيمة إلى ما تريده، أو يمكنك ضبطها من وسيط سطر الأوامر. يمكنك في حلقة `while` عرض العدد المتبقي ❷، والتوقف مؤقتًا لمدة ثانية

واحدة ❸، ثم إنقاص المتغير `timeLeft` بمقدار 1 ❹ قبل بدء الحلقة مرة أخرى، وستستمر الحلقة في التكرار طالما أن المتغير `timeLeft` أكبر من 0، ثم سينتهي العد التنازلي.

17.9.2 الخطوة الثانية: تشغيل الملف الصوتي

توجد وحدات خارجية لتشغيل الملفات الصوتية بتنسيقات مختلفة، ولكن تمثّل الطريقة السريعة والسهلة لذلك في تشغيل أيّ تطبيق يستخدمه المستخدم لتشغيل الملفات الصوتية.

سيكتشف نظام التشغيل من امتداد الملف `.wav` التطبيقَ الذي يجب تشغيله لتشغيل الملف، ويمكن أن يكون ملف `.wav` له أحد تنسيقات الملفات الصوتية الأخرى مثل `.mp3` أو `.ogg`.

يمكنك استخدام أيّ ملف صوتي موجود على حاسوبك لتشغيله في نهاية العد التنازلي، أو يمكنك تنزيل alarm.wav. أضِف ما يلي إلى شيفرتك البرمجية:

```
#! python3
# countdown.py - سكربت بسيط للعد التنازلي

import time, subprocess

--snip--

# تشغيل الملف الصوتي في نهاية العد التنازلي
subprocess.Popen(['start', 'alarm.wav'], shell=True)
```

سيعمل الملف `alarm.wav` (أو الملف الصوتي الذي تختاره) بعد انتهاء الحلقة لإعلام المستخدم بانتهاء العد التنازلي. تأكّد من تضمين `'start'` في القائمة التي تمرّرها إلى الدالة `Popen()` وتمرير وسيط الكلمات المفتاحية `shell=True` على نظام ويندوز، وتأكّد من تمرير `'open'` بدلًا من `'start'` وإزالة `shell=True` على نظام ماك macOS.

يمكنك حفظ ملف نصي في مكانٍ ما مع رسالة مثل الرسالة "انتهى وقت الاستراحة!" بدلًا من تشغيل ملف صوتي، حيث يمكنك استخدام الدالة `Popen()` لفتحه في نهاية العد التنازلي، مما يؤدي إلى إنشاء نافذة منبثقة تحتوي على رسالة بفعالية، أو يمكنك استخدام الدالة `webbrowser.open()` لفتح موقع ويب محدَّد في نهاية العد التنازلي. يمكن أن يكون منبه برنامج العد التنازلي الخاص بك أيّ شيء تريده على عكس بعض تطبيقات العد التنازلي المجانية التي تجدها على الإنترنت.

17.9.3 أفكار لبرامج مماثلة

يمثّل العد التنازلي تأخيرًا بسيطًا قبل مواصلة تنفيذ البرنامج، ويمكنك استخدامه أيضًا لتطبيقات وميزات أخرى كما يلي:

- استخدام الدالة `time.sleep()` لمنح المستخدم فرصة الضغط على الاختصار `CTRL-C` لإلغاء إجراءٍ ما مثل حذف الملفات. يمكن لبرنامجك طباعة رسالة "اضغط على `CTRL-C` للإلغاء Press CTRL-C to cancel"، ثم معالجة أيّ استثناءات `KeyboardInterrupt` باستخدام تعليمات `try` و `except`.
- يمكنك استخدام كائنات `timedelta` مع العد التنازلي طويل الأجل لقياس عدد الأيام والساعات والدقائق والثواني حتى نقطة ما (مثل عيد ميلاد أو ذكرى سنوية) في المستقبل.

17.10 أسئلة للتدريب

استخدم المعلومات التي حصلتَ عليها من من هذا الفصل للإجابة عن الأسئلة التالية:

1. ما هو توقيت يونكس Unix epoch؟
2. ما هي الدالة التي تعيد عدد الثواني منذ توقيت يونكس؟
3. كيف يمكنك إيقاف برنامجك مؤقتًا لمدة 5 ثوانٍ بالضبط؟
4. ماذا تعيد الدالة `round()`؟
5. ما الفرق بين كائن `datetime` وكائن `timedelta`؟
6. ما هو يوم الأسبوع الذي كان يوم 7 من الشهر الأول من عام 2023 باستخدام الوحدة `datetime`؟
7. لنفترض أن لديك دالة اسمها `spam()`. كيف يمكنك استدعاء هذه الدالة وتشغيل الشيفرة البرمجية الموجودة ضمنها في خيط منفصل؟
8. ما الذي يجب فعله لتجنب مشكلات التزامن مع الخيوط المتعددة؟

17.11 مشاريع للتدريب

حاول كتابة البرامج التي تؤدي المهام التي سنوضّحها فيما يلي لكسب خبرة عملية أكبر من خلال استخدام المعلومات التي حصلتَ عليها من هذا الفصل.

17.11.1 برنامج المؤقت الزمني ولكن بمظهر أجمل

وسّع مشروع المؤقت الزمني Stopwatch من خلال استخدام توابع السلاسل النصية `rjust()` و`ljust()` "لتجميل" الخرج، فبدلًا من الخرج التالي:

```
Lap #1: 3.56 (3.56)
Lap #2: 8.63 (5.07)
Lap #3: 17.68 (9.05)
Lap #4: 19.11 (1.43)
```

سيبدو الخرج كما يلي:

```
Lap # 1:   3.56 (  3.56)
Lap # 2:   8.63 (  5.07)
Lap # 3:  17.68 (  9.05)
Lap # 4:  19.11 (  1.43)
```

لاحظ أنك ستحتاج إلى نسخٍ نوعها سلاسل نصية من المتغيرات `lapNum` و `lapTime` و `totalTime` التي نوعها أعداد صحيحة وأعداد عشرية لاستدعاء توابع السلاسل النصية عليها.

استخدم بعد ذلك وحدة `pyperclip` التي وضّحناها في الفصل السادس لنسخ الخرج النصي إلى الحافظة حتى يتمكّن المستخدم من لصق الخرج بسرعة في ملف نصي أو بريد إلكتروني.

17.11.2 برنامج لتنزيل القصص الهزلية المجدول على الويب

اكتب برنامجًا يفحص مواقع الويب الخاصة بالعديد من القصص الهزلية على الويب وينزّل الصور تلقائيًا عند تحديث القصص الهزلية عن آخر زيارة للبرنامج.

يمكن لمجدول نظام تشغيلك -مثل Scheduled Tasks على ويندوز وlaunchd على نظام ماكmacOS، وcron على نظام لينكس- تشغيل برنامج بايثون الخاص بك مرة واحدة يوميًا، ويمكن لبرنامج بايثون نفسه تنزيل القصة الهزلية ثم نسخها إلى سطح المكتب بحيث يسهل العثور عليها، مما يحرّرك من الاضطرار إلى التحقق من موقع الويب بنفسك للتأكد من تحديثه.

17.12 الخلاصة

يُعَد توقيت يونكس (1 من الشهر الأول من عام 1970 عند منتصف الليل بالتوقيت العالمي المنسَّق) وقتًا مرجعيًا معياريًا للعديد من لغات البرمجة بما في ذلك لغة بايثون. تعيد الوحدة الخاصة بالدالة `time.time()` علامة يونكس الزمنية (أي قيمة عشرية لعدد الثواني منذ توقيت يونكس)، ولكن تُعَد الوحدة `datetime` أفضل لإجراء العمليات الحسابية الرياضية على التاريخ وتنسيق أو تحليل السلاسل النصية باستخدام معلومات التاريخ.

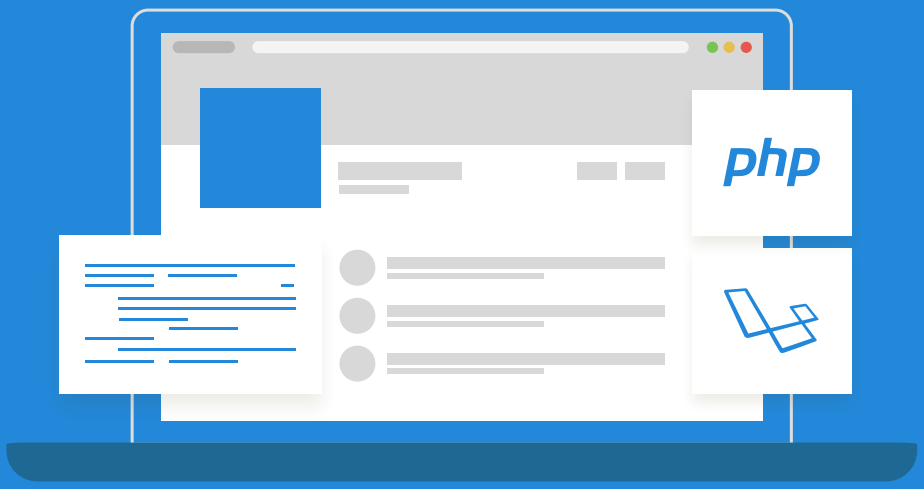
ستوقِف الدالة `time.sleep()` التنفيذ (أي لن تعود) لعددٍ معين من الثواني، حيث يمكنك استخدام ذلك لإضافة فترات توقف مؤقتة إلى برنامجك.

تُستخدَم الوحدة `threading` لإنشاء خيوط متعددة، والتي تُعَد مفيدةً عندما تريد تنزيل ملفات متعددة أو إنجاز مهام أخرى في وقت واحد، ولكن تأكد من أن الخيط يقرأ ويكتب المتغيرات المحلية فقط، وإلا فقد تواجه مشكلات في التزامن.

في الأخير، يمكن لبرامج بايثون الخاصة بك تشغيل تطبيقات أخرى باستخدام الدالة `subprocess.Popen()`، حيث يمكن تمرير وسطاء سطر الأوامر إلى استدعاء هذه الدالة لفتح مستندات محددة باستخدام التطبيق.

يمكنك أيضًا استخدام برنامج `start` أو `open` أو `see` مع الدالة `Popen()` لاستخدام ارتباطات الملفات الخاصة بحاسوبك لمعرفة التطبيق الذي سيُستخدَم لفتح مستند تلقائيًا. يمكن لبرامج بايثون الخاصة بك الاستفادة من إمكاناتها لتلبية احتياجات الأتمتة الخاصة بك باستخدام التطبيقات الأخرى الموجودة على حاسوبك.

دورة تطوير تطبيقات الويب
باستخدام لغة PHP



احترف تطوير النظم الخلفية وتطبيقات الويب
من الألف إلى الياء دون الحاجة لخبرة برمجية مسبقة

التحق بالدورة الآن

18. إرسال الرسائل باستخدام لغة بايثون

يحتاج التحقق من البريد الإلكتروني والرد عليه الكثير من الوقت، ولا يمكنك كتابة برنامج للتعامل مع جميع رسائل بريدك الإلكتروني، إذ تتطلب كل رسالة ردًا خاصًا بها، مع ذلك في إمكانك أتمتة الكثير من المهام الأخرى التي لها علاقة بالبريد الإلكتروني، وذلك بعد أن تعرف كيفية كتابة البرامج التي يمكنها إرسال واستقبال رسائل البريد الإلكتروني.

قد يكون لديك مثلًا جدول بيانات يحتوي على الكثير من سجلات العملاء وتريد إرسال رسالة تحتوي على استمارة Form خاصة بكل عميل اعتمادًا على تفاصيل عمره وموقعه، وقد لا تتمكن البرمجيات التجارية من فعل ذلك نيابةً عنك، ولكن يمكنك كتابة برنامجك الخاص لإرسال هذه الرسائل عبر البريد الإلكتروني، مما يوفّر عليك الكثير من الوقت لنسخ ولصق رسائل البريد الإلكتروني التي تحتوي على استمارات.

يمكنك أيضًا كتابة برامج لإرسال رسائل البريد الإلكتروني والرسائل النصية القصيرة SMS لإعلامك بالأشياء حتى عندما تكون بعيدًا عن حاسوبك.

إذا أجريتَ أتمتةً لمهمة تستغرق بضع ساعات لإنجازها، فلن ترغب في العودة إلى حاسوبك كل بضع دقائق للتحقق من حالة البرنامج، لذا يمكن لبرنامجك إرسال رسالة نصية إلى هاتفك عند الانتهاء فقط، مما يحرّرك من التركيز على أشياء أكثر أهمية عندما تكون بعيدًا عن حاسوبك.

سنوضّح بهذا الفصل الوحدة `EZGmail` التي تُعَد طريقة بسيطة لإرسال وقراءة رسائل البريد الإلكتروني من حسابات جيميل Gmail، وهي وحدة بايثون لاستخدام بروتوكولات البريد الإلكتروني المعيارية SMTP و IMAP.

ملاحظة: نوصي بشدة بإعداد حساب بريد إلكتروني منفصل لأيّ سكربتات ترسل أو تستقبل رسائل البريد الإلكتروني، مما يؤدي إلى منع الأخطاء الموجودة في برامجك من التأثير على حساب بريدك الإلكتروني الشخصي مثل حذف رسائل البريد الإلكتروني أو إرسال رسائل غير مرغوب بها Spam إلى جهات الاتصال الخاصة بك عن طريق الخطأ. يُفضَّل أن تجري أولًا تشغيلًا تجريبيًا من خلال تعليق الشيفرة البرمجية التي ترسل رسائل البريد الإلكتروني أو تحذفها فعليًا ووضع استدعاء مؤقت للدالة `print()` مكانها، وبالتالي يمكنك اختبار برنامجك قبل تشغيله تشغيلًا حقيقيًا.

18.1 التعامل مع رسائل البريد الإلكتروني عبر Gmail API

يمتلك جيميل ما يقرب من ثلث حصة سوق عملاء البريد الإلكتروني، إذ لا بد أنّ لديك عنوان بريد إلكتروني واحد على الأقل على جيميل. يتميز جيميل بتدابير الأمان الإضافية ومكافحة البريد الالكتروني غير المرغوب به، لذا من الأسهل التحكم في حساب جيميل باستخدام الوحدة `EZGmail` بدلًا من التحكم به باستخدام الوحدتين `smtplib` و `imapclient` اللتين سنناقشهما لاحقًا في هذا الفصل.

كتب Al Sweigart وحدة `EZGmail`، حيث تعمل هذه الوحدة فوق واجهة برمجة تطبيقات جيميل الرسمية وتوفّر دوالًا تسهّل استخدام جيميل من شيفرة بايثون.

اطّلع على تفاصيل `EZGmail` الكاملة على GitHub، حيث لا تنتِج جوجل هذه الوحدة وليست تابعة لها، واطّلع على التوثيق الرسمي لواجهة برمجة تطبيقات جيميل Gmail API.

يمكنك تثبيت وحدة `EZGmail` من خلال تشغيل الأمر `pip install --user --upgrade ezgmail` على نظام ويندوز، أو استخدم الأداة `pip3` على نظامي ماك macOS ولينكس Linux. يضمن الخيار `--upgrade` تثبيت أحدث إصدار من الحزمة، وهو أمر ضروري للتفاعل مع خدمة دائمة التغير عبر الإنترنت مثل واجهة برمجة تطبيقات جيميل.

18.1.1 تفعيل واجهة برمجة تطبيقات جيميل

يجب عليك أولًا التسجيل للحصول على حساب بريد إلكتروني على جيميل قبل أن تكتب شيفرتك البرمجية. انتقل بعد ذلك إلى صفحة البدء السريع لاستخدام بايثون، وانقر على زر تفعيل واجهة برمجة تطبيقات جيميل Enable the Gmail API في تلك الصفحة، واملأ الاستمارة التي ستظهر.

ستقدم الصفحة رابطًا للملف `credentials.json` بعد ملء الاستمارة، حيث يجب أن تنزّل هذا الملف وتضعه في المجلد نفسه لملف `.py` الخاص بك. يحتوي الملف `credentials.json` على معرّف العميل Client ID ومعلومات العميل السرية Client Secret، والتي يجب عليك التعامل معها مثل كلمة مرور حسابك على جيميل وعدم مشاركتها مع أيّ شخص آخر.

لندخِل الآن الشيفرة البرمجية التالية في الصدفة التفاعلية Interactive Shell:

```
>>> import ezgmail, os
>>> os.chdir(r'C:\path\to\credentials_json_file')
>>> ezgmail.init()
```

تأكّد من ضبط مجلد العمل الحالي على المجلد نفسه الذي يوجد بـه الملـف `credentials.json` وأنـك متصل بالإنترنت. تفتح الدالة `ezgmail.init()` متصفحك على صفحة تسجيل الدخول إلى جوجل، لذا أدخِل عنوان جيميل وكلمة مرورك. قد تحذّرك الصفحة بعدم التحقق من هذا التطـبيق "This app isn't verified"، ولكن لا بأس بذلك. انقر بعد ذلك على "خيارات متقدمة Advanced"، وانقر على خيار الانتقال إلى صفحة البـدء السريع (غير آمن) "Go to Quickstart (unsafe)". (إذا أردتَ كتابة سكربتات بايثون لأشخاص آخرين ولا تريد ظهور هذا التحذير لهم، فيجب أن تتعرّف على عمليـة التحقـق من تطـبيق جوجـل، والـتي لن نناقشـها في هـذا الفصل. انقر على خيار "السماح Allow" ثم أغلق المتصفح عندما تعرض الصفحةُ التاليـة الرسـالةَ "تريـد صـفحة البدء السريع الوصول إلى حسابك جوجل Quickstart wants to access your Google Account".

يتولّد بعد ذلك ملف `token.json` لمنح سكربتات بايثون الخاصة بك إمكانية الوصول إلى حساب جيميل الـذي أدخلتـه، ولن يفتح المتصـفح إلّا على صـفحة تسـجيل الـدخول إن لم يتمكّن من العثـور على ملـف `credentials.json` موجودٍ مسبقًا. يمكن لسكربتات بايثون الخاصة بك باسـتخدام الملفين `token.json` و `token.json` إرسالَ رسائل البريد الإلكتروني وقراءتها من حسابك على جيميل دون مطالبتك بتضمين كلمـة مرور جيميل في شيفرتك المصدرية.

18.1.2 إرسال رسائل البريد الإلكتروني من حساب جيميل

يجب أن تكون وحدة `EZGmail` قادرةً على إرسال بريد إلكتروني باستخدام استدعاء دالةٍ واحد بعـد حصـولك على الملف `token.json` كما يلي:

```
>>> import ezgmail
>>> ezgmail.send('recipient@example.com', 'Subject line', 'Body of the
email')
```

إذا أردتَ إرفاق ملفاتٍ ببريدك الإلكتروني، فيمكنك توفير وسيط قائمة إضافي للدالة `send()`:

```
>>> ezgmail.send('recipient@example.com', 'Subject line', 'Body of the
email',
['attachment1.jpg', 'attachment2.mp3'])
```

لاحظ أنه -كجزء من ميزات الأمان ومكافحة الرسائل غـير المرغـوب بهـا- قـد لا يرسـل جيميـل رسـائل بريـد إلكتروني متكررة تحتوي على النص نفسه لأنها يمكن أن تكون رسائلًا غير مرغوب بهـا، أو رسـائل بريـد إلكـتروني تحتوي على مرفقات ملفات لها الامتداد `.exe` أو `.zip` لأنها يمكن أن تكون فيروسات.

يمكنك أيضًا توفير وسائط الكلمات المفتاحية Keyword Arguments الاختيارية `cc` و `bcc` لإرسال نسخ مطابقة Carbon Copies ونسخ مطابقة مخفية Blind Carbon Copies:

```
>>> import ezgmail
>>> ezgmail.send('recipient@example.com', 'Subject line', 'Body of the
email',
cc='friend@example.com',
bcc='otherfriend@example.com,someoneelse@example.com')
```

إذا أردتَ أن تتذكر عنوان جيميل الذي ضُبِط الملف `token.json` عليه، فيمكنك فحص المتغير `ezgmail.EMAIL_ADDRESS`، حيث يُملَأ هذا المتغير فقط بعد استدعاء الدالة `ezgmail.init()` أو أي دالة أخرى خاصة بالوحدة `EZGmail`.

```
>>> import ezgmail
>>> ezgmail.init()
>>> ezgmail.EMAIL_ADDRESS
'example@gmail.com'
```

تأكّد من التعامل مع الملف `token.json` بالطريقة نفسها للتعامل مع كلمة مرورك، حيث إذا حصل شخصٌ آخر على هذا الملف، فيمكنه الوصول إلى حسابك على جيميل بالرغم من أنه لن يتمكّن من تغيير كلمة مرور حسابك على جيميل. يمكنك إبطال ملفات `token.json` الصادرة مسبقًا من خلال الانتقال إلى الرابط `https://security.google.com/settings/security/permissions?pli=1/`، ثم أبطِل الوصول إلى تطبيق البدء السريع Quickstart، ولكن يجب تشغيل الدالة `ezgmail.init()` ومتابعة عملية تسجيل الدخول مرة أخرى للحصول على ملف `token.json` جديد.

18.1.3 قراءة رسائل البريد الإلكتروني من حساب جيميل

ينظّم جيميل رسائل البريد الإلكتروني التي تمثل ردودًا على بعضها البعض ضمن سلاسل محادثات Conversation Threads. إذا سجّلتَ الدخول إلى جيميل في متصفح الويب أو من خلال أحد التطبيقات، فسترى سلاسل رسائل البريد الإلكتروني بدلًا من رسائل البريد الإلكتروني الفردية، حتى لو احتوت إحدى تلك السلاسل على رسالة بريد إلكتروني واحدة فقط.

تحتوي الوحدة `EZGmail` على كائنات `GmailThread` و `GmailMessage` لتمثيل سلاسل المحادثات ورسائل البريد الإلكتروني الفردية على التوالي، ويحتوي الكائن `GmailThread` على سمةٍ Attribute هي السمة `messages` التي تحتوي على قائمة بكائنات `GmailMessage`. تعيد الدالة `unread()` قائمةً بكائنات `GmailThread` لجميع رسائل البريد الإلكتروني غير المقروءة، والتي يمكن بعد ذلك تمريرها إلى الدالة `ezgmail.summary()` لطباعة ملخصٍ لسلاسل المحادثات في تلك القائمة:

```
>>> import ezgmail
>>> unreadThreads = ezgmail.unread() # قائمة بكائنات GmailThread
>>> ezgmail.summary(unreadThreads)
Al, Jon - Do you want to watch RoboCop this weekend? - Dec 09
Jon - Thanks for stopping me from buying Bitcoin. - Dec 09
```

تُعَد الدالة `summary()` مفيدة لعرض ملخصٍ سريع لسلاسل رسائل البريد الإلكتروني، ولكن يمكنك الوصول إلى رسائل محددة (وأجزاء منها) من خلال فحص السمة `messages` الخاصة بالكائن `GmailThread`، حيث تحتوي هذه السمة على قائمة بكائنات `GmailMessage` التي تشكّل سلسلة المحادثات، وتحتوي هذه الكائنات على السمات `subject` و `body` و `timestamp` و `sender` و `recipient` والتي بدورها توصف البريد الإلكتروني.

```
>>> len(unreadThreads)
2
>>> str(unreadThreads[0])
"<GmailThread len=2 snippet= Do you want to watch RoboCop this
weekend?'>"
>>> len(unreadThreads[0].messages)
2
>>> str(unreadThreads[0].messages[0])
"<GmailMessage from='Al Sweigart <al@inventwithpython.com>' to='Jon
Doe
<example@gmail.com>' timestamp=datetime.datetime(2018, 12, 9, 13, 28,
48)
subject='RoboCop' snippet='Do you want to watch RoboCop this
weekend?'>"
>>> unreadThreads[0].messages[0].subject
'RoboCop'
>>> unreadThreads[0].messages[0].body
'Do you want to watch RoboCop this weekend?\r\n'
>>> unreadThreads[0].messages[0].timestamp
datetime.datetime(2018, 12, 9, 13, 28, 48)
>>> unreadThreads[0].messages[0].sender
'Al Sweigart <al@inventwithpython.com>'
>>> unreadThreads[0].messages[0].recipient
'Jon Doe <example@gmail.com>'
```

تعيد الدالة `ezgmail.recent()` أحدث 25 سلسلة محادثات في حسابك على جيميل كما تفعل الدالة `ezgmail.unread()`، ولكن يمكنك تمرير وسيط الكلمات المفتاحية `maxResults` الاختياري لتغيير هذا الحد كما يلي:

```
>>> recentThreads = ezgmail.recent()
>>> len(recentThreads)
25
>>> recentThreads = ezgmail.recent(maxResults=100)
>>> len(recentThreads)
46
```

18.1.4 البحث عن رسائل البريد الإلكتروني في حساب جيميل

يمكنك البحث عن رسائل بريد إلكتروني محددة باستخدام الطريقة نفسها التي تستخدمها لإدخال استعلامات في مربع البحث على جيميل من خلال استدعاء الدالة `ezgmail.search()`:

```
>>> resultThreads = ezgmail.search('RoboCop')
>>> len(resultThreads)
1
>>> ezgmail.summary(resultThreads)
Al, Jon - Do you want to watch RoboCop this weekend? - Dec 09
```

يجب أن يؤدي الاستدعاء السابق للدالة `search()` إلى النتائج نفسها عندما تدخل الكلمة "RoboCop" في مربع البحث كما في الشكل التالي:

الشكل 90: البحث عن رسائل البريد الإلكتروني "RoboCop" في موقع جيميل الإلكتروني

تعيد الدالة `search()` قائمةً بكائنات `GmailThread` كما تفعل الدالتان `unread()` و `recent()`، ويمكنك أيضًا تمرير أيٍّ من معاملات البحث الخاصة التي يمكنك إدخالها في مربع البحث إلى الدالة `search()` مثل المعاملات التالية:

- `'label:UNREAD'`: لرسائل البريد الإلكتروني غير المقروءة.

- `'from:al@inventwithpython.com'`: من أجل رسائل البريد الإلكتروني الواردة من `al@inventwithpython.com`.
- `'subject:hello'`: لرسائل البريد الإلكتروني التي تحتوي على الكلمة "`hello`" في موضوعها.
- `'has:attachment'`: لرسائل البريد الإلكتروني التي تحتوي على ملفات مرفقة.

ملاحظة: اطّلع على القائمة الكاملة لمعاملات البحث.

18.1.5 تنزيل المرفقات من حساب جيميل

تحتوي كائنات `GmailMessage` على السمة `attachments`، والتي هي قائمة بأسماء الملفات المُرفَقة مع الرسالة، حيث يمكنك تمرير أيٍّ من هذه الأسماء إلى التابع `downloadAttachment()` الخاص بكائن `GmailMessage` لتنزيل الملفات، ويمكنك أيضًا تنزيلها جميعًا دفعةً واحدة باستخدام التابع `downloadAllAttachments()`. تحفظ الوحدة `EZGmail` المرفقات في مجلد العمل الحالي افتراضيًا، ولكن يمكنك تمرير وسيط الكلمات المفتاحية الإضافي `downloadFolder` إلى التابعين `downloadAttachment()` و `downloadAllAttachments()` أيضًا لتنزيل المجلد. لندخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import ezgmail
>>> threads = ezgmail.search('vacation photos')
>>> threads[0].messages[0].attachments
['tulips.jpg', 'canal.jpg', 'bicycles.jpg']
>>> threads[0].messages[0].downloadAttachment('tulips.jpg')
>>>
threads[0].messages[0].downloadAllAttachments(downloadFolder='vacat
ion2023')
['tulips.jpg', 'canal.jpg', 'bicycles.jpg']
```

إذا وُجِد ملف يحمل اسم الملف المرفق نفسه، فسيحل الملف المرفق الذي نزّلناه محله تلقائيًا.

تحتوي الوحدة `EZGmail` على ميزات إضافية، إذ يمكنك العثور عليها ضمن توثيقها الكامل على Github.

18.2 بروتوكول SMTP

تستخدم الحواسيب بروتوكول HTTP لإرسال صفحات الويب عبر الإنترنت، ويُستخدَم بروتوكول نقل البريد البسيط Simple Mail Transfer Protocol -أو SMTP اختصارًا- لإرسال البريد الإلكتروني، حيث يتمتع هذان البروتوكولان بالمقدار نفسه من الأهمية.

يحدّد بروتوكول SMTP كيفية تنسيق رسائل البريد الإلكتروني وتشفيرها ونقلها بين خوادم البريد وجميع التفاصيل الأخرى التي يعالجها حاسوبك بعد النقر على زر الإرسال، ولكنك لست بحاجة إلى معرفة هذه التفاصيل التقنية، لأن الوحدة `smtplib` الخاصة بلغة بايثون تبسّطها إلى بضع دوال.

يتعامل بروتوكول SMTP فقط مع إرسال رسائل البريد الإلكتروني إلى المستخدمين الآخرين، ويتعامل بروتوكول مختلف هو بروتوكول IMAP مع استرداد رسائل البريد الإلكتروني المرسَلة إليك، حيث سنوضّح هذا البروتوكول لاحقًا.

يوفّر معظم مزوّدي خدمات البريد الإلكتروني المستندة إلى الويب -بالإضافة إلى بروتوكولَي SMTP وIMAP- حاليًا إجراءات أمنية أخرى للحماية من البريد غير المرغوب به والتصيد الاحتيالي Phishing واستخدامات البريد الإلكتروني الضارة الأخرى.

تمنع هذه الإجراءات سكربتات بايثون من تسجيل الدخول إلى حساب بريد إلكتروني باستخدام وحدتي `smtplib` و `imapclient`، ولكن تحتوي العديد من هذه الخدمات على واجهات برمجة التطبيقات API ووحدات بايثون محددة تسمح للسكربتات بالوصول إليها.

سنشرح في هذا الفصل الوحدة الخاصة بخدمة جيميل، ولكنك ستحتاج إلى الرجوع إلى التوثيق الرسمي للخدمات الأخرى.

18.3 إرسال البريد الإلكتروني

قد تكون على دراية بإرسال رسائل البريد الإلكتروني من أوت لوك Outlook أو ثندربرد Thunderbird أو من خلال موقع ويب مثل جيميل Gmail أو بريد ياهو Yahoo Mail، ولكن لسوء الحظ لا تقدّم لغة بايثون واجهة مستخدم رسومية جميلة مثل تلك التي تقدّمها هذه الخدمات، لذا يمكنك بدلًا من ذلك استدعاء الدوال لإجراء الخطوات الرئيسية من بروتوكول SMTP كما هو موضّح في مثال الصدفة التفاعلية الآتي.

> **ملاحظة**: لا تدخِل المثال التالي في الصدفة التفاعلية، إذ لن ينجح الأمر، لأن `smtp.example.com` و`bob@example.com` و `MY_SECRET_PASSWORD` و `alice@example.com` هي عناصر بديلة، إذ تُعَد هذه الشيفرة البرمجية مجرد نظرة عامة على عملية إرسال بريد إلكتروني باستخدام بايثون.

```
>>> import smtplib
>>> smtpObj = smtplib.SMTP('smtp.example.com', 587)
>>> smtpObj.ehlo()
(250, b'mx.example.com at your service, [216.172.148.131]\nSIZE
35882577\
n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nCHUNKING')
```

```
>>> smtpObj.starttls()
(220, b'2.0.0 Ready to start TLS')
>>> smtpObj.login('bob@example.com', 'MY_SECRET_PASSWORD')
(235, b'2.7.0 Accepted')
>>> smtpObj.sendmail('bob@example.com', 'alice@example.com', 'Subject:
So
long.\nDear Alice, so long and thanks for all the fish. Sincerely,
Bob')
{}
>>> smtpObj.quit()
(221, b'2.0.0 closing connection ko10sm23097611pbd.52 - gsmtp')
```

سنوضّح في الأقسام التالية كل خطوة من هذه الشيفرة البرمجية مع استبدال العناصر البديلة بمعلوماتك للاتصال بخادم SMTP وتسجيل الدخول إليه، وإرسال بريد إلكتروني، وقطع الاتصال بالخادم.

18.3.1 الاتصال بخادم SMTP

إذا أعددتَ ثندربيرد أو أوت لوك أو أي برنامج آخر للاتصال بحساب بريدك الإلكتروني، فقد تكون على دراية بضبط خادم ومنفذ SMTP، حيث ستكون هذه الإعدادات مختلفة بحسب مزوّد البريد الإلكتروني، ولكن يجب أن يمكنك البحث عبر الويب عن إعدادات مزوّدك لبروتوكول SMTP للحصول على الخادم والمنفذ لاستخدامهما.

يكون عادةً اسم النطاق Domain لخادم SMTP هو اسم نطاق مزوّد بريدك الإلكتروني مع وجود البادئة `smtp.` قبله، فمثلًا خادم SMTP الخاص بشركة Verizon موجودٌ على النطاق `smtp.verizon.net`.

يسرد الجدول الآتي بعضًا من مزوّدي البريد الإلكتروني وخوادم SMTP الخاصة بهم، حيث يُعَد المنفذ Port قيمةً صحيحة وتكون دائمًا تقريبًا 587، ويستخدمه معيار تشفير الأوامر TLS.

مزوّد البريد الإلكتروني	اسم نطاق خادم SMTP
Gmail*	اسم النطاق smtp.gmail.com
Outlook.com/Hotmail.com*	اسم النطاق smtp-mail.outlook.com
Yahoo Mail*	اسم النطاق smtp.mail.yahoo.com
AT&T	اسم النطاق smpt.mail.att.net (المنفذ 465)
Comcast	اسم النطاق smtp.comcast.net
Verizon	اسم النطاق smtp.verizon.net (المنفذ 465)

الجدول 23: الاتصال بخادم SMTP للعديد من مزوّدي البريد الإلكتروني

> **ملاحظة**: تمنع الإجراءات الأمنية الإضافية شيفرة بايثون من تسجيل الدخول إلى هذه الخوادم التي وضعنا بجانب اسمها المحرف (*) باستخدام الوحدة `smtplib`، ولكن يمكن لوحدة EZGmail تجاوز هذه الصعوبة لحسابات جيميل.

إذا حصلتَ على اسم النطاق ومعلومات المنفذ لمزوّد بريدك الإلكتروني، فيمكنك إنشاء كائن `SMTP` من خلال استدعاء الدالة `smptlib.SMTP()`، وتمرير اسم النطاق كوسيط من نوع السلسلة النصية والمنفذ كوسيط من نوع عدد صحيح إليها. يمثل الكائن `SMTP` اتصالًا بخادم بريد SMTP ويمتلك توابع لإرسال رسائل البريد الإلكتروني، فمثلًا ينشئ الاستدعاء التالي كائن `SMTP` للاتصال بخادم بريد إلكتروني وهمي:

```
>>> smtpObj = smtplib.SMTP('smtp.example.com', 587)
>>> type(smtpObj)
<class 'smtplib.SMTP'>
```

يُظهِر إدخال الدالة `type(smtpObj)` وجود كائن `SMTP` مخزّنٍ في المتغير `smtpObj`، حيث ستحتاج إلى هذا الكائن لاستدعاء التوابع التي تسجل دخولك وترسل رسائل البريد الإلكتروني. إن لم ينجح استدعاء الدالة `smtplib.SMTP()`، فقد لا يدعم خادم SMTP الخاص بك بروتوكول TLS على المنفذ 587، وبالتالي يجب إنشاء كائن `SMTP` باستخدام الدالة `smtplib.SMTP_SSL()` والمنفذ 465 بدلًا من ذلك.

```
>>> smtpObj = smtplib.SMTP_SSL('smtp.example.com', 465)
```

> **ملاحظة**: إن لم تكن متصلًا بالإنترنت، فسترفع شيفرة بايثون استثناء على النحو الآتي أو أيّ استثناء آخر مشابه.
> `socket.gaierror: [Errno 11004] getaddrinfo failed`

لا تُعَد الاختلافات بين بروتوكولَي TLS و SSL مهمة بالنسبة لبرامجك، فما عليك سوى معرفة معيار التشفير الذي يستخدمه خادم SMTP الخاص بك حتى تعرف كيفية الاتصال به. سيحتوي المتغير `smtpObj` في كافة أمثلة الصدفة التفاعلية التالية على كائن `SMTP` الذي تعيده الدالة `smtplib.SMTP()` أو الدالة `smtplib.SMTP_SSL()`.

18.3.2 إرسال رسالة الترحيب "Hello" الخاصة ببروتوكول SMTP

إذا حصلنا على كائن `SMTP`، فيمكننا استدعاء التابع `ehlo()` للترحيب بخادم البريد الإلكتروني SMTP، حيث يُعَد هذا الترحيب الخطوة الأولى في بروتوكول SMTP وهو مهم لتأسيس اتصال مع الخادم.

لا حاجة لمعرفة تفاصيل هذه البروتوكولات، ولكن تأكّد من استدعاء التابع `ehlo()` أولًا بعد الحصول على كائن `SMTP`، وإلّا ستؤدي استدعاءات التوابع اللاحقة إلى حدوث أخطاء. إليك مثال لاستدعاء التابع `ehlo()` وقيمته المُعادة:

```
>>> smtpObj.ehlo()
(250, b'mx.example.com at your service, [216.172.148.131]\nSIZE
35882577\
n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nCHUNKING')
```

إذا كان العنصر الأول في المجموعة Tuple المُعادة هو العدد الصحيح 250 (رمز النجاح في بروتوكول SMTP)، فهذا يعني أن الترحيب قد نجح.

18.3.3 بدء تشفير TLS

إن كنتَ متصلًا بالمنفذ 587 على خادم SMTP (أي أنك تستخدم تشفير TLS)، فيجب عليك استدعاء التابع `starttls()` لاحقًا، إذ تؤدي هذه الخطوة المطلوبة إلى تفعيل التشفير على اتصالك. إذا كنت متصلًا بالمنفذ 465 (أي أنك تستخدم بروتوكول SSL)، فهذا يعني أن التشفير مُعَد مسبقًا، وأنه يجب عليك تخطي هذه الخطوة.

إليك مثال لاستدعاء التابع `starttls()`:

```
>>> smtpObj.starttls()
(220, b'2.0.0 Ready to start TLS')
```

يضع التابع `starttls()` اتصال SMTP الخاص بك في وضع TLS، ويخبرك العدد 220 الموجود في القيمة المُعادة أن الخادم جاهز.

18.3.4 تسجيل الدخول إلى خادم SMTP

إذا أعددتَ اتصالك المشفّر بخادم SMTP، فيمكنك تسجيل الدخول باستخدام اسم المستخدم الخاص بك (وهو عنوان بريدك الإلكتروني عادةً) وكلمة مرور بريدك الإلكتروني من خلال استدعاء التابع `login()`.

```
>>> smtpObj.login('my_email_address@example.com',
'MY_SECRET_PASSWORD')
(235, b'2.7.0 Accepted')
```

مرّر سلسلةً نصية تمثّل عنوان بريدك الإلكتروني كوسيطٍ أول وسلسلة نصية تمثّل كلمة مرورك كوسيطٍ ثانٍ إلى التابع `login()`، وتعني القيمة `235` الموجودة في القيمة المُعادة أن الاستيثاق Authentication ناجح. ترفع شيفرة بايثون الاستثناء `smtplib.SMTPAuthenticationError` لكلمات المرور غير الصحيحة.

> **ملاحظة**: كن حذرًا بشأن وضع كلمات المرور في شيفرتك المصدرية، حيث إذا نسخ شخصٌ ما برنامجك، فسيكون بإمكانه الوصول إلى حساب بريدك الإلكتروني، لذا يُفضَّل استدعاء الدالة `input()` وجعل المستخدم يكتب كلمة المرور. قد يكون اضطرارك إلى إدخال كلمة المرور في كل مرة تشغّل فيها برنامجك أمرًا غير مريح،

> ولكن تمنعك هذه الطريقة من ترك كلمة مرورك في ملف غير مشفّر على حاسوبك بحيث يمكن للمخترق أو للص الذي يسرق حاسوبك المحمول مثلًا الحصول عليها بسهولة.

18.3.5 إرسال رسالة عبر البريد الإلكتروني

سجّلنا الدخول إلى خادم SMTP الخاص بمزوّد بريدك الإلكتروني، وبالتالي يمكننا الآن استدعاء التابع `sendmail()` لإرسال البريد الإلكتروني فعليًا، حيث يبدو استدعاء هذا التابع كما يلي:

```
>>> smtpObj.sendmail('my_email_address@example.com
', 'recipient@example.com', 'Subject: So long.\nDear Alice, so long
and thanks for all the fish.
Sincerely, Bob')
{}
```

يتطلب التابع `sendmail()` ثلاثة وسطاء هي:

- عنوان بريدك الإلكتروني كسلسلة نصية (للعنوان "مِن" from الخاص بالبريد الإلكتروني).
- عنوان البريد الإلكتروني للمستلم كسلسلة نصية أو قائمةً من السلاسل النصية لمستلمين متعددين (للعنوان "إلى" to).
- نص Body البريد الإلكتروني كسلسلة نصية.

يجب أن تبدأ السلسلة النصية لنص البريد الإلكتروني بالعبارة `'Subject: \n'` لسطر موضوع البريد الإلكتروني، حيث يفصل محرف السطر الجديد `'\n'` سطر الموضوع عن النص الرئيسي للبريد الإلكتروني.

القيمة المُعادة من التابع `sendmail()` هي قاموس، إذ سيكون هناك زوج مفتاح-قيمة واحد في القاموس لكل مستلمٍ فشل تسليم البريد الإلكتروني إليه، ويعني القاموس الفارغ أن البريد الإلكتروني أُرسِل بنجاح إلى جميع المستلمين.

18.3.6 قطع الاتصال بخادم SMTP

تأكّد من استدعاء التابع `quit()` عند الانتهاء من إرسال رسائل البريد الإلكتروني، مما يؤدي إلى قطع اتصال برنامجك بخادم SMTP.

```
>>> smtpObj.quit()
(221, b'2.0.0 closing connection ko10sm23097611pbd.52 - gsmtp')
```

تعني القيمة 221 الموجودة في القيمة المُعادة انتهاءَ الجلسة.

18.4 البروتوكول IMAP

يُعَد البروتوكول SMTP بروتوكول إرسالٍ لرسائل البريد الإلكتروني، ولكن يحدّد بروتوكول الوصول إلى رسائل الإنترنت Internet Message Access Protocol -أو IMAP اختصارًا- كيفية الاتصال بخادم مزوّد البريد الإلكتروني لاسترداد رسائل البريد الإلكتروني المُرسَلة إلى عنوان بريدك الإلكتروني.

تحتوي لغة بايثون على الوحدة `imaplib`، ولكن تُعَد الوحدة `imapclient` الخارجية أسهل في الاستخدام. يقدّم هذا الفصل مقدمة لاستخدام الوحدة `IMAPClient`، لذا اطّلع على توثيقها الرسمي الكامل على موقعها الرسمي.

تنزّل الوحدة `imapclient` رسائل البريد الإلكتروني من خادم IMAP بتنسيقٍ معقد إلى حد ما، لذا قد تحتاج إلى تحويلها من هذا التنسيق إلى قيم سلاسل نصية بسيطة.

تنفّذ الوحدة `pyzmail` المهمة الصعبة المتمثلة في تحليل رسائل البريد الإلكتروني نيابةً عنك، لذا اطّلع على التوثيق الكامل لهذه الوحدة.

ثبّت الوحدتين `imapclient` و `pyzmail` من النافذة الطرفية Terminal باستخدام الأمرين `pip install --user -U imapclient==2.1.0` و `pip install --user -U pyzmail36==1.0.4` على نظام ويندوز Windows، أو باستخدام الأداة `pip3` على نظامي ماك macOS ولينكس Linux.

18.5 استرداد وحذف رسائل البريد الإلكتروني عبر بروتوكول IMAP

يُعَد البحث عن بريد إلكتروني واسترداده في لغة بايثون عملية متعددة الخطوات وتتطلب كلًا من الوحدتين الخارجيتين `imapclient` و `pyzmail`. إليك مثال كامل لتسجيل الدخول إلى خادم IMAP والبحث عن رسائل البريد الإلكتروني وجلبها، ثم استخراج نص رسائل البريد الإلكتروني منها:

```
>>> import imapclient
>>> imapObj = imapclient.IMAPClient('imap.example.com', ssl=True)
>>> imapObj.login('my_email_address@example.com',
'MY_SECRET_PASSWORD')
'my_email_address@example.com Jane Doe authenticated (Success)'
>>> imapObj.select_folder('INBOX', readonly=True)
>>> UIDs = imapObj.search(['SINCE 05-Jul-2023'])
>>> UIDs
[40032, 40033, 40034, 40035, 40036, 40037, 40038, 40039, 40040, 40041]
>>> rawMessages = imapObj.fetch([40041], ['BODY[]', 'FLAGS'])
>>> import pyzmail
```

```
>>> message = pyzmail.PyzMessage.factory(rawMessages[40041]
[b'BODY[]'])
>>> message.get_subject()
'Hello!'
>>> message.get_addresses('from')
[('Edward Snowden', 'esnowden@nsa.gov')]
>>> message.get_addresses('to')
[('Jane Doe', 'jdoe@example.com')]
>>> message.get_addresses('cc')
[]
>>> message.get_addresses('bcc')
[]
>>> message.text_part != None
True
>>> message.text_part.get_payload().decode(message.text_part.charset)
'Follow the money.\r\n\r\n-Ed\r\n'
>>> message.html_part != None
True
>>> message.html_part.get_payload().decode(message.html_part.charset)
'<div dir="ltr"><div>So long, and thanks for all the
fish!<br><br></div>-
Al<br></div>\r\n'
>>> imapObj.logout()
```

لا حاجة لحفظ جميع هذه الخطوات، إذ يمكنك العودة إلى هذا المثال العام لتحديث ذاكرتك بعد أن استعراض جميع الخطوات بالتفصيل.

18.5.1 الاتصال بخادم IMAP

احتجنا كائن `SMTP` للاتصال بخادم SMTP وإرسال البريد الإلكتروني، وبالمثل نحتاج كائن `IMAPClient` للاتصال بخادم IMAP وتلقي البريد الإلكتروني، ولكن يجب أولًا الحصول على اسم النطاق لخادم IMAP الخاص بمزوّد بريدك الإلكتروني، والذي سيكون مختلفًا عن اسم نطاق خادم SMTP.

يوضّح الجدول الآتي خوادم IMAP للعديد من مزوّدي البريد الإلكتروني:

مزوّد البريد الإلكتروني	اسم نطاق خادم IMAP
Gmail*	اسم النطاق imap.gmail.com
Outlook.com/Hotmail.com*	اسم النطاق imap-mail.outlook.com
Yahoo Mail*	اسم النطاق imap.mail.yahoo.com
AT&T	اسم النطاق imap.mail.att.net
Comcast	اسم النطاق imap.comcast.net
Verizon	اسم النطاق incoming.verizon.net

الجدول 24: خوادم IMAP للعديد من مزوّدي البريد الإلكتروني

> **ملاحظة**: تمنع الإجراءات الأمنية الإضافية شيفرة بايثون من تسجيل الدخول إلى هذه الخوادم التي وضعنا بجانب اسمها المحرف (*) باستخدام الوحدة `imapclient`.

نحصل على اسم النطاق لخادم IMAP، بعدها يمكننا استدعاء الدالة `imapclient.IMAPClient()` لإنشاء كائن `IMAPClient`.

يتطلب معظم مزوّدي البريد الإلكتروني تشفير SSL، لذا مرّر وسيط الكلمات المفتاحية `ssl=True` إلى هذه الدالة، ولندخل مثلًا ما يلي في الصدفة التفاعلية مع استخدام اسم النطاق الخاص بمزوّدك:

```
>>> import imapclient
>>> imapObj = imapclient.IMAPClient('imap.example.com', ssl=True)
```

سيحتوي المتغير `imapObj` على كائن `IMAPClient` الذي تعيده الدالة `imapclient.IMAPClient()` في كافة أمثلة الصدفة التفاعلية الموجودة في الأقسام التالية، والعميل Client هو الكائن الذي يتصل بالخادم.

18.5.2 تسجيل الدخول إلى خادم IMAP

نحصل على كائن `IMAPClient`، ثم يمكننا استدعاء التابع `login()` الخاص بهذا الكائن، وتمرير اسم المستخدم (وهو عنوان بريدك الإلكتروني عادةً) وكلمة المرور كسلاسل نصية إلى هذا التابع.

```
>>> imapObj.login('my_email_address@example.com',
'MY_SECRET_PASSWORD')
'my_email_address@example.com Jane Doe authenticated (Success)'
```

> **ملاحظة**: تذكّر ألّا تكتب كلمة المرور مباشرة في شيفرتك البرمجية، لذا صمّم برنامجك لقبول كلمة المرور التي تعيدها الدالة `input()`.

إذا رفض خادم IMAP اسم المستخدم/كلمة المرور، فسترفع شيفرة بايثون استثناء `imaplib.error`.

18.5.3 البحث عن رسالة البريد الإلكتروني

تُعَد عملية استرداد البريد الإلكتروني التي تهمك عمليةً مكونة من خطوتين بعد أن تسجّل الدخول، حيث يجب أولًا تحديد المجلد الذي تريد البحث فيه، ثم يجب استدعاء التابع `search()` الخاص بكائن `IMAPClient` وتمرير السلسلة النصية التي تمثّل الكلمات المفتاحية للبحث باستخدام بروتوكول IMAP.

ا. تحديد المجلد

يحتوي كل حساب تقريبًا على مجلد البريد الوارد `INBOX` افتراضيًا، ولكن يمكنك أيضًا الحصول على قائمة المجلدات من خلال استدعاء التابع `list_folders()` الخاص بالكائن `IMAPClient`، مما يؤدي إلى إعادة قائمة من المجموعات Tuples، حيث تحتوي كل مجموعة على معلومات حول مجلد واحد. تابع مثال الصدفة التفاعلية من خلال إدخال ما يلي:

```
>>> import pprint
>>> pprint.pprint(imapObj.list_folders())
[(('\\HasNoChildren',), '/', 'Drafts'),
(('\\HasNoChildren',), '/', 'Filler'),
(('\\HasNoChildren',), '/', 'INBOX'),
(('\\HasNoChildren',), '/', 'Sent'),
--snip--
(('\\HasNoChildren', '\\Flagged'), '/', 'Starred'),
(('\\HasNoChildren', '\\Trash'), '/', 'Trash')]
```

القيم الثلاث في كل مجموعة مثل `(('\\HasNoChildren',), '/', 'INBOX')` هي كما يلي:

- مجموعة من رايات Flags المجلد (لن نوضّح في هذا الفصل ما تمثله هذه الرايات، لذا يمكنك تجاهل هذا الحقل).
- المُحدِّد Delimiter المُستخدَم في سلسلة الاسم النصية لفصل المجلدات الأب عن المجلدات الفرعية.
- الاسم الكامل للمجلد.

يمكنك تحديد مجلدٍ للبحث فيه من خلال تمرير اسم المجلد بوصفه سلسلة نصية إلى التابع `select_folder()` الخاص بالكائن `IMAPClient`كما يلي:

```
>>> imapObj.select_folder('INBOX', readonly=True)
```

يمكنك تجاهل القيمة التي يعيدها التابع `select_folder()`، وإذا كان المجلد المحدَّد غير موجود، فسترفع شيفرة بايثون استثناء `imaplib.error`. يمنعك وسيط الكلمات المفتاحية `readonly=True` من إجراء تغييرات أو حذفٍ عن طريق الخطأ على أيٍّ من رسائل البريد الإلكتروني الموجودة في هذا المجلد أثناء استدعاءات التوابع اللاحقة. إن لم تكن ترغب في حذف رسائل البريد الإلكتروني، فيُفضَّل دائمًا ضبط الوسيط `readonly` على القيمة `True`.

ب. إجراء البحث

حدّدنا المجلد، ويمكننا الآن البحث عن رسائل البريد الإلكتروني باستخدام التابع `search()` الخاص بالكائن `IMAPClient`، حيث يكون وسيط هذا التابع قائمةً من السلاسل النصية، وتكون كل سلسلة نصية بتنسيق مفاتيح بحث IMAP، حيث سنوضّح في الجدول الآتي مفاتيح البحث Search Keys. لاحظ أن بعض خوادم IMAP قد يكون لها طرق تطبيق مختلفة فيما يتعلق بكيفية التعامل مع الرايات ومفاتيح البحث الخاصة بها، لذا قد يتطلب الأمر بعض التجارب في الصدفة التفاعلية لمعرفة كيف تتصرّف بالضبط.

يمكنك تمرير عدة سلاسل نصية لمفاتيح بحث IMAP في وسيط القائمة إلى التابع `search()`، وتكون الرسائل المُعادة هي الرسائل التي تتطابق مع جميع مفاتيح البحث. إذا أردتَ المطابقة مع أيٍّ من مفاتيح البحث، فاستخدم مفتاح البحث `OR`، ولاحظ أن مفتاحَ البحث `NOT` يتبعه مفتاح بحث كامل، وأن مفتاحَ البحث `OR` يتبعه مفتاحا بحث كاملان.

مفتاح البحث	معناه
`'ALL'`	يعيد جميع الرسائل الموجودة في المجلد. قد تواجهك قيود حجم الوحدة `imaplib` إذا طلبت جميع الرسائل الموجودة في مجلد كبير، لذا اطلع على القسم "قيود الحجم" التي سنوضحها لاحقًا.
`'BEFORE date'` و `'ON'` و `'SINCE date'` و `'date'`	تعيد مفاتيح البحث الثلاثة هذه الرسائل التي استلمها خادم IMAP قبل التاريخ `date` المُحدَّد أو فيه أو بعده على التوالي، حيث يجب أن يكون التاريخ بالتنسيق `05-Jul-2023`. يطابق مفتاح البحث `'SINCE 05-Jul-2023'` الرسائل في تاريخ 5 من الشهر السابع وبعده، ولكن يطابق مفتاح البحث `'BEFORE 05-Jul-2023'` الرسائل قبل تاريخ 5 من الشهر السابع فقط دون مطابقة رسائل هذا التاريخ.
`'SUBJECT string'` و `'BODY string'` و `'TEXT string'`	تعيد الرسائل التي تكون فيها السلسلة النصية `string` موجودة في موضوع Subject الرسالة أو نصها Body أو أيٍّ منهما على التوالي. إذا احتوت السلسلة النصية `string` على مسافات، فأحِطها بعلامات اقتباس مزدوجة مثل: `'TEXT "search with spaces"'`.
`'FROM string'` و `'TO string'` و `'CC string'` و `'BCC string'`	تعيد جميع الرسائل التي تكون فيها السلسلة النصية `string` موجودة في عنوان البريد الإلكتروني "من from"، أو عناوين "إلى to"، أو عناوين "cc" (نسخة مطابقة)، أو عناوين "bcc" (نسخة مطابقة مخفية) على التوالي. إذا كانت هناك عناوين بريد إلكتروني متعددة في السلسلة

	النصية string، فافصل بينها بمسافات وأحِط كلها بعلامات اقتباس مزدوجة مثل: 'CC "firstcc@example.com" "secondcc@example.com"'.
'SEEN' و 'UNSEEN'	تعيد جميع الرسائل مع أو بدون الراية \Seen على التوالي، حيث تحصل رسالة البريد الإلكتروني على الراية \Seen إذا وصلنا إليها باستخدام استدعاء التابع fetch() التي سنوضّحها لاحقًا أو إذا نقرنا عليها عند التحقق من البريد الإلكتروني في برنامج بريد إلكتروني أو متصفح ويب. من الشائع أن نقول أن البريد الإلكتروني "مقروء Read" بدلًا من "مُشاهَد Seen"، لكنهما يعنيان الشيء نفسه.
'ANSWERED' و 'UNANSWERED'	تعيد جميع الرسائل مع أو بدون الراية \Answered على التوالي، حيث تحصل الرسالة على الراية \Answered عند الرد عليها.
'DELETED' و 'UNDELETED'	تعيد جميع الرسائل مع أو بدون الراية \Deleted على التوالي. تُعطَى رسائل البريد الإلكتروني المحذوفة باستخدام التابع delete_messages() الرايةَ \Deleted ولكنها لا تُحذَف نهائيًا حتى نستدعي التابع expunge() (اطّلع على القسم "حذف رسائل البريد الإلكتروني" التي سنوضّحها لاحقًا). لاحظ أن بعض مزوّدي خدمة البريد الإلكتروني يحذفون نهائيًا Expunge رسائل البريد الإلكتروني تلقائيًا.
'DRAFT' و 'UNDRAFT'	تعيد جميع الرسائل مع أو بدون الراية \Draft على التوالي. تُحفَظ عادةً رسائل المسودات في مجلد منفصل هو مجلد المسودات Drafts بدلًا من مجلد البريد الوارد INBOX.
'FLAGGED' و 'UNFLAGGED'	تعيد جميع الرسائل مع أو بدون الراية \Flagged على التوالي، حيث تُستخدَم هذه الراية عادةً لوضع علامة على رسائل البريد الإلكتروني بوصفها "مهمة Important" أو "عاجلة Urgent".
'LARGER N' و 'SMALLER N'	تعيد جميع الرسائل الأكبر أو الأصغر من N بايت على التوالي.
'NOT search-key'	يعيد الرسائل التي لا يعيدها مفتاح البحث search-key.
'OR search-key1 search-key2'	يعيد الرسائل التي تطابق مفتاح البحث search-key الأول أو الثاني.

الجدول 25: مفاتيح البحث ومعانيها

إليك فيما يلي بعض الأمثلة على استدعاءات التابع search() مع معانيها:

```
imapObj.search(['ALL'])
```

يعيد جميع الرسائل الموجودة في المجلد المُحدَّد حاليًا.

```
imapObj.search(['ON 05-Jul-2023'])
```

يعيد جميع الرسائل المُرسَلة في 5 من الشهر السادس من عام 2023.

```
imapObj.search(['SINCE 01-Jan-2023', 'BEFORE 01-Feb-2023', 'UNSEEN'])
```

يعيد جميع الرسائل غير المقروءة المُرسَلة في الشهر الأول من عام 2023. لاحظ أن ذلك يعني الرسائل المُرسَلة في 1 من الشهر الأول وما بعده من الشهر الأول ولا يتضمّن 1 من الشهر الثاني.

```
imapObj.search(['SINCE 01-Jan-2023', 'FROM alice@example.com'])
```

يعيد جميع الرسائل المُرسَلة من العنوان `alice@example.com` منذ بداية عام 2023.

```
imapObj.search(['SINCE 01-Jan-2023', 'NOT FROM alice@example.com'])
```

يعيد جميع الرسائل المُرسَلة من الجميع باستثناء العنوان `alice@example.com` منذ بداية عام 2023.

```
imapObj.search(['OR FROM alice@example.com FROM bob@example.com'])
```

يعيد جميع الرسائل المُرسَلة من العنوان `alice@example.com` أو العنوان `bob@example.com`.

```
imapObj.search(['FROM alice@example.com', 'FROM bob@example.com'])
```

لا يعيد هذا البحث أيّ رسائل مطلقًا، لأن الرسائل يجب أن تتطابق مع جميع كلمات البحث المفتاحية. لا يمكن أن يكون هناك سوى عنوان "من from" واحد فقط، فمن المستحيل أن تكون الرسالة من العنوان `alice@example.com` والعنوان `bob@example.com`.

لا يعيد التابع `search()` رسائل البريد الإلكتروني، بل يعيد المعرّفات الفريدة UID لرسائل البريد الإلكتروني بوصفها قيمًا صحيحة. يمكنك بعد ذلك تمرير هذه المعرّفات الفريدة إلى التابع `fetch()` للحصول على محتوى البريد الإلكتروني.

تابع مثال الصدفة التفاعلية من خلال إدخال ما يلي:

```
>>> UIDs = imapObj.search(['SINCE 05-Jul-2023'])
>>> UIDs
[40032, 40033, 40034, 40035, 40036, 40037, 40038, 40039, 40040, 40041]
```

خزّنا قائمة معرّفات الرسائل (للرسائل المُستلمة في 5 من الشهر السابع وما بعده) التي يعيدها التابع `search()` في المتغير `UIDs`. تكون قائمة المعرّفات UIDs المُعادة على حاسوبك مختلفة عن القائمة الموضحة في مثالنا، لأنها فريدة لحساب بريد إلكتروني معين.

استخدم قيم المعرّف الفريد UID التي تلقيتها وليس القيم الواردة في هذا الفصل عندما تمرّرها لاحقًا إلى استدعاءات دوال أخرى.

ج. قيود الحجم

إذا تطـابق بحثـك مـع عـدد كبـير من رسـائل البريـد الإلكـتروني، فقـد ترفـع شـيفرة بـايثون الاسـتثناء `imaplib.error: got more than 10000 bytes`، وعندها يجب قطع الاتصال بخادم IMAP وإعادة الاتصال به والمحاولة مرة أخرى.

وُضِع هذا القيد لمنع برامج بايثون الخاصة بك من استهلاك الكثـير من الـذاكرة، ولكن يكـون الحـد الأقصـى للحجم الافتراضي غالبًا صغيرًا جدًا، إذ يمكنك تغييره من 10000 بايت إلى 10000000 بـايت من خلال تشـغيل الشيفرة البرمجية التالية:

```
>>> import imaplib
>>> imaplib._MAXLINE = 10000000
```

يُفترَض أن تمنع الشيفرة البرمجية السابقة ظهور رسالة الخطأ مـرة أخـرى، لـذا قـد تـرغب في جعـل هـذين السطرين جزءًا من كل برنامج IMAP تكتبه.

18.5.4 جلب بريد إلكتروني ووضع علامة عليه كمقروء

يمكنك استدعاء التابع `fetch()` الخـاص بكـائن `IMAPClient` للحصـول على محتـوى البريـد الإلكـتروني الفعلي بعد حصولك على قائمة المعرّفات الفريدة UID التي ستكون الوسيط الأول لهذا التابع، والوسـيط الثـاني هو القائمة `['BODY[]']` التي تطلب من التـابع `fetch()` تنزيـل كـل المحتـوى الخـاص بنص رسـائل البريـد الإلكتروني المُحدَّدة في قائمة المعرّفات الفريدة UID الخاصة بك.

لنتابع الآن مثالنا على الصدفة التفاعلية:

```
>>> rawMessages = imapObj.fetch(UIDs, ['BODY[]'])
>>> import pprint
>>> pprint.pprint(rawMessages)
{40040: {'BODY[]': 'Delivered-To: my_email_address@example.com\r\n'
        'Received: by 10.76.71.167 with SMTP id '
--snip--
        '\r\n'
        '------=_Part_6000970_707736290.1404819487066--\r\n',
    'SEQ': 5430}}
```

استورد الوحدة `pprint` ومرّر القيمة المُعادة من التابع `fetch()` والمُخزَّنة في المتغير `rawMessages` إلى الدالة `pprint.pprint()` "لطباعتها بمظهر جميل Pretty Print"، وسترى أن هذه القيمة المُعادة هي قاموس متداخل للرسائل ذات المعرفات الفريدة UID بوصفها مفاتيحًا.

تُخزَّن كل رسالة كقاموس له مفتاحان هما: `'BODY[]'` و `'SEQ'`، حيث يُربَط المفتاح `'BODY[]'` مع النص الفعلي للبريد الإلكتروني.

يُعَد المفتاح `'SEQ'` مُخصَّصًا للرقم التسلسلي Sequence Number، والذي له دورٌ مماثل للمعرّف الفريد UID، ولكن يمكنك تجاهله. يُعَد محتوى الرسالة الموجود في المفتاح `'BODY[]'` غير مفهوم إلى حد كبير، فهو بتنسيق اسمه RFC 822، وهو مصمم لتقرأه خوادم IMAP، ولكن لا حاجة إلى فهم هذا التنسيق، إذ سنوضّحه لاحقًا عند شرح وحدة `pyzmail` في هذا الفصل.

استدعينا الدالة `select_folder()` مع وسيط الكلمات المفتاحية `readonly=True` عند تحديد مجلد للبحث فيه، حيث يؤدي ذلك إلى منعك من حذف رسالة بريد إلكتروني عن طريق الخطأ، ولكنه يعني أيضًا عدم وضع علامة على رسائل البريد الإلكتروني بوصفها مقروءة إذا جلبتها باستخدام التابع `fetch()`، لذا إذا أردتَ وضع علامة على رسائل البريد الإلكتروني بوصفها مقروءة عند جلبها، فيجب تمرير الوسيط `readonly=False` إلى الدالة `select_folder()`. إذا كان المجلد المُحدَّد في وضع القراءة فقط، فيمكنك إعادة تحديد المجلد الحالي باستدعاء آخر للدالة `select_folder()` مع وسيط الكلمات المفتاحية `readonly=False` كما يلي:

```
>>> imapObj.select_folder('INBOX', readonly=False)
```

18.5.5 الحصول على عناوين البريد الإلكتروني من رسالة خام Raw Message

لا تُعَد الرسائل الخام التي يعيدها التابع `fetch()` مفيدة جدًا للأشخاص الذين يريدون قراءة رسائل بريدهم الإلكتروني فقط، لذا تحلّل الوحدة `pyzmail` هذه الرسائل الخام وتعيدها بوصفها كائنات `PyzMessage`، مما يجعل أقسام الموضوع والنص والحقل "إلى To" والحقل "من From" والأقسام الأخرى من البريد الإلكتروني قابلة للوصول بسهولة من شيفرة بايثون الخاصة بك.

18.6 تطبيق عملي: إرسال رسائل لتذكير الأعضاء بدفع مستحقاتهم

لنفترض أنك تطوعتَ لتعقّب دفع الأعضاء لمستحقاتهم في نادي تطوع إلزامي، حيث تُعَد هذه المهمة مملةً جدًا، إذ تتضمّن استخدام جدول بيانات يحتوي جميع الأشخاص الذين دفعوا في كلّ شهر وإرسال رسائل تذكير عبر البريد الإلكتروني إلى الأشخاص الذين لم يدفعوا، وبالتالي ستمر على كامل جدول البيانات بنفسك وتنسخ وتلصق رسالة البريد الإلكتروني نفسها لكلّ مَن تأخر في سداد مستحقاته، إذًا لنكتب سكربتًا ينفّذ هذه المهمة نيابةً عنك.

إليك الخطوات العامة التي سيطبّقها برنامجك:

1. قراءة البيانات من جدول بيانات إكسل.
2. البحث عن جميع الأعضاء الذين لم يسدّدوا مستحقاتهم للشهر الأخير.
3. البحث عن عناوين بريدهم الإلكتروني وإرسال رسائل تذكير مُخصَّصة لهم.

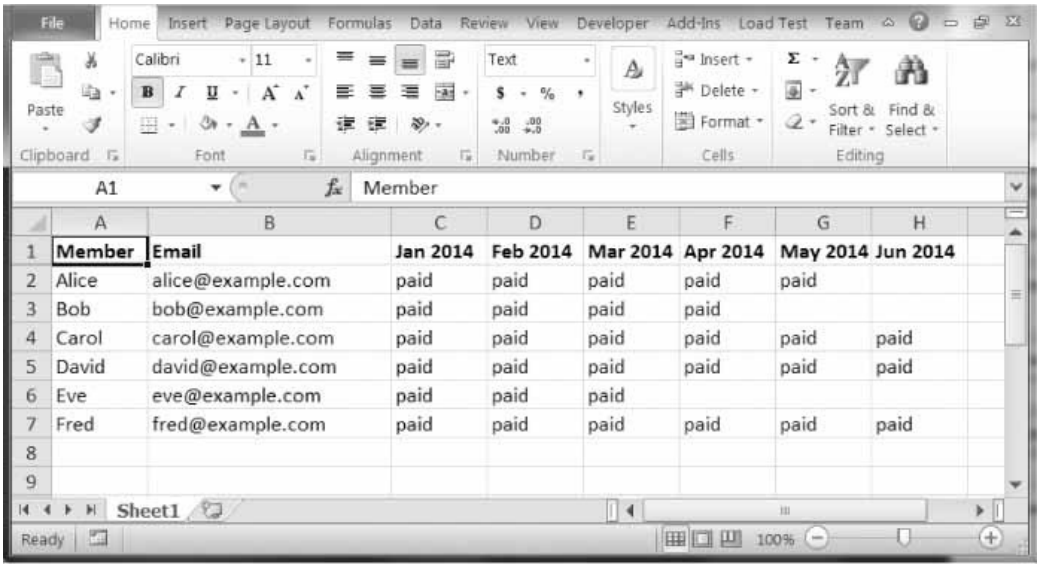
يجب أن تطبّق شيفرتك البرمجية الخطوات التالية:

1. فتح وقراءة خلايا مستند إكسل باستخدام الوحدة `openpyxl` كما تعلّمنا في الفصل الثالث عشر.
2. إنشاء قاموس للأعضاء الذين لم يسددوا مستحقاتهم.
3. تسجيل الدخول إلى خادم SMTP من خلال استدعاء `smtplib.SMTP()` و `ehlo()` و `starttls()` و `login()`.
4. إرسال رسالة تذكير مُخصَّصة عبر البريد الإلكتروني من خلال استدعاء التابع `sendmail()` إلى جميع الأعضاء الذين لم يسددوا مستحقاتهم.

افتح تبويبًا جديدًا في محرّرك لإنشاء ملف جديد واحفظه بالاسم `sendDuesReminders.py`.

18.6.1 الخطوة الأولى: فتح ملف إكسل

لنفترض أن جدول بيانات إكسل الذي تستخدمه لتعقّب دفعات مستحقات العضوية يشبه الشكل التالي، وهو موجود في ملف اسمه `duesRecords.xlsx` ويمكنك تنزيله من nostarch.



	A	B	C	D	E	F	G	H
1	Member	Email	Jan 2014	Feb 2014	Mar 2014	Apr 2014	May 2014	Jun 2014
2	Alice	alice@example.com	paid	paid	paid	paid	paid	
3	Bob	bob@example.com	paid	paid	paid	paid		
4	Carol	carol@example.com	paid	paid	paid	paid	paid	paid
5	David	david@example.com	paid	paid	paid	paid	paid	paid
6	Eve	eve@example.com	paid	paid	paid			
7	Fred	fred@example.com	paid	paid	paid	paid	paid	paid
8								
9								

الشكل 91: جدول تعقّب دفعات مستحقات الأعضاء

يحتوي جدول البيانات على اسم كل عضو وعنوان بريده الإلكتروني، ويكون لكل شهر عمودٌ لتعقّب حالات الدفع الخاصة بالأعضاء، حيث تُميَّز الخلية الخاصة بكل عضو بالكلمة "paid" بعد دفع المستحقات.

يجب أن يفتح البرنامج الملف duesRecords.xlsx ويعرف العمود الخاص بالشهر الأخير من خلال قراءة السمة sheet.max_column. اطّلع على الفصل الخاص بالتعامل مع جداول بيانات إكسل باستخدام بايثون لمزيد من المعلومات حول الوصول إلى الخلايا في ملفات جداول بيانات إكسل باستخدام الوحدة openpyxl.

أدخِل الشيفرة البرمجية التالية في تبويب محرّر ملفاتك:

```
#! python3

# sendDuesReminders.py - إرسال رسائل البريد الإلكتروني بناءً على حالة الدفع في جدول البيانات

import openpyxl, smtplib, sys

# فتح جدول البيانات والحصول على حالة المستحقات الأخيرة

❶ wb = openpyxl.load_workbook('duesRecords.xlsx')

❷ sheet = wb.get_sheet_by_name('Sheet1')

❸ lastCol = sheet.max_column

❹ latestMonth = sheet.cell(row=1, column=lastCol).value

# التحقق من حالة الدفع لكل عضو

# تسجيل الدخول إلى حساب البريد الإلكتروني

# إرسال رسائل التذكير عبر البريد الإلكتروني
```

نستورد الوحدات openpyxl و smtplib و sys، ثم نفتح الملف duesRecords.xlsx ونخزّن كائن Workbook الناتج في المتغير wb ❶.

نحصل بعد ذلك على الورقة Sheet1 ونخزّن الكائن Worksheet الناتج في المتغير sheet ❷. أصبح لدينا كائن Worksheet، وبالتالي يمكننا الآن الوصول إلى الصفوف والأعمدة والخلايا، حيث نخزّن العمود الأعلى في المتغير lastCol ❸، ثم نستخدم الصف رقم 1 والعمود lastCol للوصول إلى الخلية التي يجب أن تحتوي على الشهر الأخير، حيث نحصل على القيمة الموجودة في هذه الخلية ونخزّنها في المتغير latestMonth ❹.

18.6.2 الخطوة الثانية: البحث عن جميع الأعضاء الذين لم يدفعوا مستحقاتهم

حدّدنا رقم العمود للشهر الأخير (المُخزَّن في المتغير lastCol)، ويمكننا الآن المرور ضمن حلقة على جميع الصفوف بعد الصف الأول الذي يحتوي على ترويسات الأعمدة لمعرفة الأعضاء الذين يكون لديهم النص "paid" في الخلية الخاصة بمستحقات ذلك الشهر.

إن لم يدفع العضو، فيمكنك الحصول على اسم العضو وعنوان بريده الإلكتروني من العمودين 1 و2 على التوالي، حيث ستُدخِل هذه المعلومات في القاموس `unpaidMembers` الذي سيتعقّب جميع الأعضاء الذين لم يدفعوا في الشهر الأخير. أدخِل الشيفرة البرمجية التالية إلى برنامج `sendDuesReminder.py`:

```
#! python3
# sendDuesReminders.py - إرسال رسائل البريد الإلكتروني بناءً على حالة الدفع في جدول البيانات
--snip--
# التحقق من حالة الدفع لكل عضو
unpaidMembers = {}
❶ for r in range(2, sheet.max_row + 1):
    ❷ payment = sheet.cell(row=r, column=lastCol).value
    if payment != 'paid':
        ❸ name = sheet.cell(row=r, column=1).value
        ❹ email = sheet.cell(row=r, column=2).value
        ❺ unpaidMembers[name] = email
```

تُعِد الشيفرة البرمجية السابقة القاموس الفارغ `unpaidMembers`، ثم تمر ضمن حلقة على جميع الصفوف بعد الصف الأول ❶، ثم تُخزِّن القيمة الموجودة في العمود الأخير ضمن المتغير `payment` لكل صف❷. إذا لم يساوِ المتغير `payment` القيمة `'paid'`، فستُخزِّن قيمة العمود الأول في المتغير `name` ❸، وتُخزِّن قيمة العمود الثاني في المتغير `email` ❹، ويُضاف المتغيران `name` و `email` إلى القاموس `unpaidMembers` ❺.

18.6.3 الخطوة الثالثة: إرسال رسائل تذكير مخصصة عبر البريد الإلكتروني

حصلنا على قائمة بجميع الأعضاء غير الدافعين، وحان الوقت الآن لإرسال رسائل تذكير لهؤلاء الأعضاء عبر البريد الإلكتروني. أدخِل الشيفرة البرمجية التالية إلى برنامجك، ولكن مع عنوان بريدك الإلكتروني ومعلومات مزوّدك الحقيقية:

```
#! python3
# sendDuesReminders.py - إرسال رسائل البريد الإلكتروني بناءً على حالة الدفع في جدول البيانات
--snip--
```

```
# تسجيل الدخول إلى حساب البريد الإلكتروني
smtpObj = smtplib.SMTP('smtp.example.com', 587)
smtpObj.ehlo()
smtpObj.starttls()
smtpObj.login('my_email_address@example.com', sys.argv[1])
```

أنشئ كائن SMTP من خلال استدعاء الدالة smtplib.SMTP() ومرّر إليها اسم النطاق والمنفذ الخاص بمزوّدك، ثم استدعِ التوابع ehlo() و starttls()، ثم استدعِ التابع login() ومرّر إليه عنوان بريدك الإلكتروني والقائمة sys.argv[1] التي ستخزّن السلسلة النصية لكلمة مرورك، حيث ستدخِل كلمة المرور بوصفها وسيط سطر أوامر في كل مرة تشغّل فيها البرنامج لتجنّب حفظ كلمة مرورك في شيفرتك المصدرية.

يسجّل برنامجك الدخول إلى حساب بريدك الإلكتروني، ثم يجب أن يمر على القاموس unpaidMembers ويرسل بريدًا إلكترونيًا مُخصَّصًا إلى عنوان البريد الإلكتروني لكل عضو. أضِف ما يلي إلى برنامج sendDuesReminders.py:

```
#! python3
# sendDuesReminders.py - إرسال رسائل البريد الإلكتروني بناءً على حالة الدفع في جدول البيانات
--snip--
# إرسال رسائل التذكير عبر البريد الإلكتروني
for name, email in unpaidMembers.items():
    ❶ body = "Subject: %s dues unpaid.\nDear %s,\nRecords show that you
have not
    paid dues for %s. Please make this payment as soon as possible. Thank
you!'" %
    (latestMonth, name, latestMonth)
    ❷ print('Sending email to %s...' % email)
    ❸ sendmailStatus = smtpObj.sendmail('my_email_address@example.com',
email,
body)
    ❹ if sendmailStatus != {}:
        print('There was a problem sending email to %s: %s' % (email,
```

```
      sendmailStatus))

  smtpObj.quit()
```

تمر الشيفرة البرمجية السابقة ضمن حلقة على الأسماء ورسائل البريد الإلكتروني الموجودة في القاموس `unpaidMembers`، ونخصّص رسالةً لكل عضو لم يدفع، حيث تتضمّن هذه الرسالة الشهر الأخير واسم العضو، ونخزّنها في المتغير `body` ❶. نطبع بعد ذلك خرجًا يفيد بأننا نرسل بريدًا إلكترونيًا إلى عنوان البريد الإلكتروني لهذا العضو ❷، ثم نستدعي التابع `sendmail()`، ونمرّر إليه عنوان البريد الإلكتروني والرسالة المُخصَّصة ❸، ونخزّن القيمة المُعادة في المتغير `sendmailStatus`. تذكّر أن التابع `sendmail()` يعيد قيمة قاموس غير فارغ إذا أبلغ خادم SMTP عن خطأ أثناء إرسال هذا البريد الإلكتروني، لذا يتحقق الجزء الأخير من حلقة `for` ❹ مما إذا كان القاموس المُعاد غير فارغ، فإذا كان كذلك، فسيطبع عنوان البريد الإلكتروني للمستلم والقاموس المُعاد. نستدعي التابع `quit()` لقطع الاتصال بخادم SMTP بعد أن ينتهي البرنامج من إرسال كافة رسائل البريد الإلكتروني. ستكون النتيجة كما يلي عند تشغيل البرنامج:

```
Sending email to alice@example.com...
Sending email to bob@example.com...
Sending email to eve@example.com…
```

يتلقّى المستلمون رسالة بريد إلكتروني حول دفعاتهم الفائتة وستشبه البريد الإلكتروني الذي ترسله يدويًا.

18.7 إرسال رسائل نصية عبر بوابات البريد الإلكتروني لخدمة SMS

تكون الهواتف الذكية في متناول أيدينا أكثر من الحواسيب، لذلك تُعَد الرسائل النصية وسيلةً فورية وموثوقية لإرسال الإشعارات أكثر من البريد الإلكتروني، وتكون الرسائل النصية أقصر، ممّا يزيد من احتمالية أن يتمكّن الشخص من قراءتها. الطريقة الأسهل والأكثر موثوقية لإرسال رسائل نصية هي استخدام بوابة البريد الإلكتروني لخدمة الرسائل القصيرة SMS (أو Short Message Service)، وهذه البوابة هي خادم بريد إلكتروني يُعِدّه مزوّد الهاتف المحمول لتلقي الرسائل النصية عبر البريد الإلكتروني بعد ذلك يوجّهها إلى المستلم بوصفها رسالة نصية.

يمكنك كتابة برنامج لإرسال رسائل البريد الإلكتروني باستخدام الوحدتين `ezgmail` أو `smtplib`، حيث يشكّل كلٌّ من رقم الهاتف وخادم البريد الإلكتروني لشركة الهاتف عنوانَ البريد الإلكتروني للمستلم، وسيكون موضوع ونص Body البريد الإلكتروني هو نص الرسالة النصية، فمثلًا يمكنك إرسال رسالة نصية إلى رقم الهاتف 415-555-1234 الذي يملكه عميل شركة Verizon من خلال إرسال بريد إلكتروني إلى العنوان 4155551234@vtext.com.

يمكنك العثور على بوابة البريد الإلكتروني لخدمة الرسائل القصيرة SMS الخاصة بمزوّد الهاتف المحمول من خلال إجراء بحث على الويب عن اسم مزوّد بوابة البريد الإلكتروني لخدمة الرسائل القصيرة، ولكن يوضّح الجدول الآتي هذه البوابات للعديد من مزوّدي الخدمة المشهورين. يمتلك العديد من المزوّدين خوادم بريد إلكتروني منفصلة للرسائل النصية القصيرة، والتي تحدّد أن تحتوي الرسائل على 160 محرفًا، وخوادم أخرى منفصلة لخدمة رسائل الوسائط المتعددة MMS، والتي ليس لها حد أقصى لعدد المحارف. إذا أردتَ إرسال صورة، فيجب استخدام بوابة MMS وإرفاق الملف بالبريد الإلكتروني.

إن لم تعرف مزوّد الهاتف المحمول للمستلم، فيمكنك تجربة استخدام موقعٍ للبحث عن شركة الاتصالات، والذي يجب أن يوفّر شركة الاتصالات الخاصة برقم الهاتف، حيث يمكنك العثور على هذه المواقع من خلال البحث في الويب عن مزود الهاتف المحمول لرقمٍ ما.

ستتيح لك العديد من هذه المواقع البحث عن الأرقام مجانًا بالرغم من أنها ستفرض عليك رسومًا إذا كنت بحاجة إلى البحث عن مئات أو آلاف أرقام الهواتف من خلال واجهة برمجة التطبيقات الخاصة بها

يوضّح الجدول الآتي بوابات البريد الإلكتروني المتاحة لخدمة الرسائل القصيرة الخاصة بمزوّدي خدمات الهاتف المحمول:

مزود الهاتف الخليوي	بوابة SMS	بوابة MMS
AT&T	البوابة number@txt.att.net	البوابة number@mms.att.net
Boost Mobile	البوابة number@sms.myboostmobile.com	بوابة SMS نفسها
Cricket	البوابة number@sms.cricketwireless.net	البوابة number@mms.cricketwireless.net
Google Fi	البوابة number@msg.fi.google.com	بوابة SMS نفسها
Metro PCS	البوابة number@mymetropcs.com	بوابة SMS نفسها
Republic Wireless	البوابة number@text.republicwireless.com	بوابة SMS نفسها
Sprint	البوابة number@messaging.sprintpcs.com	البوابة number@pm.sprint.com
T-Mobile	البوابة number@tmomail.net	بوابة SMS نفسها
U.S. Cellular	البوابة number@email.uscc.net	البوابة number@mms.uscc.net
Verizon	البوابة number@vtext.com	البوابة number@vzwpix.com

Virgin Mobile	البوابة number@vmobl.com	البوابة number@vmpix.com
XFinity Mobile	البوابة number@vtext.com	البوابة number@mypixmessages.com

الجدول 26: بوابات البريد الإلكتروني لخدمة الرسائل القصيرة لمزوّدي خدمات الهاتف المحمول

تُعَد بوابات البريد الإلكتروني لخدمة SMS مجانية وسهلة الاستخدام، ولكن لها بعضُ العيوب الرئيسية وهي:

- لا يوجد أيّ ضمان بأن النص سيصل مباشرةً أو قد لا يصل أبدًا.
- لا توجد طريقة لمعرفة فشل النص في الوصول.
- لا توجد طريقة للرد خاصةٌ بمستلم النص.
- قد تحظرك بوابات SMS إذا أرسلتَ عددًا كبيرًا جدًا من رسائل البريد الإلكتروني، ولا توجد طريقة لمعرفة عدد الرسائل التي ستكون "أكثر من الحد المسموح".
- لا يعني أن بوابة SMS تسلّم رسالة نصية اليوم أنها ستعمل غدًا.
- يُعَد إرسال النصوص عبر بوابة SMS مثاليًا عندما تحتاج إلى إرسال رسالة عابرة غير عاجلة، وإذا كنت بحاجة إلى خدمة أكثر موثوقية، فاستخدم خدمة بوابة SMS التي ليست عبر البريد الإلكتروني كما سنوضّح لاحقًا.

18.8 إرسال رسائل نصية باستخدام خدمة Twilio

ستتعلّم في هذا القسم كيفية التسجيل في خدمة Twilio المجانية واستخدام وحدة بايثون الخاصة بها لإرسال رسائل نصية. خدمة Twilio هي خدمة بوابة SMS، مما يعني أنها تسمح لك بإرسال رسائل نصية من برامجك عبر الإنترنت. يحتوي الحساب التجريبي المجاني لخدمة Twilio على كمية محدودة من الرصيد وستكون النصوص مسبوقة بجملة أن النص مُرسَل من حساب Twilio تجريبي "Sent from a Twilio trial account"، ولكن قد تكون هذه الخدمة التجريبية مناسبة لبرامجك الشخصية. ليست خدمة Twilio خدمة بوابة SMS الوحيدة، فإن لم تفضل استخدام Twilio، فيمكنك العثور على خدمات بديلة من خلال البحث عبر الإنترنت عن "free sms" أو "gateway" أو "python sms api" أو حتى "بدائل twilio".

ثبّت الوحدة `twilio` باستخدام الأمر `pip install --user --upgrade twilio` على نظام ويندوز Windows (أو استخدم الأداة `pip3` على نظامي ماك macOS ولينكس Linux) قبل التسجيل للحصول على حساب Twilio.

> **ملاحظة**: يُعَد هذا القسم خاصًا بالولايات المتحدة الأمريكية، ولكن تقدم Twilio خدمات الرسائل النصية القصيرة لدول أخرى غير الولايات المتحدة، لذا اطّلع على موقع Twilio الرسمي لمزيد من المعلومات، حيث ستعمل وحدة `twilio` ودوالها باستخدام الطريقة نفسها خارج الولايات المتحدة الأمريكية.

18.8.1 التسجيل للحصول على حساب Twilio

انتقل إلى موقع Twilio الرسمي واملأ استمارة التسجيل، ولكن يجب التحقق من رقم الهاتف المحمول الذي تريد إرسال الرسائل النصية إليه بعد التسجيل للحصول على حساب جديد. انتقل إلى صفحة معرّفات المتصل التي جرى التحقق منها Verified Caller IDs وأضِف رقم هاتف يمكنك الوصول إليه، ثم سترسل خدمة Twilio رمزًا إلى هذا الرقم والذي يجب أن تدخله للتحقق منه، حيث يكون هذا التحقق ضروريًا لمنع الأشخاص من استخدام الخدمة لإرسال رسائل نصية غير مرغوب فيها إلى أرقام هواتف عشوائية. ستتمكّن الآن من إرسال رسائل نصية إلى رقم الهاتف باستخدام الوحدة `twilio`.

توفّر خدمة Twilio لحسابك التجريبي رقم هاتف لاستخدامه بوصفه مرسلًا للرسائل النصية، وستحتاج أيضًا معرّف SID ومفتاح الاستيثاق auth token الخاصين بحسابك، إذ يمكنك العثور على هذا المعرّف والمفتاح في صفحة لوحة التحكم Dashboard عندما تسجّل الدخول إلى حسابك على Twilio، حيث تعمل هذه القيم بوصفها اسم مستخدم وكلمة مرور Twilio عند تسجيل الدخول من برنامج بايثون.

18.8.2 إرسال رسائل نصية

ثبّت الوحدة `twilio` وسجّل على حساب Twilio، ثم تحقق من رقم هاتفك وسجّل رقم هاتف Twilio، ثم ستحصل على المعرّف SID ومفتاح الاستيثاق الخاصين بحسابك، وستكون أخيرًا جاهزًا لإرسال رسائل نصية لنفسك من سكربتات بايثون الخاصة بك .

تُعَد شيفرة بايثون الفعلية بسيطةً إلى حدٍ ما بالمقارنة مع جميع خطوات التسجيل. أدخِل ما يلي في الصدفة التفاعلية أثناء اتصال حاسوبك بالإنترنت، مع استبدال قيم المتغيرات `accountSID` و `authToken` و`myTwilioNumber` و `myCellPhone` بمعلوماتك الحقيقية:

```
❶ >>> from twilio.rest import Client
  >>> accountSID = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
  >>> authToken  = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
❷ >>> twilioCli = Client(accountSID, authToken)
  >>> myTwilioNumber = '+14955551234'
```

```
>>> myCellPhone = '+14955558888'

❸ >>> message = twilioCli.messages.create(body='Mr. Watson - Come here
- I want
to see you.', from_=myTwilioNumber, to=myCellPhone)
```

يُفترَض أن تتلقى رسالة نصية بعد لحظات قليلة من كتابة السطر الأخير، وهذه الرسالة النصـية هي: "Sent from your Twilio trial account - Mr. Watson - Come here - I want to see you".

يجب استيراد الوحدة `twilio` باستخدام التعليمـة `from twilio.rest import Client`، وليس باستخدام التعليمة `import twilio` فقط ❶ وفقًا للطريقة التي جرى فيها إعداد هذه الوحدة. خزّن معرّف SID الخاص بحسابك في المتغير `accountSID` وخزّن مفتاح الاستيثاق الخـاص بـك في المتغـير `authToken` ثم استدعِ الدالة `Client()` ومـرّر إليهـا `accountSID` و `authToken`. يعيـد اسـتدعاء الدالـة `Client()` كـائن `Client` ❷، حيث يحتوي هذا الكائن على السمة Attribute التي هي `messages`، والتي بدورها تحتوي على التابع `create()` الذي يمكنك استخدامه لإرسال رسائل نصية، وهو التـابع الـذي يوجّـه خـوادم Twilio لإرسـال رسـالتك النصـية. خـزّن رقم Twilio ورقم هاتفـك المحمـول في المتغـيرين `myTwilioNumber` و`myCellPhone`، ثم استدعِ التابع `create()` ومرّر إليه وسطاء الكلمات المفتاحية Keyword Arguments التي تحدد نص الرسالة النصية ورقم المرسل (`myTwilioNumber`) ورقم المستلم (`myCellPhone`) ❸.

يحتوي الكائن `Message` الذي يعيده التابع `create()` على معلومات حول الرسالة النصية المُرسَـلة. تـابع مثال الصدفة التفاعلية من خلال إدخال ما يلي:

```
>>> message.to
'+14955558888'
>>> message.from_
'+14955551234'
>>> message.body
'Mr. Watson - Come here - I want to see you.'
```

يجب أن تحتـوي السـمات `to` و `from_` و `body` على رقم هاتفـك المحمـول ورقم Twilio والرسـالة على التوالي.

لاحظ أن رقم الهاتف المرسِل موجود في السمة `from_` مـع شـرطة سـفلية في النهايـة وليس `from`، لأن الكلمة `from` هي كلمة مفتاحية في لغة بايثون، إذ لا بد أنك رأيتها مستخدمةً في صيغة تعليمة الاستيراد `from modulename import *` مثلًا، لذلك لا يمكن استخدامها بوصفها اسمًا للسمة.

تابع مثال الصدفة التفاعلية بما يلي:

```
>>> message.status
'queued'
>>> message.date_created
datetime.datetime(2023, 7, 8, 1, 36, 18)
>>> message.date_sent == None
True
```

يجب أن تعطي السمة `status` سلسلة نصية، ويجب أن تعطي السمات `date_created` و`date_sent` كائن `datetime` إذا أُنشِئت وأُرسِلت الرسالة.

قد يبدو غريبًا ضبط السمة `status` على القيمة `'queued'` وضبط السمة `date_sent` على القيمة `None` عندما تتلقى الرسالة النصية مسبقًا، والسبب في ذلك هو أنك التقطتَ الكائن `Message` في المتغير `message` قبل إرسال النص فعليًا.

يجب إعادة جلب الكائن `Message` حتى تتمكّن من رؤية أحدث نسخة من السمتين `status` و `date_sent`. تحتوي كل رسالة من رسائل Twilio على معرّف سلسلة نصية SID فريد يمكن استخدامه لجلب آخر تحديث من الكائن `Message`.

تابع مثال الصدفة التفاعلية من خلال إدخال ما يلي:

```
>>> message.sid
'SM09520de7639ba3af137c6fcb7c5f4b51'
❶ >>> updatedMessage = twilioCli.messages.get(message.sid)
>>> updatedMessage.status
'delivered'
>>> updatedMessage.date_sent
datetime.datetime(2023, 7, 8, 1, 36, 18)
```

يعطي إدخال التعليمة `message.sid` معرّف SID الطويل الخاص بهذه الرسالة، ويمكنك استرداد كائن `Message` جديد مع أحدث المعلومات من خلال تمرير هذا المعرّف SID إلى التابع `get()` الخاص بعميل Twilio ❶، حيث تكون السمات `status` و `date_sent` صحيحة في كائن `Message` الجديد.

تُضبَط السمة `status` على إحدى القيم التالية التي يكون نوعها سلسلة نصية: `'queued'` أو `'sending'` أو `'sent'` أو `'delivered'` أو `'undelivered'` أو `'failed'`.

> **ملاحظة**: يُعَد استلام الرسائل النصية باستخدام خدمة Twilio أكثر تعقيدًا بعض الشيء من إرسالها، إذ تتطلب خدمة Twilio أن يكون لديك موقع ويب يشغّل تطبيقه الويب، ويُعَد ذلك خارج نطاق هذا الفصل.

18.9 تطبيق عملي: وحدة لإرسال رسائل نصية

يُحتمَل أن يكون الشخص الذي سترسل إليه رسائل نصية من برامجك هو أنت، إذ تُعَد الرسائل النصية طريقة رائعة لإرسال إشعارات لنفسك عندما تكون بعيدًا عن حاسوبك. إذا أردتَ أتمتة مهمة مملة باستخدام برنامج يستغرق تشغيله بضع ساعات، فيمكنك جعله يُعلِمكَ برسالةٍ نصية عند الانتهاء، أو قد يكون لديك برنامج مجدول ليعمل خلال فترات زمنية منتظمة ويحتاج إلى الاتصال بك في بعض الأحيان مثل برنامج التحقق من الطقس الذي يرسل إليك رسالة تذكيرية بأن تجلب مظلتك معك.

سنوضّح فيما يلي برنامج بايثون صغير يحتوي على الدالة `textmyself()` التي ترسل رسالة نمرّرها إلى هذه الدالة كوسيط نوعه سلسلة نصية. افتح تبويبًا جديدًا في محرّرك لإنشاء ملف جديد وأدخِل الشيفرة البرمجية التالية، مع وضع معلوماتك الخاصة مكان معرّف SID ومفتاح الاستيثاق الخاصين بالحساب وأرقام الهاتف، واحفظ الملف بالاسم `textMyself.py`.

```
#! python3

# textMyself.py - تعريف الدالة textmyself() التي ترسل رسالة نصية نمرّرها إليها بوصفها سلسلة نصية

# القيم المُحدَّدة مسبقًا:

accountSID = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'

authToken  = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'

myNumber = '+15559998888'

twilioNumber = '+15552225678'

from twilio.rest import Client

❶ def textmyself(message):

  ❷ twilioCli = Client(accountSID, authToken)

  ❸ twilioCli.messages.create(body=message, from_=twilioNumber,
to=myNumber)
```

يخزِّن هذا البرنامج معرّف SID ومفتاح الاستيثاق الخاصين بالحساب والـرقم المرسِـل والـرقم المسـتلِم، ثم يعرّف الدالة `textmyself()` التي تأخذ وسيطًا ❶، وينشئ كائن `Client` ❷، ويستدعي التابع `create()` مع الرسالة التي مرّرتها ❸.

إذا أردتَ إتاحة الدالة `textmyself()` لبرامجك الأخرى، فما عليك سوى وضع الملف `textMyself.py` في المجلد نفسه الذي يحتوي على سكربت بايثون الخاص بك، وإذا أردتَ أن يرسل أحـد برامجـك رسـالة نصـية إليك، فأضِف إليه ما يلي:

```
import textmyself
textmyself.textmyself('The boring task is finished.')
```

يجب التسجيل في خدمة Twilio وكتابة الشيفرة البرمجية الخاصة بإرسال الرسائل النصية مرة واحدة فقط، ثم يمكنك إرسال رسالة نصية من أيٍّ من برامجك الأخرى من خلال كتابة سطرين فقط من الشيفرة البرمجية.

18.10 أسئلة للتدريب

1. ما هو البروتوكول المُستخدَم لإرسال رسائل البريد الإلكـتروني؟ ومـا هـو الـبروتوكول المُسـتخدَم لفحص واستلام رسائل البريد الإلكتروني؟
2. ما هي دوال/توابع الوحدة `smtplib` الأربع التي يجب استدعاؤها لتسجيل الدخول إلى خادم SMTP؟
3. ما هما دالتا/تابعا الوحدة `imapclient` اللتان يجب استدعاؤهما لتسجيل الدخول إلى خادم IMAP؟
4. ما هو نوع الوسيط الذي يمكنك تمريره إلى التابع `imapObj.search()`؟
5. ماذا تفعل إذا تلقّت شيفرتك البرمجية رسالة الخطأ `got more than 10000 bytes`؟
6. تعالج الوحـدة `imapclient` الاتصـال بخـادم IMAP والعثـور على رسـائل البريـد الإلكـتروني، فمـا هي الوحدة التي تعالج قراءة رسائل البريد الإلكتروني التي تجمعها الوحدة `imapclient`؟
7. ما هي ملفات `credentials.json` و `token.json` عند استخدام واجهة برمجة تطبيقـات جيميـل Gmail API؟
8. ما الفرق بين كائنات "سلسلة المحادثات thread" وكائنات "الرسالة" بواجهة برمجة تطبيقات جيميل؟
9. كيف يمكنك العثور على رسـائل البريـد الإلكـتروني الـتي تحتـوي على ملفـات مرفقـة باسـتخدام التـابع `ezgmail.search()`؟
10. ما هي المعلومات الثلاث التي تحتاجها من خدمة Twilio قبل أن تتمكن من إرسال رسائل نصية؟

18.11 مشاريع للتدريب

حاول كتابة البرامج التي تؤدي المهام التي سنوضّحها فيما يلي لكسب خبرة عملية أكبر.

18.11.1 برنامج لإرسال رسائل بريد إلكتروني لإنجاز مهمة روتينية عشوائية

اكتب برنامجًا يأخذ قائمةً بعناوين البريد الإلكتروني لأشخاص وقائمةً بمهام روتينية يجب تنفيذها ويسند المهام الروتينية للأشخاص عشوائيًا، وأرسل بريدًا إلكترونيًا لكل شخص بالمهام الروتينية المُسنَدة إليه.

احتفظ أيضًا بسجل للمهام الروتينية المُسنَدة مسبقًا لكل شخص لتتمكّن من التأكد من أن البرنامج يتجنب إعطاء أيّ شخص المهمة الروتينية نفسها التي أنجزها سابقًا، ويمكنك جدولة البرنامج لتشغيله مرة واحدة في الأسبوع تلقائيًا.

إذا مرّرتَ قائمةً إلى الدالة `random.choice()`، فستعيد عنصرًا مُحدَّدًا عشوائيًا من القائمة. يمكن أن يبدو جزء من شيفرتك البرمجية كما يلي:

```
chores = ['dishes', 'bathroom', 'vacuum', 'walk dog']
randomChore = random.choice(chores)
chores.remove(randomChore)     # أُنجِزت هذه المهمة الروتينية، لذا يجب إزالتها
```

18.11.2 برنامج للتذكير بإحضار المظلة

وضّحنا في الفصل الثاني عشر كيفية استخدام الوحدة `requests` لاستخراج البيانات من موقع الطقس، لذا اكتب برنامجًا يعمل قبل أن تستيقظ في الصباح مباشرةً ويتحقق مما إذا كانت السماء تمطر في ذلك اليوم. إذا كانت ستمطر، فاطلب من البرنامج أن يرسل لك رسالة تذكيرية بضرورة إحضار مظلة قبل مغادرة المنزل.

18.11.3 برنامج لإلغاء الاشتراك التلقائي

اكتب برنامجًا يبحث في حساب بريدك الإلكتروني ليجد جميع روابط إلغاء الاشتراك في جميع رسائل بريدك الإلكتروني، ويفتحها في المتصفح تلقائيًا.

يجب على هذا البرنامج أن يسجّل الدخول إلى خادم IMAP الخاص بمزوّد بريدك الإلكتروني، وينزّل جميع رسائل بريدك الإلكتروني، ويمكنك استخدام المكتبة Beautiful Soup التي وضّحناها في الفصل الثاني عشر للتحقق من النسخة التي فيها كلمة إلغاء الاشتراك Unsubscribe ضمن الوسم link في شيفرة HTML.

يمكنك استخدام الدالة `webbrowser.open()` لفتح جميع هذه الروابط لعناوين URL تلقائيًا في المتصفح بعد الحصول على قائمة بهذه العناوين، ثم يجب المرور على الخطوات الإضافية وإكمالها يدويًا لإلغاء الاشتراك بهذه القوائم، حيث يتضمن ذلك النقر على الرابط للتأكيد في معظم الحالات.

يوفّر هذا السكربت عليك الاضطرار إلى المرور على جميع رسائل بريدك الإلكتروني بحثًا عن روابط إلغاء الاشتراك، ويمكنك بعد ذلك إعطاء هذا السكربت إلى أصدقائك حتى يتمكنوا من تشغيله على حسابات بريدهم الإلكتروني، ولكن تأكّد من أن كلمة مرور بريدك الإلكتروني غير مكتوبة في شيفرتك المصدرية.

18.11.4 التحكم في حاسوبك من خلال البريد الإلكتروني

اكتب برنامجًا يتحقق من حساب البريد الإلكتروني كل 15 دقيقة بحثًا عن أيّ تعليمات ترسلها إليه عبر البريد الإلكتروني وينفّذ تلك التعليمات تلقائيًا.

يُعَد BitTorrent مثلًا نظام تنزيل يستخدم تقنية الند للند peer-to-peer، حيث يمكنك تنزيل ملفات الوسائط الكبيرة على حاسوبك المنزلي باستخدام برنامج BitTorrent مجاني مثل البرنامج qBittorrent. إذا أرسلتَ رابط BitTorrent (رابطًا قانونيًا وليس رابط قرصنة) إلى البرنامج عبر البريد الإلكتروني، فسيتحقق البرنامج من بريده الإلكتروني ويعثر على هذه الرسالة ويستخرج الرابط، ثم يشغّل برنامج qBittorrent لبدء تنزيل الملف.

يمكنك بهذه الطريقة جعل حاسوبك المنزلي يبدأ التنزيلات أثناء تواجدك بعيدًا عن المنزل، ويمكن الانتهاء من التنزيل (القانوني وغير المقرصن) بحلول وقت عودتك إلى المنزل.يوضّح الفصل السابع عشر كيفية تشغيل البرامج على حاسوبك باستخدام الدالة `subprocess.Popen()`،

فمثلًا سيؤدي الاستدعاء التالي إلى تشغيل برنامج qBittorrent مع ملف تورنت:

```
qbProcess = subprocess.Popen(['C:\\Program Files (x86)\\qBittorrent\\
qbittorrent.exe', 'shakespeare_complete_works.torrent'])
```

يجب أن يتأكد البرنامج من أن رسائل البريد الإلكتروني تأتي منك، إذ قد ترغب في اشتراط أن تحتوي رسائل البريد الإلكتروني على كلمة مرور، لأنه من السهل إلى حدٍ ما أن يزيّف المخترقون عنوان "من from" في رسائل البريد الإلكتروني.

يجب أن يحذف البرنامج رسائل البريد الإلكتروني التي يجدها حتى لا يكرر التعليمات في كل مرة يتحقق فيها من حساب البريد الإلكتروني، واجعل البرنامج أيضًا يرسل لك بريدًا إلكترونيًا أو رسالة تأكيد في كل مرة ينفّذ فيها أمرًا. من الجيد استخدام دوال التسجيل الموضّحة في الفصل الحادي عشر لكتابة سجل ملف نصي يمكنك التحقق منه في حالة ظهور أخطاء، نظرًا لأنك لن تجلس أمام الحاسوب الذي يشغّل البرنامج.

يتمتع برنامج qBittorrent وتطبيقات BitTorrent الأخرى بميزةٍ تمكّنه من الإغلاق تلقائيًا بعد اكتمال التنزيل، حيث وضّحنا في الفصل السابق كيف يمكنك تحديد موعد إنهاء التطبيق المُشغَّل باستخدام التابع `wait()` لكائنات `Popen`. سيوقِف استدعاء التابع `wait()` التنفيذ حتى يتوقف البرنامج qBittorrent، ثم يمكن لبرنامجك إرسال بريد إلكتروني أو رسالة نصية إليك لإعلامك باكتمال التنزيل.

هناك الكثير من الميزات المحتملة الـتي يمكنـك إضـافتها إلى هـذا المشـروع، ولكن إذا واجهتـك مشـكلة، فيمكنك تنزيل مثال تطبيق هذا البرنامج من nostarch.

18.12 الخلاصة

نتواصل مع بعضنا بعضًا عبر الإنترنت وعبر شبكات الهاتف المحمـول باسـتخدام العديـد من الطـرق، ولكن يُعَد البريد الإلكتروني والرسائل النصية الطرق الأكثر انتشارًا، حيث يمكن لبرامجك التواصـل عـبر هـذه القنـوات، مما يمنحها ميزات إشعارات جديدة. يمكنك أيضًا كتابة برامج تعمل على حواسيب مختلفة وتتواصل مع بعضـها بعضًا مباشرةً عبر البريد الإلكتروني، حيث يرسل أحد البرامج رسائل البريد الإلكتروني باستخدام بروتوكول SMTP بينما يستردها البرنامج الآخر باستخدام بروتوكول IMAP.

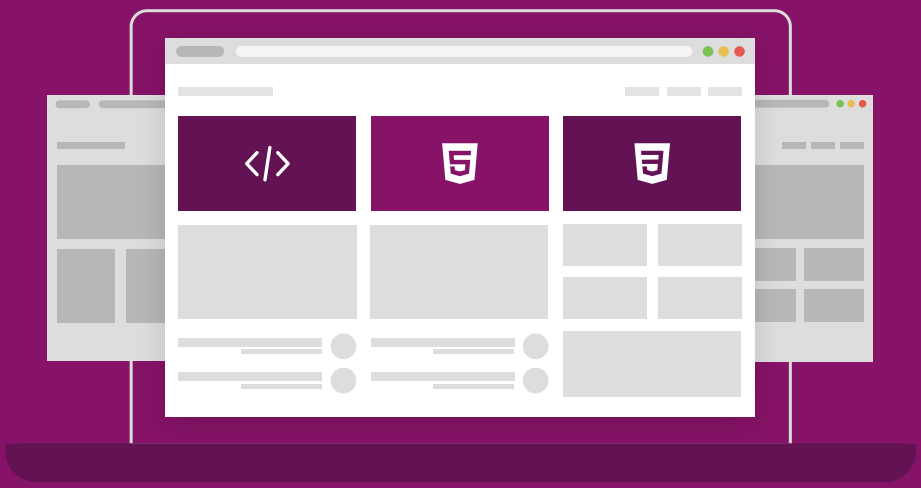
توفّر وحدة `smtplib` الخاصة بلغة بايثون دوالًا لاستخدام بروتوكول SMTP لإرسال رسائل البريد الإلكتروني عبر خادم SMTP الخـاص بمـزوّد بريـدك الإلكـتروني، وتـتيح الوحـدتان `imapclient` و `pyzmail` الخارجيتـان الوصولَ إلى خوادم IMAP واسترداد رسائل البريد الإلكتروني المرسلة إليك. يُعَد بروتوكول IMAP أكثر تعقيدًا من بروتوكول SMTP، ولكنه قوي جدًا ويسمح بالبحث عن رسائل بريد إلكتروني معينة وتنزيلهـا وتحليلهـا لاسـتخراج موضوع ونص البريد الإلكتروني بوصفها قيمًا نوعها سلسلة نصية.

لا تسمح بعض خدمات البريد الإلكتروني الشائعة مثل جيميـل بـأن تسـتخدم بروتوكـولات SMTP و IMAP المعيارية للوصول إلى خدماتها كإجراء احترازي من الرسائل غير المرغوب بها وبهدف الأمـان، ولكن تعمـل وحـدة EZGmail بمثابة مغلِّف مناسب لواجهة برمجة تطبيقات جيميل، ممـا يسـمح لسـكربتات بـايثون الخاصـة بـك بالوصول إلى حسابك على جيميل. نوصي بشدة بإعداد حساب جيميـل منفصـل لاسـتخدام سـكربتاتك حـتى لا تتسبب الأخطاء المحتملة في برنامجك في حدوث مشكلات لحسابك الشخصي على جيميل.

تختلف الرسائل النصية عن البريد الإلكـتروني بعض الشـيء، لأنـه هنـاك حاجـة إلى أكـثر من مجـرد اتصـال بالإنترنت لإرسال رسائل نصية قصيرة SMS على عكس البريد الإلكتروني، حيث توفر خدمات مثل خدمة Twilio وحداتٍ تسمح بإرسال رسائل نصية من برامجك. ستتمكّن بعد إجراء عملية الإعـداد الأوليـة من إرسـال الرسـائل النصية باستخدام سطرين فقط من الشيفرة البرمجية.

ستتمكّن باستخدام هذه الوحدات إلى جانب مهاراتك الأخـرى من برمجـة الشـروط المُحـدَّدة الـتي بموجبهـا يجب على برامجك إرسال الإشعارات أو التذكيرات، وبالتالي ستصل برامجك الآن إلى ما هو أبعـد من الحاسـوب الذي تعمل عليه.

دورة تطوير واجهات المستخدم



مميزات الدورة

- بناء معرض أعمال قوي بمشاريع حقيقية
- وصول مدى الحياة لمحتويات الدورة
- تحديثات مستمرة على الدورة مجانًا
- شهادة معتمدة من أكاديمية حسوب
- إرشادات من المدربين على مدار الساعة
- من الصفر دون الحاجة لخبرة مسبقة

اشترك الآن

19. معالجة الصور باستخدام لغة بايثون

ستصادف ملفات الصور الرقمية طوال الوقت إذا كان لديك كاميرا رقمية أو حتى إذا رفعتَ صورًا من هاتفك على حسابك على فيسبوك أو انستغرام مثلًا، وقد تعرف كيفية استخدام برامج الرسوميات الأساسية مثل الرسام Microsoft Paint أو Paintbrush، أو حتى التطبيقات الأكثر تقدمًا مثل أدوبي فوتوشوب Adobe Photoshop، ولكن إذا كنت بحاجة إلى تعديل عددٍ كبير من الصور، فيمكن أن يكون إنجاز ذلك يدويًا مهمة طويلة ومملة.

تُعَد Pillow وحدةَ بايثون خارجية للتفاعل مع ملفات الصور، وتحتوي هذه الوحدة على العديد من الدوال التي تسهّل قص محتوى الصورة وتغيير حجمه وتعديله. يمكن لبايثون Python تعديل مئات أو آلاف الصور آليًا بسهولة بفضل القدرة على معالجة الصور باستخدام الطريقة نفسها التي تستخدمها برامجٌ مثل برنامج الرسام أو الفوتوشوب. يمكنك تثبيت وحدة `Pillow` من خلال تشغيل الأمر `pip install --user -U pillow==9.2.0`.

19.1 أساسيات الصور الحاسوبية

يجب أن تفهم أساسيات كيفية تعامل الحواسيب مع الألوان والإحداثيات في الصور وكيفية العمل مع الألوان والإحداثيات في الوحدة `Pillow` من أجل معالجة الصور، لذا ثبّت هذه الوحدة قبل المتابعة.

19.1.1 الألوان وقيم RGBA

تمثّل البرامج الحاسوبية لونًا في صورة بوصفه قيمة RGBA، والتي هي مجموعة من الأعداد التي تحدّد مقدار اللون الأحمر والأخضر والأزرق وقيمة ألفا Alpha (أو الشفافية Transparency) في اللون.

كل قيمة من قيم هذه المكونات هي عددٌ صحيح قيمته من 0 (عدم وجود لون على الإطلاق) إلى 255 (الحد الأقصى). تُسنَد قيم RGBA إلى البكسلات، والبكسل Pixel هو أصغر نقطة من لون واحد يمكن أن تظهرها شاشة الحاسوب، فهناك ملايين البكسلات على الشاشة، ويعبّر إعداد RGB الخاص بالبكسل عن درجة اللون التي يجب أن يعرضها بدقة. تحتوي الصور أيضًا على قيمة ألفا لإنشاء قيم RGBA، حيث إذا عُرِضت صورة على الشاشة فوق صورة خلفية أو خلفية سطح مكتب، فإن قيمة ألفا تحدِّد مقدار الخلفية التي يمكنك رؤيتها عبر بكسل الصورة.

تُمثَّل قيم RGBA في الوحدة Pillow باستخدام مجموعة Tuple مكونة من أربع قيم صحيحة، فمثلًا يُمثَّل اللون الأحمر باستخدام المجموعة `(255, 0, 0, 255)`، حيث يحتوي هذا اللون على الحد الأقصى من اللون الأحمر، ولا يحتوي على اللون الأخضر أو الأزرق، ويحتوي على الحد الأقصى من قيمة ألفا، مما يعني أنه مُعتَم Opaque تمامًا. يُمثَّل اللون الأخضر باستخدام المجموعة `(0, 255, 0, 255)`، ويُمثَّل اللون الأزرق باستخدام المجموعة `(0, 0, 255, 255)`، ويُمثَّل اللون الأبيض الذي هو مزيج من كل الألوان باستخدام المجموعة `(255, 255, 255, 255)`، واللون الأسود الذي لا لون له هو `(0, 0, 0, 255)`.

إذا كانت قيمة الشفافية للون هي 0، فسيكون اللون غير مرئي، وتصبح قيم RGB غير مهمة، وبالتالي سيبدو اللون الأحمر غير المرئي مثل اللون الأسود غير المرئي.

تستخدم الوحدة Pillow أسماء الألوان المعيارية التي تستخدمها لغة HTML، حيث يوضح الجدول الآتي مجموعة مختارة من أسماء الألوان المعيارية وقيم RGBA الخاصة بها:

اسم اللون	قيمة RGBA الخاصة به
White (الأبيض)	(255, 255, 255, 255)
Green (الأخضر)	(0, 128, 0, 255)
Gray (الرمادي)	(128, 128, 128, 255)
Black (الأسود)	(0, 0, 0, 255)
Red (الأحمر)	(255, 0, 0, 255)
Blue (الأزرق)	(0, 0, 255, 255)
Yellow (الأصفر)	(255, 255, 0, 255)
Purple (البنفسجي)	(128, 0, 128, 255)

الجدول 27: أسماء الألوان المعيارية وقيم RGBA الخاصة بها

توفّر الوحدة `Pillow` الدالة `ImageColor.getcolor()`، وبالتالي لن تكون مضطرًا إلى حفظ قيم RGBA للألوان التي تريد استخدامها، وتأخذ هذه الدالة سلسلة نصية تمثّل اسم اللون كوسيطٍ أول لها والسلسلة النصية `'RGBA'` كوسيطٍ ثانٍ لها، وتعيد مجموعة RGBA.

أدخِل ما يلي في الصدفة التفاعلية Interactive Shell لمعرفة كيفية عمل هذه الدالة:

```
❶ >>> from PIL import ImageColor
❷ >>> ImageColor.getcolor('red', 'RGBA')
 (255, 0, 0, 255)
❸ >>> ImageColor.getcolor('RED', 'RGBA')
 (255, 0, 0, 255)
 >>> ImageColor.getcolor('Black', 'RGBA')
 (0, 0, 0, 255)
 >>> ImageColor.getcolor('chocolate', 'RGBA')
 (210, 105, 30, 255)
 >>> ImageColor.getcolor('CornflowerBlue', 'RGBA')
 (100, 149, 237, 255)
```

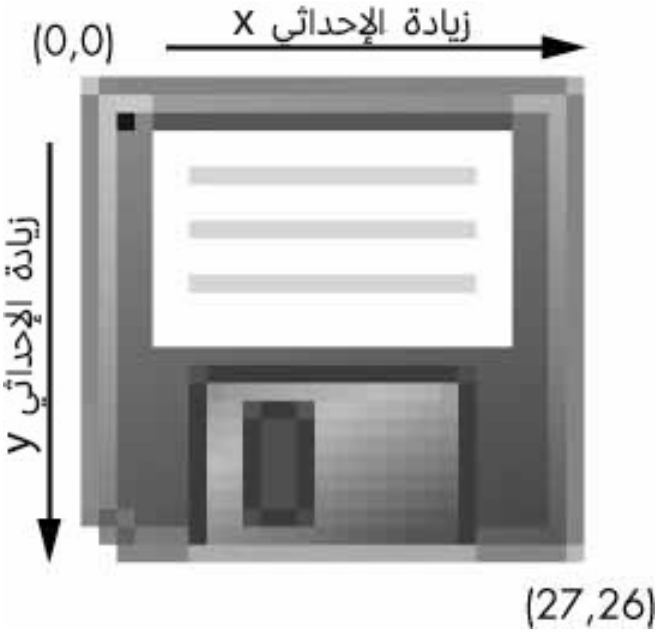
يجب أولًا استيراد الوحدة `ImageColor` من PIL ❶ (وليس من `Pillow`). السلسلة النصية لاسم اللون التي تمررها إلى الدالة `ImageColor.getcolor()` غير حساسة لحالة الأحرف، لذا يعطي تمرير السلسلة النصية 'red' ❷ وتمرير السلسلة النصية 'RED' ❸ مجموعة RGBA نفسها، ويمكنك أيضًا تمرير أسماء ألوان غير اعتيادية مثل `'chocolate'` و `'Cornflower Blue'`.

تدعم الوحدة Pillow عددًا كبيرًا من أسماء الألوان من `'aliceblue'` إلى `'whitesmoke'`، حيث يمكنك العثور على القائمة الكاملة لأكثر من 100 اسم لون معياري في الموارد الموجودة على nostarch.

19.1.2 الإحداثيات والمجموعات المربعة

تُعنوَن بكسلات الصورة باستخدام إحداثيات x و y، والتي تُحدِّد موقع البكسل الأفقي والعمودي على التوالي في الصورة، وتكون نقطة الأصل Origin (أو مبدأ الإحداثيات) هي البكسل الموجود في الزاوية العلوية اليسرى من الصورة ونحدّدها بالصيغة (0, 0)، حيث يمثل الصفر الأول الإحداثي x الذي يبدأ من الصفر عند نقطة الأصل وتزداد قيمته من اليسار إلى اليمين، ويمثل الصفر الثاني الإحداثي y الذي يبدأ من الصفر عند نقطة الأصل وتزداد قيمته نزولًا إلى أسفل الصورة.

تزداد إحداثيات y باتجاه الأسفل، إذ يُعَد ذلك عكس الطريقة التي استخدمناها سابقًا لإحداثيات y في حصص الرياضيات في المدرسة. يوضح الشكل التالي كيفية عمل هذا النظام من الإحداثيات:

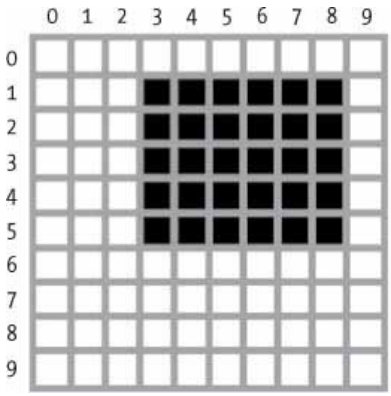


الشكل 92: إحداثيات x و y لصورة أبعادها 27×28 لأحد أنواع أجهزة تخزين البيانات القديمة

تأخذ العديد من دوال وتوابع الوحدة `Pillow` وسيطًا نوعه مجموعة مربعة Box Tuple، وهذا يعني أن الوحدة `Pillow` تتوقع مجموعةً مؤلفة من أربعة إحداثيات صحيحة تمثل منطقةً مستطيلة في الصورة، والأعداد الصحيحة الأربعة هي بالترتيب كما يلي:

- Left: الإحداثي x للحافة اليسرى من المربع.
- Top: الإحداثي y للحافة العلوية من المربع.
- Right: الإحداثي x لبكسل واحد على يمين الحافة اليمنى القصوى للمربع، ويجب أن يكون هذا العدد الصحيح أكبر من العدد الصحيح الأيسر Left.
- Bottom: الإحداثي y لبكسل واحد تحت الحافة السفلية للمربع، ويجب أن يكون هذا العدد الصحيح أكبر من العدد الصحيح العلوي Top.

لاحظ أن المربع يتضمن الإحداثيات اليسرى والعليا حتى الوصول إلى الإحداثيات اليمنى والسفلى ولكنه لا يتضمنها، فمثلًا تمثل المجموعة المربعة `(3, 1, 9, 6)` جميع البكسلات الموجودة في المربع الأسود في الشكل التالي:



الشكل 93: المنطقة التي تمثّلها

المجموعة المربعة (3, 1, 9, 6)

19.2 معالجة الصور باستخدام الوحدة Pillow

عرفنا كيفية عمل الألوان والإحداثيات في الوحدة `Pillow`، وسنستخدمها الآن لمعالجة الصـور. سنسـتخدم الصورة التالية لجميع أمثلة الصدفة التفاعلية في هذا الفصل:

الشكل 94: القطة زوفي Zophie

ضع ملف الصورة `zophie.png` في مجلد العمل الحالي، ثم ستكون جاهزًا لتحميل صورة هـذه القطـة إلى شيفرة بايثون كما يلي:

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
```

يمكنك تحميل الصورة من خلال استيراد الوحدة `Image` من الوحدة `Pillow` واستدعاء الدالة `Image.open()`، ثم تمرّر اسم ملف الصورة إلى هذه الدالة، ويمكنك بعد ذلك تخزين الصورة المُحمَّلة في المتغير `CatIm`. اسم الوحدة `Pillow` هو `PIL` لجعلها متوافقة مع الإصدارات السابقة من وحدةٍ أقدم اسمها Python Imaging Library، ولذلك يجب تشغيل التعليمة `from PIL import Image` بدلًا من التعليمة `from Pillow import Image`، ويجب استخدام تعليمة الاستيراد `from PIL import Image` بدلًا من التعليمة `import PIL` وفقًا للطريقة التي أعدّ بها منشئو `Pillow` هذه الوحدة.

إن لم يكن ملف الصورة موجودًا في مجلد العمل الحالي، فغيّر مجلد العمل إلى المجلد الذي يحتوي على ملف الصورة من خلال استدعاء الدالة `os.chdir()`:

```
>>> import os
>>> os.chdir('C:\\folder_with_image_file')
```

تعيد الدالة `Image.open()` قيمة من نوع البيانات كائن `Image`، وهي الطريقة التي تمثّل بها الوحدة Pillow الصورة بوصفها قيمة بايثون. يمكنك تحميل كائن `Image` من ملف صورة (بأيّ صيغة) من خلال تمرير السلسلة النصية التي تمثل اسم الملف إلى الدالة `Image.open()`، ويمكن حفظ أيّ تغييرات تجريها على كائن `Image` في ملف صورة (بأيّ صيغة أيضًا) باستخدام التابع `save()`.

تُجرَى جميع عمليات التدوير وتغيير الحجم والقص والرسم وغيرها من عمليات معالجة الصور من خلال استدعاءات التوابع الموافقة لهذه العمليات مع كائن `Image`.

سنفترض أنك استوردتَ الوحدة `Image` الخاصة بالوحدة `Pillow` وأن لديك صورة القطة Zophie مُخزَّنة في المتغير `catIm` لاختصار الأمثلة في هذا الفصل.

تأكّد من وجود الملف `zophie.png` في مجلد العمل الحالي حتى تتمكّن الدالة `Image.open()` من العثور عليه، وإلّا فيجب تحديد المسار المطلق الكامل في وسيط السلسلة النصية للدالة `Image.open()`.

19.2.1 العمل مع نوع البيانات Image

يحتوي الكائن `Image` على العديد من السمات Attributes المفيدة التي توفر لك معلومات أساسية حول ملف الصورة الذي جرى تحميله منه مثل: عرضه وارتفاعه، واسم الملف، وصيغة الرسوميات (مثل JPEG أو GIF أو PNG).أدخل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
>>> catIm.size
```

```
❶ (816, 1088)

❷ >>> width, height = catIm.size

❸ >>> width
   816
❹ >>> height
   1088
 >>> catIm.filename
 'zophie.png'
 >>> catIm.format
 'PNG'
 >>> catIm.format_description
 'Portable network graphics'
❺ >>> catIm.save('zophie.jpg')
```

ننشئ كائن `Image` من ملف الصورة `zophie.png` ونخزّن هذا الكائن في المتغير `catIm`، ثم يمكننا أن نرى أن سمة الحجم `size` الخاصة بهذا الكائن تحتوي على مجموعةٍ Tuple تتألف من عرض الصورة وارتفاعها بالبكسل ❶. يمكننا إسناد القيم الموجودة في هذه المجموعة إلى المتغيرين `width` و `height` ❷ للوصول إلى العرض ❸ والارتفاع ❹ لوحدهما. تمثّل السمة `filename` اسم الملف الأصلي، وتمثّل السمات `format` و`format_description` -التي هي سلاسل نصية- صيغة الصورة للملف الأصلي، مع كون السمة `format_description` أكثر تفصيلًا.

أخيرًا، يؤدي استدعاء التابع `save()` وتمرير `'zophie.jpg'` إلى هذا التابع إلى حفظ صورة جديدة بالاسم `zophie.jpg` على قرص حاسوبك الصلب ❺. ترى الوحدة `Pillow` أن امتداد الملف هو `.jpg` وتحفظ الصورة تلقائيًا بصيغة الصورة JPEG. يُفترَض أن يكون لديك الآن صورتان هما `zophie.png` و `zophie.jpg` على قرص حاسوبك الصلب، حيث يعتمد هذان الملفان على الصورة نفسها، ولكنهما غير متطابقين بسبب اختلاف صيغتيهما.

توفر الوحدة `Pillow` أيضًا الدالة `Image.new()` التي تعيد كائن `Image`، حيث تشبه هذه الدالة إلى حدٍ كبيرالدالة `Image.open()`، باستثناء أن الصورة التي يمثلها كائن الدالة `Image.new()` فارغة. وسطاء الدالة `Image.new()` هي كما يلي:

- السلسلة النصية `'RGBA'` التي تضبط نمط الألوان على القيمة RGBA، إذ توجد أنماط أخرى لن نوضّحها في هذا الفصل.
- حجم الصورة الذي نمثّله بمجموعة مكونة من عددين صحيحين لعرض الصورة الجديدة وارتفاعها.
- لون الخلفية الذي يجب أن تبدأ به الصورة، ونمثّله بمجموعة مكونة من أربعة أعداد صحيحة لقيمة RGBA، حيث يمكنك استخدام القيمة التي تعيدها الدالة `ImageColor.getcolor()` لهذا الوسيط، لكن تدعم الدالة `Image.new()` تمرير سلسلة نصية تمثّل اسم اللون المعياري فقط.

أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> from PIL import Image
❶ >>> im = Image.new('RGBA', (100, 200), 'purple')
>>> im.save('purpleImage.png')
❷ >>> im2 = Image.new('RGBA', (20, 20))
>>> im2.save('transparentImage.png')
```

ننشئ كائن `Image` لصورة عرضها 100 بكسل وطولها 200 بكسل مع خلفية بنفسجية ❶، ثم نحفظ هذه الصورة في الملف `purpleImage.png`.

نستدعي بعد ذلك الدالة `Image.new()` مرةً أخرى لإنشاء كائن `Image` آخر مع تمرير المجموعة `(20, 20)` التي تمثّل الأبعاد دون تمرير شيء للون الخلفية ❷.

يُعَد اللون الأسود غير المرئي `(0, 0, 0, 0)` هو اللون الافتراضي المُستخدَم عند عدم تحديد وسيط اللون، وبالتالي فإن الصورة الثانية لها خلفية شفافة، ثم نحفظ هذا المربع الشفاف الذي أبعاده 20×20 في الملف `transparentImage.png`.

19.2.2 قص الصور

يمثّل قص الصورة تحديد منطقة مستطيلة من الصورة وإزالة كل شيء خارج هذا المستطيل. يأخذ التابع `crop()` مع كائنات `Image` مجموعة مربعة ويعيد كائن `Image` يمثل الصورة التي قصّها.

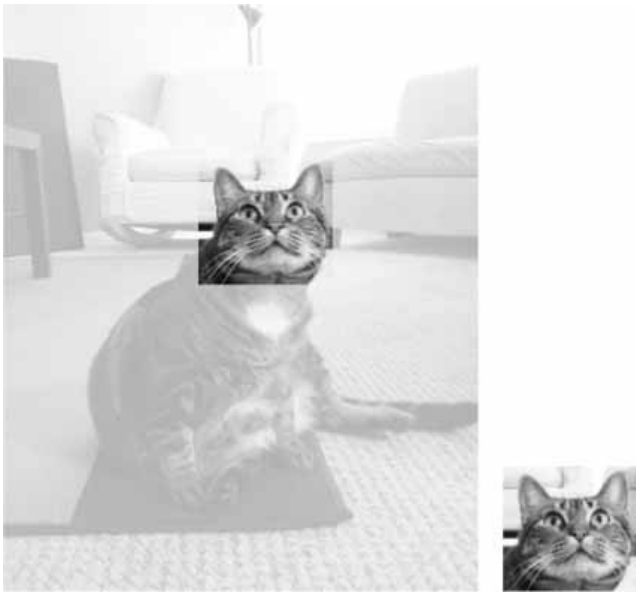
يترك التابع `crop()` كائن `Image` الأصلي دون تغيير بعد القص، ويعيد كائن `Image` جديد.

تذكّر أن المجموعة المربعة (أي الجزء المقصوص في هذه الحالة) يتضمّن العمود الأيسر والصف العلوي من البكسلات وحتى الوصول إلى العمود الأيمن والصف السفلي من البكسلات دون تضمينها.

أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
>>> croppedIm = catIm.crop((335, 345, 565, 560))
>>> croppedIm.save('cropped.png')
```

ننشئ كائن `Image` جديد للصورة المقصوصة، ونخزّن هـذا الكـائن في المتغـير `croppedIm`، ثم نسـتدعي التابع `save()` مع `croppedIm` لحفظ الصورة المقصوصة في الملف `cropped.png`. سينشَأ الملف الجديـد `cropped.png` من الصورة الأصلية كما في الشكل التالي:



الشكل 95: تكون الصورة الجديدة هي الجزء المقصوص من الصورة الأصلية

19.2.3 نسخ ولصق الصور في صور أخرى

يعيد التابع `copy()` كائن `Image` جديد يحتوي على الصورة نفسها للكـائن `Image` الـذي اسـتدعيناه معـه، ويُعَد ذلك مفيدًا إذا كنت بحاجة إلى إجراء تغييرات على الصورة ولكنك تريد الاحتفاظ بنسـخة دون تغيـيرات من النسخة الأصلية،

فمثلًا أدخِل ما يلي في الصدفة التفاعلية:

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
>>> catCopyIm = catIm.copy()
```

يحتوي المتغيران `catIm` و `catCopyIm` على كائني `Image` منفصلين، وكلاهما لهما الصورة نفسهما. أصبح لديك الآن كائن `Image` مُخزَّن في المتغير `catCopyIm`، ويمكنك الآن تعديل المتغير `catCopyIm` كما تريد وحفظه باسم ملف جديد، مع ترك الملف `zophie.png` دون تغيير.

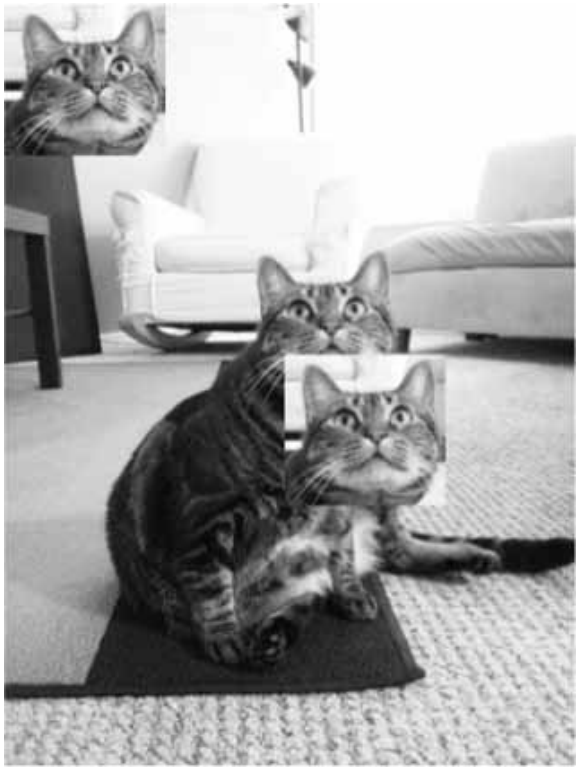
لنحاول مثلًا تعديل المتغير `catCopyIm` باستخدام التابع `paste()`، حيث يُستدعَى هذا التابع مع كائن `Image` ويلصق صورة أخرى فوقه. إذًا لنتابع مثال الصدفة من خلال لصق صورة أصغر على `catCopyIm` كما يلي:

```
>>> faceIm = catIm.crop((335, 345, 565, 560))
>>> faceIm.size
(230, 215)
>>> catCopyIm.paste(faceIm, (0, 0))
>>> catCopyIm.paste(faceIm, (400, 500))
>>> catCopyIm.save('pasted.png')
```

نمرّر أولًا مجموعة مربعة للمنطقة المستطيلة في الصورة `zophie.png` التي تحتوي على وجه القطة إلى التابع `crop()`، مما يؤدي إلى إنشاء كائن `Image` يمثل جزءًا مقصوصًا أبعاده 215×230، والذي نخزّنه في المتغير `faceIm`. يمكننا الآن لصق `faceIm` فوق `catCopyIm`، حيث يأخذ التابع `paste()` وسيطين هما: كائن `Image` المصدَر "Source" ومجموعة من إحداثيات x و y لمكان لصق الزاوية العلوية اليسرى من كائن `Image` المصدر على كائن `Image` الرئيسي.

استدعينا في مثالنا التابع `paste()` مرتين مع `catCopyIm`، ومرّرنا المجموعة `(0, 0)` في المرة الأولى والمجموعة `(400, 500)` في المرة الثانية، مما يؤدي إلى لصق `faceIm` على `catCopyIm` مرتين، حيث نلصق الزاوية العلوية اليسرى من `faceIm` عند الإحداثيات `(0, 0)` على `catCopyIm` في المرة الأولى، ونلصق الزاوية العلوية اليسرى من `faceIm` عند الإحداثيات `(400, 500)`.

أخيرًا، نحفظ المتغير `catCopyIm` المُعدَّل في الملف `pasted.png`، وستبدو الصورة كما يلي:



الشكل 96: القطة زوفي بعد لصق وجهها مرتين

> **ملاحظة**: لا يستخدم التابعان `copy()` و `paste()` في الوحدة Pillow حافظة Clipboard حاسوبك بالرغم من أن اسميهما يدلان على ذلك.

لاحظ أن التابع `paste()` يعدّل كائن `Image` في المكان نفسه، أي أنه لا يعيد كائن `Image` جديد مع الصورة المُلصَقة، ولكن إذا أردتَ استدعاء هذا التابع مع الاحتفاظ أيضًا بالنسخة غير المُعدَّلة من الصورة الأصلية، فيجب نسخ الصورة أولًا ثم استدعاء التابع `paste()` مع تلك النسخة.

لنفترض أنك تريد وضع رأس القطة على الصورة بأكملها كما في الشكل التالي، حيث يمكنك تحقيق هذا التأثير باستخدام بضع حلقات `for` فقط، لذا تابع مثال الصدفة التفاعلية من خلال إدخال ما يلي:

```
>>> catImWidth, catImHeight = catIm.size
>>> faceImWidth, faceImHeight = faceIm.size
❶ >>> catCopyTwo = catIm.copy()
❷ >>> for left in range(0, catImWidth, faceImWidth):
    ❸ for top in range(0, catImHeight, faceImHeight):
        print(left, top)
```

```
        catCopyTwo.paste(faceIm, (left, top))
    0 0
    0 215
    0 430
    0 645
    0 860
    0 1075
    230 0
    230 215
--snip--
    690 860
    690 1075
>>> catCopyTwo.save('tiled.png')
```

الشكل 97: حلقات for المتداخلة المستخدمة
مع التابع paste() لتكرار وجه القطة

نخزّن عرض وارتفاع `catIm` في المتغيرين `catImWidth` و `catImHeight`، ثم ننشئ نسخة من `catIm` ونخزّنها في المتغير `catCopyTwo` ❶. أصبح لدينا الآن نسخة يمكننا لصقها، وبالتالي نبدأ بتكرار لصق `faceIm` على `catCopyTwo`، حيث يبدأ المتغير `left` الخاص بحلقة `for` الخارجية من القيمة `0` ويزداد بمقدار `faceImWidth(230)` ❷، ويبدأ المتغير `top` الخاص بحلقة `for` الداخلية من القيمة `0` ويزداد بمقدار `faceImHeight(215)` ❸. تعطي حلقات `for` المتداخلة هذه قيمًا للمتغيرين `left` و `top` للصق شبكةٍ من

صـور faceIm فـوق كـائن Image الـذي هـو catCopyTwo كمـا هـو موضّـح في الشـكل السـابق. نطبـع قيم المتغيرين left و top لرؤية كيفية عمل هذه الحلقات المتداخلة، ثم نحفظ الصورة catCopyTwo المُعدَّلة في الملف tiled.png بعد اكتمال اللصق.

19.2.4 تغيير حجم الصورة

يُستدعَى التابع resize() مع الكائن Image ويعيد كائن Image جديد مع العـرض والارتفـاع المُحـدَّدين، حيث يقبل هذا التابع وسـيطًا هـو مجموعـة مكونـة من عـددين صـحيحين يمثلان العـرض والارتفـاع الجديـدين للصورة المُعادة. أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
❶ >>> width, height = catIm.size
❷ >>> quartersizedIm = catIm.resize((int(width / 2), int(height / 2)))
>>> quartersizedIm.save('quartersized.png')
❸ >>> svelteIm = catIm.resize((width, height + 300))
>>> svelteIm.save('svelte.png')
```

نسند القيمـتين الموجـودتين في المجموعـة catIm.size إلى المتغـيرين width و height ❶، حيث يؤدي اسـتخدام هـذين المتغـيرين بـدلًا من catIm.size[0] و catIm.size[1] إلى جعـل بقيـة الشـيفرة البرمجية أكثر قابلية للقراءة.

يمرّر استدعاء التابع resize() الأول القيمة int(width / 2) للعرض الجديد والقيمة int(height / 2) للارتفاع الجديد ❷، لذا سيكون لكائن Image الذي يعيده التابع resize() نصف طـول ونصـف عـرض الصـورة الأصـلية أو ربـع حجم الصـورة الأصـلية. يقبـل التـابع resize() الأعـداد الصـحيحة فقـط في وسـيط المجموعة الخاص به، ولذلك يجب تغليف عمليتي القسمة على 2 باستدعاء الدالة int().

يحافظ تغيير الحجم على النسب نفسـها للعـرض والارتفـاع، ولكن لا حاجـة إلى أن يكـون العـرض والارتفـاع الجديدين المُمرَّرين إلى التابع resize() متناسبين مع الصورة الأصلية. يحتوي المتغير svelteIm على كـائن Image له العرض الأصلي ولكن يكون ارتفاعه أكبر من الطول الأصلي بمقـدار 300 بكسـل ❸، ممـا يمنح القطـة مظهرًا أرشق. لاحظ أن التابع resize() لا يعدّل كائن Image ذاته، بل يعيد كائن Image جديد.

19.2.5 تدوير وقلب الصور

يمكن تدوير الصور باستخدام التابع `rotate()` الذي يعيد كائن `Image` جديد للصورة المُدوَّرة ويترك كائن `Image` الأصلي دون تغيير. وسيط التابع `rotate()` هو عدد صحيح أو عدد عشري يمثّل عدد الدرجات لتدوير الصورة بعكس اتجاه عقارب الساعة. أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
>>> catIm.rotate(90).save('rotated90.png')
>>> catIm.rotate(180).save('rotated180.png')
>>> catIm.rotate(270).save('rotated270.png')
```

لاحظ كيفية سَلسَلة استدعاءات التوابع من خلال استدعاء التابع `save()` مباشرةً مع كائن `Image` الذي يعيده التابع `rotate()`. يؤدي الاستدعاء الأول للتابعين `rotate()` و `save()` إلى إنشاء كائن `Image` جديد يمثّل الصورة المُدوَّرة بعكس اتجاه عقارب الساعة بمقدار 90 درجة وحفظ الصورة المُدوَّرة في الملف `rotated90.png`، ويفعل الاستدعاءان الثاني والثالث الشيء نفسه، ولكن بمقدار 180 درجة و270 درجة. ستبدو النتائج كما يلي:

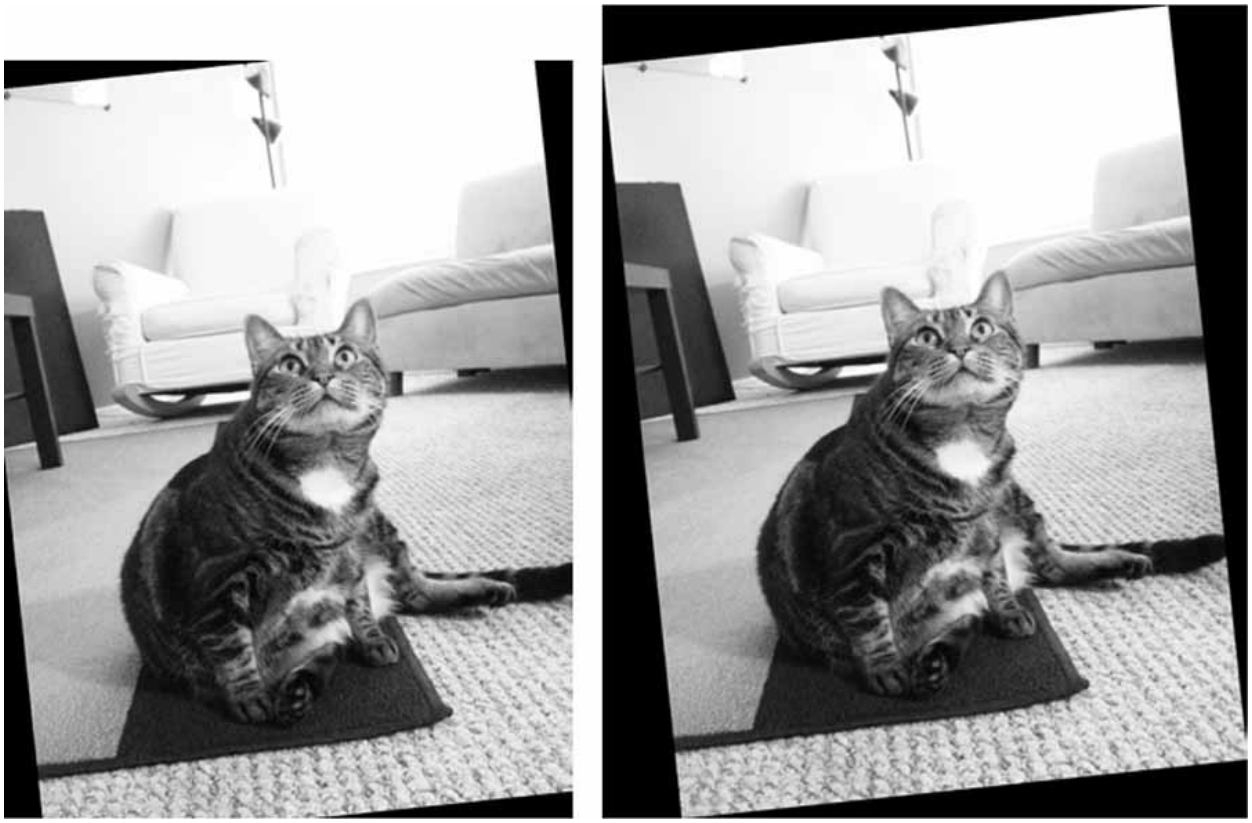
الشكل 98: الصورة الأصلية على اليسار، والصورة المُدوَّرة بعكس اتجاه عقارب الساعة

لاحظ أن عرض الصورة وارتفاعها يتغيران عند تدوير الصورة بمقدار 90 أو 270 درجة، ولكن إذا دوّرتَ الصورة بمقدار آخر، فستحافظ الصورة على الأبعاد الأصلية. نستخدم في نظام ويندوز Windows خلفية سوداء لملء أي فراغات ناتجة عن التدوير، كما هو موضح في الشكل التالي، بينما نستخدم في نظام ماك macOS بكسلات شفافة لهذه الفراغات.

يحتوي التابع `rotate()` على وسيط كلمات مفتاحية Keyword Argument اختياري هو `expand`، حيث يمكن ضبط هذا الوسيط على القيمة `True` لتكبير أبعاد الصورة لتناسب الصورة الجديدة المُدوَّرة بالكامل. أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> catIm.rotate(6).save('rotated6.png')
>>> catIm.rotate(6, expand=True).save('rotated6_expanded.png')
```

يدوّر الاستدعاء الأول الصورة بمقدار 6 درجات ويحفظها في الملف `rotate6.png` كما هو موضَّح على يسار الشكل التالي، ويدوّر الاستدعاء الثاني الصورة بمقدار 6 درجات مع ضبط الوسيط `expand` على القيمة `True` ويحفظها في الملف `rotate6_expanded.png` كما هو موضَّح على يمين الشكل التالي:



الشكل 99: تدوير الصورة تدويرًا عاديًا (باليسار) والتدوير مع الوسيط `expand=True` (باليمين)

يمكنك أيضًا قلب الصورة Flip Mirror باستخدام التابع `transpose()`، حيث يجب تمرير إما المعامل `Image.FLIP_LEFT_RIGHT` أو المعامل `Image.FLIP_TOP_BOTTOM` إلى هذا التابع. أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> catIm.transpose(Image.FLIP_LEFT_RIGHT).save('horizontal_flip.png')
>>> catIm.transpose(Image.FLIP_TOP_BOTTOM).save('vertical_flip.png')
```

ينشئ التابع `transpose()` -مثل التابع `rotate()`- كائن `Image` جديد. مرّرنا في مثالنا المعامل `Image.FLIP_LEFT_RIGHT` إلى هذا التابع لقلب الصورة أفقيًا ثم حفظنا النتيجة في الملف `horizontal_flip.png`، ويمكننا قلب الصورة عموديًا من خلال تمرير `Image.FLIP_TOP_BOTTOM` ونحفظ النتيجة في الملف `vertical_flip.png`.

ستبدو النتائج كما هو موضّح في الشكل التالي:

الشكل 100: النتيجة الظاهرة في الملف vertical_flip.png

19.2.6 تغيير البكسلات الفردية

يمكن استرداد لون البكسل الفـردي أو ضـبطه باسـتخدام التـابعين `getpixel()` و `putpixel()`، حيث يأخذ هذان التابعان مجموعةً تمثل إحداثيات x و y للبكسل، ويأخذ التابع `putpixel()` أيضًا وسيطًا إضافيًا هو مجموعة تمثّل لون البكسل، فهذا الوسيط هو مجموعة RGBA مؤلفة من أربعة أعداد صحيحة أو مجموعـة RGB مؤلفة من ثلاثة أعداد صحيحة. أدخل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> from PIL import Image
❶ >>> im = Image.new('RGBA', (100, 100))
❷ >>> im.getpixel((0, 0))
(0, 0, 0, 0)
❸ >>> for x in range(100):
    for y in range(50):
        ❹ im.putpixel((x, y), (210, 210, 210))

>>> from PIL import ImageColor
❺ >>> for x in range(100):
```

```
        for y in range(50, 100):
            ❻ im.putpixel((x, y), ImageColor.getcolor('darkgray', 'RGBA'))
>>> im.getpixel((0, 0))
(210, 210, 210, 255)
>>> im.getpixel((0, 50))
(169, 169, 169, 255)
>>> im.save('putPixel.png')
```

ننشئ أولًا صورةً جديدة، والتي هي مربع شفاف أبعاده 100×100 ❶، ثم نستدعي التابع `getpixel()` مع بعض الإحداثيات في هذه الصورة، مما يؤدي إلى إعادة المجموعة `(0, 0, 0, 0)` لأن الصورة شفافة ❷.

يمكن تلوين البكسلات في هذه الصورة من خلال استخدام حلقات `for` متداخلة للمرور على جميع البكسلات في النصف العلوي من الصورة ❸ وتلوين كلّ بكسل باستخدام التابع `putpixel()` ❹، حيث مرّرنا مجموعة RGB هي `(210, 210, 210)` باللون الرمادي الفاتح إلى التابع `putpixel()`.

لنفترض أننا نريد تلوين النصف السفلي من الصورة باللون الرمادي الداكن، ولكننا لا نعرف مجموعة RGB للون الرمادي الداكن، حيث لا يقبل التابع `putpixel()` اسم لون معياري مثل `'darkgray'`، لذلك يجب استخدام الدالة `ImageColor.getcolor()` للحصول على مجموعة اللون المقابلة للون `'darkgray'`.

لنمر الآن ضمن حلقة على البكسلات الموجودة في النصف السفلي من الصورة ❺، ونمرّر القيمة المُعادة من الدالة `ImageColor.getcolor()` إلى التابع `putpixel()` ❻، ويجب أن يكون لدينا الآن صورة رمادية فاتحة في النصف العلوي ورمادية داكنة في النصف السفلي كما هو موضّح في الشكل التالي. يمكنك استدعاء التابع `getpixel()` مع بعض الإحداثيات للتأكد من أن لون أيّ بكسل هو ما تتوقعه.

أخيرًا، احفظ الصورة في الملف `putPixel.png`.

الشكل 101: الصورة putPixel.png

لا يُعَد رسم بكسل واحد في كل مرة على الصورة أمـرًا مريحًـا للغايـة، فـإذا كنت بحاجـة إلى رسـم الأشـكال، فاستخدم دوال الوحدة `ImageDraw` التي سنوضّحها لاحقًا.

19.3 تطبيق عملي: إضافة شعار إلى صورة

لنفترض أن لديك مهمة مملة تتمثل في تغيير حجم آلاف الصور وإضافة شعار صغير يمثل علامة مائيـة في زاوية كل من هذه الصور، إذ قد يستغرق ذلك الأمر وقتًا طويلًا باستخدام برنامج رسوميات أساسـي مثـل برنـامج الرسام Paint أو Paintbrush. يمكن لتطبيق رسوميات أكثر تقدمًا مثل برنامج الفوتوشوب Photoshop إنجاز معالجةٍ لمجموعة من الصور، ولكنه يكلف مئات الدولارات. إذًا لنكتب سكربتًا ينجز هذه المهمة نيابةً عنك.

لنفترض أن الشكل التالي هو الشعار الذي تريد إضـافته إلى الزاويـة اليمـنى السـفلية من كـل صـورة، وهـذا الشعار هو رمزٌ لقطة سوداء ذات حدود بيضاء مع جعل بقية الصورة شفافة:

الشكل 102: الشعار المراد
إضافته إلى الصورة

إليك الخطوات العامة التي يجب أن يطبّقها برنامجك:

1. تحميل صورة الشعار.
2. المرور ضمن حلقة على جميع الملفات ذات الامتداد `.png` و `.jpg` الموجودة في مجلد العمل.
3. التحقق مما إذا كانت الصورة أعرض أو أطول من 300 بكسل.
4. إذا كان الأمر كذلك، فيجب تقليل العرض أو الارتفاع (الأكبر) إلى 300 بكسل وتقليص البعد الآخر بمقدارٍ متناسب معه.
5. لصق صورة الشعار في زاوية الصورة.
6. حفظ الصور المُعدَّلة في مجلد آخر.

وبالتالي يجب أن تطبّق شيفرتك البرمجية الخطوات التالية:

1. فتح الملف `catlogo.png` بوصفه كائن `Image`.
2. المرور ضمن حلقة على السلاسل النصية التي يعيدها التابع `os.listdir('.')`.
3. الحصول على عرض الصورة وارتفاعها من السمة `size`.
4. حساب العرض والارتفاع الجديدين للصورة التي غيّرنا حجمها.
5. استدعاء التابع `resize()` لتغيير حجم الصورة.
6. استدعاء التابع `paste()` للصق الشعار.
7. استدعاء التابع `save()` لحفظ التغييرات باستخدام اسم الملف الأصلي.

19.3.1 الخطوة الأولى: فتح صورة الشعار

افتح تبويبًا جديدًا في محرّرك لإنشاء ملف جديد للمشروع، وأدخِل الشيفرة البرمجية التالية، واحفظها بالاسم `resizeAndAddLogo.py`:

```
#! python3
# resizeAndAddLogo.py - تغيير حجم جميع الصور الموجودة في مجلد العمل الحالي لتلائم مربعًا
# أبعاده 300x300، ثم إضافة الصورة catlogo.png إلى الزاوية السفلية اليمنى
import os
from PIL import Image
❶ SQUARE_FIT_SIZE = 300
❷ LOGO_FILENAME = 'catlogo.png'

❸ logoIm = Image.open(LOGO_FILENAME)
❹ logoWidth, logoHeight = logoIm.size
# المرور ضمن حلقة على جميع الملفات الموجودة في مجلد العمل
# التحقق مما إذا كانت الصورة بحاجة إلى تغيير حجمها
# حساب العرض والارتفاع الجديدين لتغيير الحجم
# تغيير حجم الصورة
```

```
    # إضافة الشعار

    # حفظ التغييرات
```

سهّلنا تغيير البرنامج لاحقًا من خلال إعداد الثابتين `SQUARE_FIT_SIZE` ❶ و`LOGO_FILENAME` ❷ في بداية البرنامج، فلنفترض أن الشعار الذي تضيفه ليس رمزًا لقطة، أو لنفترض أنك تقلّل البعد الأكبر للصور الناتجة إلى قيمة مغايرة عن 300 بكسل، إذ يمكنك فتح الشيفرة البرمجية وتغيير تلك القيم مرة واحدة فقط باستخدام هذه الثوابت في بداية البرنامج، أو يمكنك إجراء ذلك بحيث تأخذ قيم هذه الثوابت من وسطاء سطر الأوامر.

إن لم تستخدم هذه الثوابت، فيجب عليك البحث في الشيفرة البرمجية عن جميع نسخ القيم `300` و`'catlogo.png'` ووضع قيمٍ أخرى لمشروعك الجديد مكانها، وبالتالي يجعل استخدام الثوابت برنامجك أعم.

تعيد الدالة `Image.open()` كائن `Image` للشعار ❸. نسند القيم الواردة من السمة `logoIm.size` إلى المتغيرين `logoWidth` و `logoHeight` لسهولة القراءة ❹.

ملاحظة: تُعَد بقية البرنامج شيفرة هيكلية للتعليقات الموجودة في نهاية الشيفرة البرمجية السابقة حاليًا.

19.3.2 الخطوة الثانية: المرور ضمن حلقة على جميع الملفات وفتح الصور

يجب الآن العثور على جميع الملفات ذات الامتداد `.png` و `.jpg` في مجلد العمل الحالي، ولكننا لا نريد إضافة صورة الشعار إلى صورة الشعار نفسها، لذلك يجب على البرنامج تخطي أيّ صورة لاسم ملف مماثل لقيمة الثابت `LOGO_FILENAME`. إذًا أضِف ما يلي إلى شيفرتك البرمجية:

```
    #! python3

    # resizeAndAddLogo.py - تغيير حجم جميع الصور الموجودة في مجلد العمل الحالي لتلائم مربعًا

    # أبعاده 300x300، ثم إضافة الصورة catlogo.png إلى الزاوية السفلية اليمنى

    import os

    from PIL import Image

    --snip--

    os.makedirs('withLogo', exist_ok=True)

    # المرور ضمن حلقة على جميع الملفات الموجودة في مجلد العمل

❶ for filename in os.listdir('.'):
```

```
❷ if not (filename.endswith('.png') or filename.endswith('.jpg')) \
    or filename == LOGO_FILENAME:
  ❸ continue      # تخطي الملفات التي ليست صورًا وملف الشعار نفسه
❹ im = Image.open(filename)
  width, height = im.size
--snip--
```

أولًا، ينشئ استدعاء التابع `os.makedirs()` مجلدًا بالاسم `withLogo` لتخزين الصور النهائية مع الشعار بدلًا من الكتابة فوق ملفات الصور الأصلية، ويمنع وسيط الكلمات المفتاحية `exist_ok=True` التابع `os.makedirs()` من رفع استثناء إذا كان المجلد `withLogo` موجودًا مسبقًا. نمر ضمن حلقة على جميع الملفات الموجودة في مجلد العمل باستخدام التابع `os.listdir('.')` ❶، وتتحقق تعليمة `if` الطويلة ❷ عبر هذه الحلقة مما إذا كانت جميع أسماء الملفات لا تنتهي بالامتداد `.png` أو `.jpg`، فإذا كان الأمر كذلك أو كان الملف صورة الشعار نفسه، فيجب أن تتخطاه الحلقة وتستخدم التعليمة `continue` ❸ للانتقال إلى الملف التالي. إذا كان اسم الملف `filename` ينتهي بالامتداد `.png` أو `.jpg` (وليس ملف الشعار)، فيمكنك فتحه بوصفه كائن `Image` ❹ وضبط العرض `width` والارتفاع `height` الخاصين به.

19.3.3 الخطوة الثالثة: تغيير حجم الصور

يجب أن يغيّر البرنامج حجم الصورة فقط إذا كان العرض أو الارتفاع أكبر من قيمة الثابت `SQUARE_FIT_SIZE` (أي 300 بكسل في مثالنا) فقط، ومن أجل هذا ضع الشيفرة البرمجية الخاصة بتغيير الحجم ضمن تعليمة `if` التي تتحقق من متغيرات العرض `width` والارتفاع `height`. إذًا أضِف الشيفرة البرمجية التالية إلى برنامجك:

```
#! python3
# resizeAndAddLogo.py - تغيير حجم جميع الصور الموجودة في مجلد العمل الحالي لتلائم مربعًا

# أبعاده 300x300، ثم إضافة الصورة catlogo.png إلى الزاوية السفلية اليمنى
import os
from PIL import Image
--snip--
  # التحقق مما إذا كانت الصورة بحاجة إلى تغيير حجمها
  if width > SQUARE_FIT_SIZE and height > SQUARE_FIT_SIZE:
```

```
    # حساب العرض والارتفاع الجديدين لتغيير الحجم
    if width > height:
      ❶ height = int((SQUARE_FIT_SIZE / width) * height)
        width = SQUARE_FIT_SIZE
    else:
      ❷ width = int((SQUARE_FIT_SIZE / height) * width)
        height = SQUARE_FIT_SIZE
    # تغيير حجم الصورة
    print('Resizing %s...' % (filename))
  ❸ im = im.resize((width, height))
--snip--
```

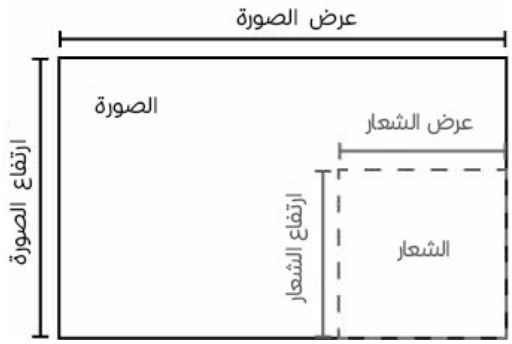
إذا كانت الصورة بحاجة إلى تغيير حجمها، فيجب معرفة ما إذا كان عرض الصورة أو ارتفاعها أكبر من قيمة الثابت `SQUARE_FIT_SIZE` (أي 300 بكسل في مثالنا)، وإذا كان العرض `width` أكبر من الارتفاع `height`، فيجب تقليل الارتفاع بمقدار النسبة نفسها لتقليل العرض ❶، وهذه النسبة هي قيمة الثابت `SQUARE_FIT_SIZE` مقسومة على العرض الحالي، وتساوي قيمة الارتفاع `height` الجديدة النسبة مضروبة بقيمة الارتفاع `height` الحالية. يعيد معامل القسمة قيمة عشرية، ولكن يتطلب التابع `resize()` أن تكون الأبعاد أعدادًا صحيحة، لذا تذكّر تحويل النتيجة إلى عدد صحيح باستخدام الدالة `int()`. أخيرًا، ستُضبَط قيمة العرض `width` الجديدة على قيمة الثابت `SQUARE_FIT_SIZE`.

إذا كان الارتفاع `height` أكبر أو يساوي العرض `width` (عالجنا كلتا الحالتين في تعليمة `else`)، فستُجرَى العملية الحسابية نفسها باستثناء التبديل بين المتغيرين `height` و `width` ❷.

سيحتوي المتغيران `width` و `height` على أبعاد الصورة الجديدة، لذا مرّرهما إلى التابع `resize()` وخزّن كائن `Image` المُعاد في المتغير `im` ❸.

19.3.4 الخطوة الرابعة: إضافة الشعار وحفظ التغييرات

يجب لصق الشعار في الزاوية السفلية اليمنى سواء تغيّر حجم الصورة أم لا، ويعتمد المكان الذي يجب لصق الشعار فيه على حجم الصورة وحجم الشعار، حيث يوضح الشكل التالي كيفية حساب موضع اللصق. سيكون الإحداثي الأيسر لمكان لصق الشعار هو عرض الصورة مطروحًا منه عرض الشعار، وسيكون الإحداثي العلوي لمكان لصق الشعار هو ارتفاع الصورة مطروحًا منه ارتفاع الشعار.



الشكل 103: كيفية حساب موضع اللصق

يجب أن تكون الإحداثيات اليسرى والعلوية لوضع الشعار في الزاوية السفلية اليمنى هي عرض/ارتفاع الصورة مطروحًا منه عرض/ارتفاع الشعار.

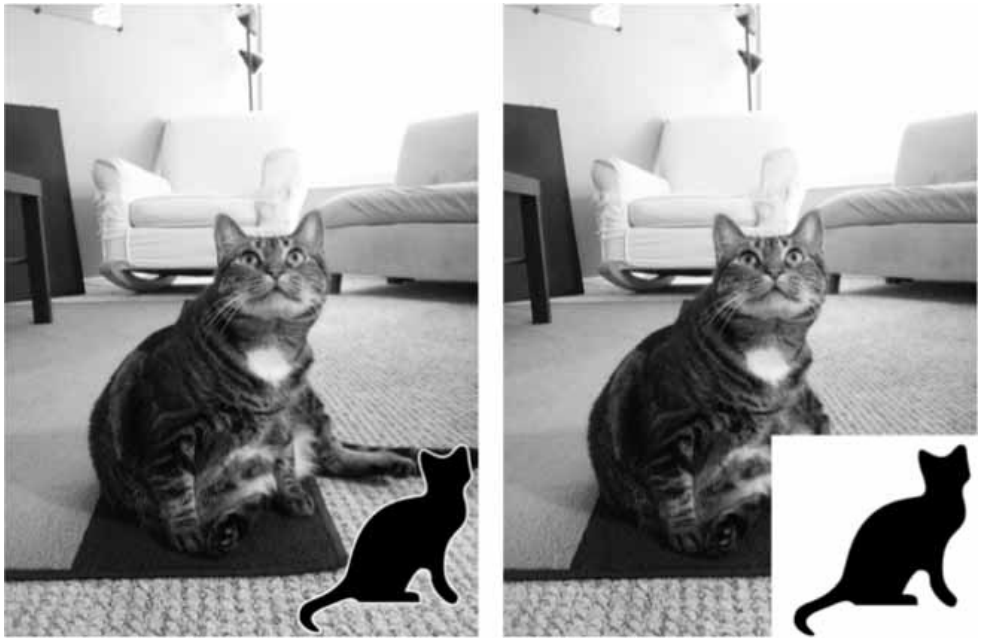
يجب أن تحفظ شيفرتك البرمجية كائن `Image` المُعدَّل بعد لصق الشعار في الصورة. ولهذا أضِف ما يلي إلى برنامجك:

```
#! python3
# resizeAndAddLogo.py - تغيير حجم جميع الصور الموجودة في مجلد العمل الحالي لتلائم مربعًا
# أبعاده 300x300، ثم إضافة الصورة catlogo.png إلى الزاوية السفلية اليمنى
import os
from PIL import Image
--snip--
  # التحقق مما إذا كانت الصورة بحاجة إلى تغيير حجمها
  --snip--
  # إضافة الشعار
❶ print('Adding logo to %s...' % (filename))
❷ im.paste(logoIm, (width - logoWidth, height - logoHeight), logoIm)
  # حفظ التغييرات
❸ im.save(os.path.join('withLogo', filename))
```

تطبع الشيفرة البرمجية الجديدة رسالة تخبر المستخدم بإضافة الشعار ❶، وتلصق `logoIm` على `im` عند الإحداثيات المحسوبة ❷، وتحفظ التغييرات في اسم ملف ضمن المجلد `withLogo` ❸. سيبدو الخرج كما يلي عند تشغيل هذا البرنامج مع الملف `zophie.png` بوصفه الصورة الوحيدة في مجلد العمل:

```
Resizing zophie.png...
Adding logo to zophie.png…
```

سنغيّر الصـورة `zophie.png` إلى صـورة بحجم 300×225 بكسـل تشـبه الشـكل التـالي، وتـذكّر أن التـابع `paste()` لن يلصق البكسلات الشفافة إن لم تمرّر `logoIm` إلى الوسيط الثالث للتـابع `paste()`. يمكن لهـذا البرنامج تغيير حجم مئات الصور وإضافة شعار إليها تلقائيًا في بضع دقائق فقط.



الشكل 104: تغيير الصورة zophie.png إلى صورة بحجم 300×225

غيّرنا حجم الصـورة `zophie.png` وأضـفنا الشـعار (على اليسـار). إذا نسـيت الوسـيط الثـالث، فستُنسَـخ البكسلات الشفافة في الشعار بوصفها بكسلات بيضاء (على اليمين)

19.3.5 أفكار لبرامج مماثلة

يمكن أن تكون القدرة على دمج الصور أو تعديل أحجامهـا دفعـة واحـدة مفيـدة في العديـد من التطبيقـات، حيث يمكنك كتابة برامج مماثلة تنجز المهام التالية:

- إضافة نص أو عنوان URL لموقع ويب إلى الصور.
- إضافة علامات زمنية Timestamps إلى الصور.
- نسخ الصور أو نقلها إلى مجلدات مختلفة بناءً على أحجامها.
- إضافة علامة مائية شفافة تقريبًا إلى الصورة لمنع الآخرين من نسخها.

19.4 الرسم على الصور

إذا أردتَ رسم خطوط أو مستطيلات أو دوائر أو أشكال بسيطة أخرى على صورة ما، فاستخدم الوحدة `ImageDraw` الخاصة بوحدة `Pillow`. أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> from PIL import Image, ImageDraw
>>> im = Image.new('RGBA', (200, 200), 'white')
>>> draw = ImageDraw.Draw(im)
```

نستورد أولًا الوحدتين `Image` و `ImageDraw`، ثم ننشئ صورة جديدة، والتي هي صورة بيضاء أبعادها 200×200 ، ونخزّن كائن `Image` في المتغير `im`. نمرّر كائن `Image` إلى الدالة `ImageDraw.Draw()` للحصول على كائن `ImageDraw`، حيث يحتوي هذا الكائن على عدة توابع لرسم الأشكال والنصوص على كائن `Image`. نخزّن بعد ذلك كائن `ImageDraw` في المتغير `draw` حتى نتمكّن من استخدامه بسهولة في المثال التالي.

19.4.1 رسم الأشكال

ترسم توابع الكائن `ImageDraw` التالية أنواعًا مختلفة من الأشكال على الصورة، وتُعَد معاملات `fill` و`outline` لهذه التوابع اختيارية وستُضبَط افتراضيًا على اللون الأبيض إذا تُرِكت دون تحديد.

ا. رسم النقاط

يرسم التابع `point(xy, fill)` بكسلات فردية، حيث يمثل الوسيط `xy` قائمةً من النقاط التي تريد رسمها، إذ يمكن أن تكون هذه القائمة قائمةً بمجموعات Tuples إحداثيات x و y مثل `[(x, y), (x, y), ...]`، أو قائمة بإحداثيات x و y بدون مجموعات مثل `[x1, y1, x2, y2, ...]`. يمثّل الوسيط `fill` لون النقاط وهو إما مجموعة RGBA أو سلسلة نصية لاسم اللون مثل `'red'`، ويُعَد هذا الوسيط اختياريًا.

ب. رسم الخطوط

يرسم التابع `line(xy, fill, width)` خطًا أو سلسلةً من الخطوط، حيث يمثّل الوسيط `xy` إما قائمة من المجموعات مثل `[(x, y), (x, y), ...]`، أو قائمةً من الأعداد الصحيحة مثل `[x1, y1, x2, y2, ...]`، وتُعَد كلّ نقطة واحدةً من النقاط المتصلة على الخطوط التي ترسمها.

يمثّل الوسيط `fill` الاختياري لون الخطوط باستخدام اسم اللون أو مجموعة RGBA، ويمثّل الوسيط `width` الاختياري عرض الخطوط وتكون قيمته الافتراضية 1 إذا تُرِك دون تحديد.

ج. رسم المستطيلات

يرسم التابع `rectangle(xy, fill, outline)` مستطيلًا، حيث يمثّل الوسيط `xy` مجموعة مربعة وفق الصيغة `(left, top, right, bottom)`، إذ تحدّد القيم `left` و `top` إحداثيات x و y للزاوية العلوية اليسرى للمستطيل، بينما تحدد القيم `right` و `bottom` الزاوية السفلية اليمنى.

يمثّل الوسيط الاختياري `fill` اللونَ الذي سيملأ الجزء الداخلي من المستطيل، ويمثّل الوسيط الاختياري `outline` لون المخطط المحيط بالمستطيل.

د. رسم الأشكال البيضاوية

يرسم التابع `ellipse(xy, fill, outline)` شكلًا بيضاويًا، وإذا كان عرض وارتفاع هذا الشكل متطابقين، فسيرسم هذا التابع دائرة.

يمثّل الوسيط `xy` المجموعة المربعة `(left, top, right, bottom)` التي تمثّل مربعًا يحتوي على الشكل البيضاوي بدقة، ويمثّل الوسيط الاختياري `fill` لون الجزء الداخلي من الشكل البيضاوي، ويمثل الوسيط الاختياري `outline` لون المخطط المحيط بالشكل البيضاوي.

ه. رسم المضلعات

يرسم التابع `polygon(xy, fill, outline)` مضلعًا عشوائيًا، حيث يمثّل الوسيط `xy` قائمةً من المجموعات مثل `[(x, y), (x, y), ...]` أو أعدادًا صحيحة مثل `[x1, y1, x2, y2, ...]`، والتي تمثّل النقاط التي تربط أضلاع المضلع، ويُربَط الزوج الأخير من الإحداثيات بالزوج الأول تلقائيًا.

يمثل الوسيط الاختياري `fill` لون الجزء الداخلي من المضلع، ويمثّل الوسيط الاختياري `outline` لون المخطط المحيط بالمضلع.

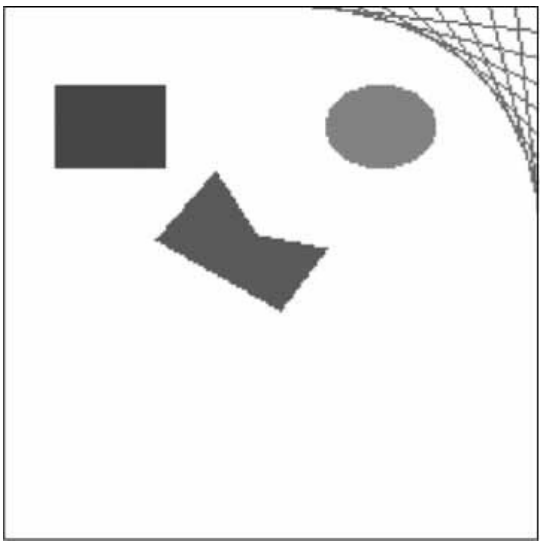
و. تطبيق عملي لرسم الأشكال

أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> from PIL import Image, ImageDraw
>>> im = Image.new('RGBA', (200, 200), 'white')
>>> draw = ImageDraw.Draw(im)
❶ >>> draw.line([(0, 0), (199, 0), (199, 199), (0, 199), (0, 0)],
fill='black')
❷ >>> draw.rectangle((20, 30, 60, 60), fill='blue')
```

```
❸ >>> draw.ellipse((120, 30, 160, 60), fill='red')

❹ >>> draw.polygon(((57, 87), (79, 62), (94, 85), (120, 90), (103,
113)),
 fill='brown')

❺ >>> for i in range(100, 200, 10):
    draw.line([(i, 0), (200, i - 100)], fill='green')

 >>> im.save('drawing.png')
```

ننشئ كائن `Image` لصورة بيضاء أبعادها 200×200، ونمـرّره إلى الدالـة `ImageDraw.Draw()` للحصـول على كائن `ImageDraw` الذي نخزّنه في المتغير `draw`، حيث يمكنـك اسـتدعاء توابـع الرسـم مـع هـذا المتغـير. رسمنا مخططًا رفيعًا باللون الأسود عند حواف الصـورة ❶، ومسـتطيلًا أزرق تكـون زاويتـه العلويـة اليسـرى عنـد النقطة (20, 30) وزاويته السفلية اليمنى عند النقطة (60, 60) ❷، وشكلًا بيضاويًا باللون الأحمر نحدّده باستخدام مربعٍ من النقطة (120, 30) إلى النقطة (160, 60) ❸، ومضلعًا بنيًا بخمس نقاط ❹، ونمطًا Pattern من الخطوط الخضراء مرسومة باستخدام حلقة `for` ❺. سيبدو الملف `drawing.png` الناتج كما يلي:



الشكل 105: صورة drawing.png الناتجة

توجد العديد من توابع رسم الأشكال الأخرى لكائنات `ImageDraw`، لذا اطّلع على توثيقها الكامل على موقع وحدة Pillow الرسمي.

19.4.2 رسم النصوص

يحتوي كائن `ImageDraw` أيضًا على التابع `text()` لرسم النصوص على الصور، حيث يأخذ هذا التابع أربعة وسطاء هي: `xy` و `text` و `fill` و `font`:

- الوسيط `xy` هو مجموعة مكونة من عددين صحيحين تحدّد الزاوية العلوية اليسرى من مربع النص.
- الوسيط `text` هو سلسلة النص الذي تريد كتابته.
- الوسيط `fill` الاختياري هو لون النص.
- الوسيط `font` الاختياري هو كائن `ImageFont`، حيث يُستخدَم هذا الوسيط لضبط خط النص وحجمه (سنوضّح هذا الوسيط بمزيد من التفصيل في القسم التالي).

من الصعب معرفة حجم كتلة النص التي لها خط معين مسبقًا، لذا توفّر الوحدة `ImageDraw` التابع `textsize()`، ووسيطه الأول هو سلسلة النص الذي تريد قياس حجمه، والوسيط الثاني هو كائن `ImageFont` اختياري. يعيد التابع `textsize()` مجموعة مكونة من عددين صحيحين تمثّل العرض والارتفاع الذي سيكون عليه النص الذي له خطٌ محدّد إذا كان مكتوبًا على الصورة، حيث يمكنك استخدام هذا العرض والارتفاع لمساعدتك في حساب المكان الذي تريد وضع النص فيه على صورتك.

تُعَد الوسطاء الثلاثة الأولى للتابع `text()` واضحةً، ولكن لنلقِ نظرة على الوسيط الرابع الاختياري كائن `ImageFont` قبل أن نستخدم التابع `text()` لرسم نص على صورة.

يأخذ كل من التابعين `text()` و `textsize()` كائن `ImageFont` اختياري بوصفه الوسيط الأخير لهما. لننشئ أحد هذه الكائنات من خلال تشغيل التعليمة التالية:

```
>>> from PIL import ImageFont
```

استوردنا الوحدة `ImageFont` الخاصة بوحدة `Pillow`، ويمكننا الآن استدعاء الدالة `ImageFont.truetype()` التي تأخذ وسيطين. الوسيط الأول هو سلسلة نصية لملف TrueType الخاص بالخط، وهو ملف الخط الفعلي الموجود على قرص حاسوبك الصلب، ويكون لملف TrueType الامتداد `.ttf`، حيث يمكن العثور عليه في المجلدات التالية:

- على نظام ويندوز: `C:\Windows\Fonts`.
- على نظام ماك: `/Library/Fonts` و `/System/Library/Fonts`.
- على نظام لينكس: `/usr/share/fonts/truetype`.

لا تحتاج إلى إدخال هذه المسارات بوصفها جزءًا من السلسلة النصية لملف TrueType لأن لغة بايثون تعرف أنها ستبحث تلقائيًا عن الخطوط في هذه المجلدات، ولكنها ستعرض خطأً إن لم تتمكّن من العثور على الخط الذي حدّدته.

الوسيط الثاني للدالة `ImageFont.truetype()` هو عدد صحيح يمثّل حجم الخط بالنقاط بدلًا من البكسلات. ضع في بالك أن الوحدة Pillow تنشئ صور PNG بكثافة 72 بكسلًا لكل بوصة افتراضيًا، والنقطة هي 1/72 من البوصة.

أدخِل مثلًا ما يلي في الصدفة التفاعلية مع وضع اسم المجلد الفعلي الذي يستخدمه نظام تشغيلك مكان الثابت `FONT_FOLDER`:

```
>>> from PIL import Image, ImageDraw, ImageFont
>>> import os
❶ >>> im = Image.new('RGBA', (200, 200), 'white')
❷ >>> draw = ImageDraw.Draw(im)
❸ >>> draw.text((20, 150), 'Hello', fill='purple')
>>> fontsFolder = 'FONT_FOLDER' # مثلًا '/Library/Fonts'
❹ >>> arialFont = ImageFont.truetype(os.path.join(fontsFolder,
'arial.ttf'), 32)
❺ >>> draw.text((100, 150), 'Howdy', fill='gray', font=arialFont)
>>> im.save('text.png')
```

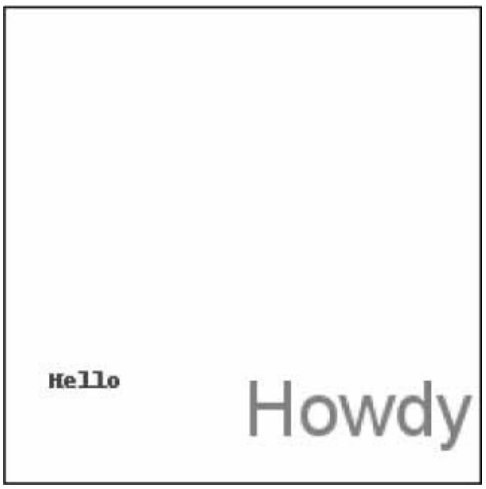
نستورد الوحدات `Image` و `ImageDraw` و `ImageFont` و `os`، ثم ننشئ كائن `Image` لصورة بيضاء جديدة أبعادها 200×200 ❶، وننشئ كائن `ImageDraw` من الكائن `Image` ❷. نستخدم التابع `text()` لرسم النص "Hello" عند النقطة (20, 150) باللون البنفسجي ❸.

لم نمرّر الوسيط الرابع الاختياري في استدعاء التابع `text()`، لذا لم نخصّص خط وحجم هذا النص.

يمكن ضبط خط وحجم النص من خلال تخزين اسم المجلد (مثل /Library/Fonts) في المتغير `fontsFolder`، ثم نستدعي الدالة `ImageFont.truetype()`، ونمرر إليها ملف `.ttf` للخط الذي نريده متبوعًا بعدد صحيح يمثّل حجم الخط ❹.

نخزّن الكائن `Font` الذي نحصل عليه من الدالة `ImageFont.truetype()` في المتغير `arialFont` مثلًا، ثم نمرّر هذا المتغير إلى التابع `text()` في وسيط الكلمات المفتاحية الأخير. يرسم استدعاء التابع `text()` ❺ النص "Howdy" عند النقطة (100, 150) باللون الرمادي بخط Arial وبحجم 32 نقطة.

سيبدو الملف `text.png` الناتج كما في الشكل التالي:



الشكل 106: صورة text.png الناتجة

19.5 أسئلة للتدريب

1. ما هي قيمة RGBA؟
2. كيف يمكنك الحصول على قيمة RGBA للون `'CornflowerBlue'` من وحدة `Pillow`؟
3. ما هي المجموعة المربعة Box Tuple؟
4. ما هي الدالة التي تعيد كائن `Image` لملف صورة اسمه `zophie.png` مثلًا؟
5. كيف يمكنك معرفة عرض وارتفاع الصورة الخاصة بكائن `Image`؟
6. ما هو التابع الذي يمكنك استخدامه للحصول على كائن `Image` لصورة أبعادها 100×100 باستثناء الربع الأيسر السفلي منها؟
7. كيف يمكنك حفظ كائن `Image` بوصفه ملف صورة بعد إجراء تغييرات عليه؟
8. ما هي الوحدة التي تحتوي على الشيفرة البرمجية الخاصة بالوحدة `Pillow` لرسم الأشكال؟
9. لا تحتوي كائنات `Image` على توابع للرسم، إذًا ما هو نوع الكائنات التي تحتوي على توابع للرسم، وكيف تحصل عليها؟

19.6 مشاريع للتدريب

حاول كتابة البرامج التي تؤدي المهام التي سنوضّحها فيما يلي لكسب خبرة عملية أكبر.

19.6.1 توسيع وإصلاح برامج تطبيقنا العملي

يعمل برنامج `resizeAndAddLogo.py` الموجود في هذا الفصل مع ملفات PNG و JPEG، ولكن تدعم الوحدة `Pillow` العديد من صيغ الصور الأخرى، إذًا لنوسّع هذا البرنامج لمعالجة صور GIF و BMP أيضًا. توجد مشكلة صغيرة هي أن البرنامج لا يعدّل ملفات PNG و JPEG إلّا إذا كانت امتدادات الملفات الخاصة بها بأحرف صغيرة، فمثلًا سيعالج هذا البرنامج الملف `zophie.png` ولن يعالج الملف `zophie.PNG`، لذا عدّل الشيفرة البرمجية بحيث يكون التحقق من امتداد الملف غير حساس لحالة الأحرف.

أخيرًا، يُفترَض أن يكون الشعار المُضاف إلى الزاوية السفلية اليمنى مجرد علامة صغيرة، ولكن إذا كانت الصورة بحجم الشعار نفسه تقريبًا، فستبدو النتيجة كما في الشكل التالي، لذا عدّل البرنامج `resizeAndAddLogo.py` بحيث يجب أن يكون عرض وارتفاع الصورة على الأقل ضعف عرض وارتفاع صورة الشعار قبل لصقه، وإلّا فيجب تخطي إضافة الشعار.

الشكل 107: جعل الصورة بحجم الشعار نفسه

ستبدو النتائج غير جميلة عندما لا تكون الصورة أكبر بكثير من الشعار.

19.6.2 تحديد مجلدات الصور الموجودة على القرص الصلب

قد تنقل الملفات من الكاميرا الرقمية إلى مجلدات مؤقتة في مكانٍ ما على القرص الصلب ثم تنسى هذه المجلدات، لذا من الجيد كتابة برنامج يمكنه فحص القرص الصلب بأكمله والعثور على مجلدات الصور التي نسيت مكانها.

حاول كتابة برنامج يمر على كل مجلد موجودٍ على قرص حاسوبك الصلب ويعثر على مجلدات الصور المُحتملة، لذا يجب عليك أولًا تحديد ما يعنيه مجلد الصور، إذًا لنفترض أن مجلد الصور هو أيّ مجلد أكثر من نصف ملفاته صور، ولكن يجب أيضًا تحديد ما هي ملفات الصور، حيث يجب أن يكون لملف الصورة الامتداد `.png` أو `.jpg`. تُعَد الصور الرقمية صورًا كبيرة، إذ يجب أن يكون عرض وارتفاع ملف الصورة أكبر من 500 بكسل، فمعظم صور الكاميرا الرقمية يبلغ عرضها وارتفاعها عدة آلاف من البكسلات. إليك شيفرة هيكلية تقريبية لما قد يبدو عليه هذا البرنامج:

```
#! python3
# استيراد الوحدات وكتابة التعليقات لوصف هذا البرنامج
for foldername, subfolders, filenames in os.walk('C:\\'):
  numPhotoFiles = 0
  numNonPhotoFiles = 0
  for filename in filenames:
    # التحقق مما إذا كان امتداد الملف ليس .png أو .jpg
    if TODO:
      numNonPhotoFiles += 1
      continue     # الانتقال إلى اسم الملف التالي
    # فتح ملف الصورة باستخدام الوحدة Pillow
    # التحقق مما إذا كان العرض والارتفاع أكبر من 500
    if TODO:
      # الصورة كبيرة بما يكفي لعدّها صورة
      numPhotoFiles += 1
    else:
      # الصورة صغيرة جدًا بحيث لا يمكن عدّها صورة
      numNonPhotoFiles += 1
  # إذا كان أكثر من نصف الملفات هي صور، فاطبع المسار المطلق للمجلد
  if TODO:
  print(TODO)
```

يجب أن يطبع البرنامج عند تشغيله المسار المطلق لأي مجلدات صور على الشاشة.

19.6.3 برنامج لإنشاء بطاقات جلوس مخصصة

أنجزنا في الفصل الخامس عشر مشروعًا تدريبيًا لإنشاء دعوات مخصصة لقائمة من الضيوف موجودة في ملف نص عادي، لذا أضِف على هذا المشروع لإنشاء صور لبطاقات الجلوس المخصصة لضيوفك باستخدام الوحدة `pillow`. أنشئ ملف صورة باسم الضيف وبعض زخارف الزهور لكل من الضيوف المدرجين في الملف `guests.txt` الذي يتوفّر على الموارد الموجودة على موقع nostarch، وتتوفر أيضًا صورة زهرة ذات ملكية عامة دون حقوق نشر في هذه الموارد.

أضِف مستطيلًا أسود على حواف صورة الدعوة بحيث تكون كدليل للقص عند طباعة الصورة للتأكد من أن جميع بطاقات جلوس لها الحجم نفسه. تُضبَط ملفات PNG التي تنتجها وحدة `Pillow` على 72 بكسلًا لكل بوصة، لذا تتطلب البطاقة التي أبعادها 5×4 بوصة صورةً بحجم 360×288 بكسل.

19.7 الخلاصة

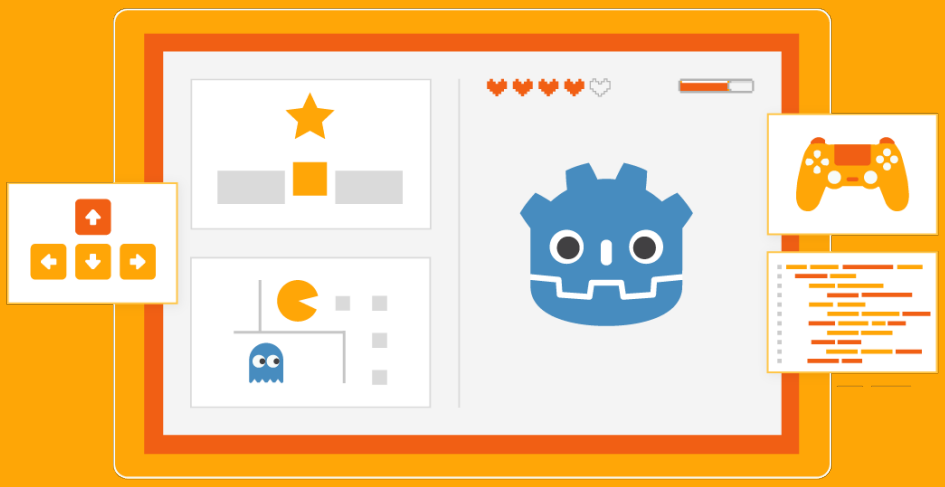
تتكون الصور من مجموعة من البكسلات، وكل بكسل له قيمة RGBA للون الخاص به ويمكن تحديد مكانه باستخدام إحداثيات x و y، وهناك صيغتان شائعتان للصور هما JPEG و PNG، حيث يمكن لوحدة `pillow` التعامل مع هذه الصيغ للصور وغيرها من الصيغ.

إذا حمّلنا صورة إلى كائن `Image`، فستُخزَّن أبعاد العرض والارتفاع الخاصة بها بوصفها مجموعة مكونة من عددين صحيحين في السمة `size`. تمتلك كائنات نوع البيانات `Image` أيضًا توابعًا لمعالجة الصور الشائعة وهي: `crop()` و `copy()` و `paste()` و `resize()` و `rotate()` و `transpose()`. يمكنك حفظ كائن `Image` في ملف صورة من خلال استدعاء التابع `save()`.

إذا أردتَ أن يرسم برنامجك أشكالًا على الصور، فاستخدم توابع الوحدة `ImageDraw` لرسم النقاط والخطوط والمستطيلات والأشكال البيضاوية والمضلعات، وتوفر هذه الوحدة أيضًا توابعًا لرسم النصوص مع استخدام الخط والحجم الذي تختاره.

توفّر التطبيقات المتقدمة وذات الكلفة العالية مثل برنامج الفوتوشوب ميزات معالجة آلية لحزمة من الصور، ولكن يمكنك استخدام سكربتات بايثون لإجراء العديد من التعديلات نفسها مجانًا. كتبنا في الفصول السابقة برامج بايثون للتعامل مع الملفات النصية العادية وجداول البيانات وملفات PDF وغيرها، ووسّعنا قدراتك البرمجية لمعالجة الصور أيضًا باستخدام وحدة `pillow`.

دورة تطوير الألعاب



مميزات الدورة

- بناء معرض أعمال قوي بمشاريع حقيقية
- وصول مدى الحياة لمحتويات الدورة
- تحديثات مستمرة على الدورة مجانًا
- شهادة معتمدة من أكاديمية حسوب
- إرشادات من المدربين على مدار الساعة
- من الصفر دون الحاجة لخبرة مسبقة

اشترك الآن

20. التحكم بلوحة المفاتيح والفأرة

من المفيد معرفة وحدات بـايثون Python المختلفـة لتحريـر جـداول البيانـات وتنزيـل الملفـات وتشـغيل البرامج، ولكن لا توجد في بعض الأحيان أيّ وحداتٍ للتطبيقات التي تحتاج إلى العمل معها، فالأدوات الأساسية لأتمتة المهام على حاسوبك هي البرامج التي تكتبها وتتحكم في لوحة المفاتيح والفأرة مباشرةً، حيث يمكن لهذه البرامج التحكم في التطبيقات الأخرى من خلال إرسال ضغطات مفاتيح افتراضية ونقرات افتراضية بالفأرة إليهـا كما لو أنك جالس أمام حاسوبك وتتفاعل مع التطبيقات بنفسك.

تُعرَف هذه التقنية باسم أتمتة واجهة المستخدم الرسومية Graphical User Interface Automation أو GUI automation اختصارًا، إذ يمكن لبرامجك باستخدام هذه التقنية فعل أيّ شيء يمكن أن يفعله المستخدم الجالس أمام الحاسوب باستثناء سكب القهـوة على لوحـة المفـاتيح طبعًـا. يمكن عَـدّ أتمتـة واجهـة المسـتخدم الرسومية كبرمجة ذراع آلية، حيث يمكنك برمجة الذراع الآلية للكتابـة باسـتخدام لوحـة المفـاتيح وتحريـك الفـأرة نيابةً عنك، وتُعَد هذه التقنية مفيدة خاصةً للمهام التي تتضمن الكثير من النقر أو ملء الاستمارات.

تبيع بعض الشركات حلـولَ الأتمتـة المبتكـرة والمكلفـة، والـتي تُسـوَّق عـادةً بأنهـا أتمتـة العمليـات الآليـة Robotic Process Automation أو RPA اختصارًا، حيث لا تختلف هذه المنتجات فعليًا عن سكربتات بايثون التي يمكنك إنشاؤها بنفسـك باسـتخدام الوحـدة `pyautogui` الـتي تحتـوي على دوال لمحاكـاة حركـات الفـأرة ونقـرات الأزرار وتمريـر عجلـة الفـأرة. سنوضّـح في هـذا الفصـل مجموعـة فرعيـة فقـط من مـيزات الوحـدة `PyAutoGUI`، حيث يمكنك العثور على التوثيق الكامل على موقعها الرسمي.

20.1 تثبيت الوحدة pyautogui

يمكن لوحدة `pyautogui` إرسال ضغطات المفاتيح ونقرات الفأرة الافتراضـية إلى أنظمـة تشـغيل ويـندوز Windows وماك macOS ولينكس Linux، حيث يمكن لمستخدمي ويندوز وماك macOS ببسـاطة اسـتخدام

أداة pip لتثبيت الوحدة PyAutoGUI، ولكن يجب على مستخدمي نظام لينكس أولًا تثبيت بعض البرامج الـتي تعتمد عليها وحدة PyAutoGUI، لذا افتح نافذة طرفية Terminal وأدخِل الأوامر التالية:

```
sudo apt install scrot python3-tk python3-dev
```

يمكنك تثبيت الوحدة PyAutoGUI من خلال تشـغيل الأمـر `pip install --user pyautogui`، ولكن لا تستخدم الأمر sudo مع الأداة pip، إذ يمكن أن تثبِّتَ وحداتٍ مع تثبيت بايثون الـذي يسـتخدمه نظـام التشغيل، مما يتسبب في حدوث تعارضات مع أيّ سكربتات تعتمد على ضبطها الأصلي، ولكن يجب استخدام الأمر sudo عند تثبيت التطبيقات باستخدام apt.

يمكنـك اختبـار صـحة تثـبيت الوحـدة PyAutoGUI من خلال تشـغيل الأمـر `import pyautogui` في الصدفة التفاعلية Interactive Shell والتحقق من وجود رسائل خطأ.

> **ملاحظة**: لا تحفظ برنامجك بالاسم pyautogui.py، إذ ستستورد لغة بايثون برنامجك بدلًا من الوحدة PyAutoGUI وستتلقّى رسائل خطأ مثل الرسالة `AttributeError: module 'pyautogui' has no attribute 'click'` عند تشغيل الأمر `import pyautogui`.

20.2 إعداد تطبيقات إمكانية الوصول Accessibility على نظام macOS

لا يسمح نظام ماك للبرامج بالتحكم في الفأرة أو لوحة المفاتيح كإجراءٍ أمني، لذا يجب ضبط البرنـامج الـذي يشغّل سكربت بايثون ليكون تطبيقًا لإمكانية الوصول لكي تعمل وحدة PyAutoGUI على نظام تشغيل ماك، إذ لن يكون لاستدعاءات دوال PyAutoGUI أيّ تأثير بدون إجراء هذه الخطوة.

اجعل تطبيقك مفتوحًا سواء شغّلته من محرّر Mu أو بيئة IDLE أو الطرفية Terminal، ثم افتح "تفضـيلات النظام System Preferences" وانتقل إلى التبـويب "إمكانيـة الوصـول Accessibility". سـتظهر التطبيقـات المفتوحة حاليًا تحت العنوان "السماح للتطبيقات التالية بالتحكم في حاسوبك Allow the apps below to control your computer". تحقّق من تطـبيق Mu أو IDLE أو الطرفيـة Terminal أو أيّ تطـبيق تسـتخدمه لتشغيل سكربتات بايثون الخاصة بك، وسيُطلَب منك إدخال كلمة مرورك لتأكيد هذه التغييرات.

20.3 البقاء على المسار الصحيح

يجب أن تعرف كيفية التهرب من المشكلات التي قد تواجهـك قبـل الانتقـال إلى أتمتـة واجهـة المسـتخدم الرسومية، فمثلًا يمكن لسكربت بايثون تحريك الفأرة والكتابة من خلال ضغطات المفاتيح بسـرعة مذهلـة، وقـد يكون الأمر سريعًا جدًا بحيث لا تتمكّن البرامج الأخرى من مجاراة هذه السرعة، وإذا حدث خطـأٌ مـا مـع اسـتمرار برنامجك في تحريك الفأرة، فسيكون من الصعب معرفة ما يفعله البرنامج بالضبط أو كيفية حل هـذه المشـكلة. كما يمكن أن يخرج برنامجك عن السيطرة بالرغم من أنه يتبع تعليماتك بطريقة مثالية مثل المكـانس المسـحورة

من فيلم The Sorcerer's Apprentice من إنتاج شركة ديزني، والتي ظلت تملأ حوض ميكي بالماء ثم تملأه أكثر من اللازم، وقد يكون إيقاف البرنامج أمرًا صعبًا إذا كانت الفأرة تتحرك من تلقاء نفسها، مما يمنعك من النقر على نافذة محرّر Mu لإغلاقه. توجد لحسن الحظ عدة طرق لمنع مشاكل أتمتة واجهة المستخدم الرسومية أو حلها، والتي سنوضّحها فيما يلي.

20.3.1 التوقف المؤقت والفشل الآمن

إذا ظهر خطأ في برنامجك ولم تتمكّن من استخدام لوحة المفاتيح والفأرة لإغلاقه، فيمكنك استخدام ميزة الفشل الآمن في وحدة `PyAutoGUI`. حرّك الفأرة بسرعة إلى إحدى زوايا الشاشة الأربعة مثلًا، حيث يكون لكل استدعاء للدالة الخاصة بوحدة `PyAutoGUI` تأخير قدره 10 جزء من الثانية بعد تنفيذ الإجراء الخاص بها ليمنحك وقتًا كافيًا لتحريك الفأرة إلى الزاوية. إذا وجدَت وحدة `PyAutoGUI` بعد ذلك أن مؤشر الفأرة في الزاوية، فستطلق الاستثناء `pyautogui.FailSafeException`. لن يكون للتعليمات التي ليست تابعة لوحدة `PyAutoGUI` هذا التأخير الذي مقداره 10 جزء من الثانية.

إذا وجدت نفسك في موقف تحتاج فيه إلى إيقاف برنامج `PyAutoGUI`، فما عليك سوى تحريك الفأرة بسرعة باتجاه الزاوية لإيقافه.

20.3.2 إغلاق كل شيء من خلال تسجيل الخروج

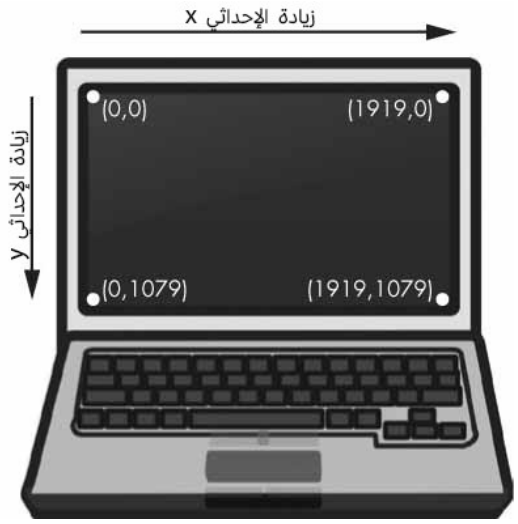
قد تكون أبسط طريقة لإيقاف برنامج أتمتة واجهة المستخدم الرسومية الخارج عن السيطرة هي تسجيل الخروج، مما يؤدي إلى إيقاف تشغيل جميع البرامج التي تكون قيد التشغيل. مفتاح اختصار تسجيل الخروج هو `CTRL-ALT-DEL` في نظامي ويندوز ولينكس، وهو `⌘-Shift-Option-Q` على نظام ماك.

ستفقد أيّ عمل غير محفوظ عند تسجيل الخروج، ولكنك لن تضطر إلى الانتظار حتى تنتهي عملية إعادة التشغيل الكاملة للحاسوب.

20.4 التحكم في حركة الفأرة

ستتعلّم في هذا القسم كيفية تحريك الفأرة وتعقّب موضعها على الشاشة باستخدام الوحدة `PyAutoGUI`، ولكن يجب أولًا أن تفهم كيفية عمل هذه الوحدة مع الإحداثيات.

تستخدم دوال الفأرة الخاصة بوحدة `PyAutoGUI` إحداثيات x و y، حيث يبين الشكل التالي نظام إحداثيات شاشة الحاسوب، وهو مشابه لنظام الإحداثيات المستخدَم مع الصور الذي ناقشناه في الفصل السابق، إذ توجد نقطة الأصل Origin حيث تكون قيمة x و y صفر في الزاوية العلوية اليسرى من الشاشة، وتزداد إحداثيات x باتجاه اليمين، وتزداد إحداثيات y باتجاه الأسفل. تكون جميع الإحداثيات أعدادًا صحيحة موجبة، إذ لا توجد إحداثيات سالبة.



الشكل 108: إحداثيات شاشة الحاسوب بدقة
مقدارها 1920×1080

تمثّل الدقة Resolution عدد البكسلات لعرض وطول الشاشة، حيث إذا كانت دقة شاشتك مضبوطة على القيمة 1920×1080، فستكون إحداثيات الزاوية العلوية اليسرى هو (0, 0)، وستكون إحداثيات الزاوية السفلية اليمنى هو (1919, 1079).

تعيد الدالة `pyautogui.size()` مجموعةً Tuple مكوّنة من عددين صحيحين لعرض الشاشة وارتفاعها بالبكسل. لندخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import pyautogui
>>> wh = pyautogui.size() # الحصول على دقة الشاشة
>>> wh
Size(width=1920, height=1080)
>>> wh[0]
1920
>>> wh.width
1920
```

تعيد الدالة `pyautogui.size()` المجموعة `(1920, 1080)` على حاسوب دقته 1920×1080، إذ قد تختلف القيمة المُعادة اعتمادًا على دقة شاشتك. يُعَد الكائن `Size` الذي تعيده الدالة `size()` مجموعة مُسمَّاة Named Tuples، حيث يكون للمجموعات المُسماة فهارس رقمية مثل المجموعات العادية وأسماء سمات Attribute مثل الكائنات، إذ يُقيَّم كل من `wh[0]` و `wh.width` بأنه عرض الشاشة. لن نشرح المجموعات المُسمَّاة في هذا الفصل، ولكن تذكّر فقط أنه يمكنك استخدامها باستخدام الطريقة نفسها التي تستخدم بها المجموعات العادية.

20.4.1 تحريك الفأرة

تعرّفنا على مفهوم إحداثيات الشاشة، ويمكننا الآن تحريك الفأرة، حيث تحرّك الدالة `pyautogui.moveTo()` مؤشر الفأرة مباشرةً إلى موضعٍ محدّد على الشاشة. تشكّل القيم الصحيحة لإحداثيات x و y الوسيطين الأول والثاني لهذه الدالة على التوالي، ويحدّد وسيط الكلمات المفتاحية Keyword Argument الاختياري `duration` -الذي هو عدد صحيح أو عشري- عدد الثواني التي يجب أن يستغرقها تحريك الفأرة للوصول إلى وِجهتها، وإذا تركتَ هذا الوسيط دون تحديد، فإن القيمة الافتراضية هي `0` للحركة الفورية، وتكون جميع وسطاء الكلمات المفتاحية `duration` في دوال `PyAutoGUI` اختيارية. أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import pyautogui
>>> for i in range(10): # تحريك الفأرة في مربع
...     pyautogui.moveTo(100, 100, duration=0.25)
...     pyautogui.moveTo(200, 100, duration=0.25)
...     pyautogui.moveTo(200, 200, duration=0.25)
...     pyautogui.moveTo(100, 200, duration=0.25)
```

يحرّك المثال السابق مؤشر الفأرة باتجاه عقارب الساعة وفق نمطٍ مربع بين الإحداثيات الأربعة المُعطات 10 مرات، حيث تستغرق كل حركة ربع ثانية كما يحدّده وسيط الكلمات المفتاحية `duration=0.25`، وإن لم تمرّر الوسيط الثالث إلى أيٍّ من استدعاءات الدالة `pyautogui.moveTo()`، فسينتقل مؤشر الفأرة من نقطة إلى أخرى مباشرةً.

تحرّك الدالة `pyautogui.move()` مؤشر الفأرة نسبةً إلى موضعه الحالي، حيث يحرّك المثال التالي الفأرة وفق نمط المربع نفسه، ولكنه يبدأ المربع من أيّ مكان توجد فيه الفأرة على الشاشة عند بدء تشغيل الشيفرة البرمجية:

```
>>> import pyautogui
>>> for i in range(10):
...     pyautogui.move(100, 0, duration=0.25)   # إلى اليمين
...     pyautogui.move(0, 100, duration=0.25)   # للأسفل
...     pyautogui.move(-100, 0, duration=0.25)  # إلى اليسار
...     pyautogui.move(0, -100, duration=0.25)  # للأعلى
```

تأخذ الدالة `pyautogui.move()` أيضًا ثلاثة وسطاء هي: عـدد البكسـلات الـتي يجب تحريكهـا أفقيًـا إلى اليمين، وعدد البكسلات التي يجب تحريكها عموديًـا للأسـفل، والمـدة الـتي يجب أن يسـتغرقها إكمـال الحركـة (اختياريًا). سيؤدي استخدام العدد الصحيح السالب مع الوسيط الأول أو الثاني إلى تحريـك الفـأرة إلى اليسـار أو للأعلى على التوالي.

20.4.2 الحصول على موضع الفأرة

يمكنك تحديد موضع الفـأرة الحـالي من خلال اسـتدعاء الدالـة `pyautogui.position()` الـتي سـتعيد المجموعة المُسمَّاة `Point` لموضعي x و y الخاصين بمؤشر الفأرة عند اسـتدعاء الدالـة. أدخِـل مثلًا مـا يلي في الصدفة التفاعلية مع تحريك الفأرة بعد كل استدعاء:

```
>>> pyautogui.position() # الحصول على موضع الفأرة الحالي
Point(x=311, y=622)
>>> pyautogui.position() # الحصول على موضع الفأرة الحالي مرة أخرى
Point(x=377, y=481)
>>> p = pyautogui.position() # الحصول على موضع الفأرة الحالي مرة أخرى
>>> p
Point(x=1536, y=637)
>>> p[0] # يقع الإحداثي x عند الفهرس 0
1536
>>> p.x # يوجد الإحداثي x أيضًا في السمة x
1536
```

ستختلف قيمك المُعادة اعتمادًا على مكان مؤشر الفأرة.

20.5 التحكم في تفاعل الفأرة

تعرّفتَ كيفية تحريك الفأرة ومعرفة مكانها على الشاشة، وأصبحتَ الآن جاهزًا لبدء النقر والسحب والتمرير.

20.5.1 النقر باستخدام الفأرة

يمكنـك إرسـال نقـرة افتراضـية باسـتخدام الفـأرة إلى حاسـوبك من خلال اسـتدعاء التـابع `pyautogui.click()`، حيث تستخدم هذه النقرة زر الفأرة الأيسر افتراضيًا وتُطبَّـق في أيّ مكـان يوجـد فيـه مؤشر الفأرة حاليًا. يمكنك تمرير إحداثيات x و y لهذه النقرة كوسيط أول وثـانٍ اختيـاريين إلى التـابع إذا أدرتَ أن تُطبَّق في مكانٍ آخر غير موضع الفأرة الحالي.

إذا أدرتَ تحديد زر الفأرة الذي يجب استخدامه، فضمّن وسيط الكلمات المفتاحية `button` مع قيم `'left'` أو `'middle'` أو `'right'`، فمثلًا سيؤدي الاستدعاء التالي:

```
pyautogui.click(100, 150, button='left')
```

إلى النقر على زر الفأرة الأيسر عند الإحداثيات (100, 150)، بينما سيؤدي الاستدعاء `pyautogui.click(200, 250, button='right')` إلى النقر بزر الفأرة الأيمن، وهذا عند الإحداثيات (200, 250).

أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import pyautogui
>>> pyautogui.click(10, 5) # تحريك الفأرة إلى الإحداثيات (10, 5) ثم النقر
```

يُفترَض أن ترى مؤشر الفأرة يتحرك بالقرب من الزاوية العلوية اليسرى من الشاشة ثم يحدث النقر مرة واحدة. يمكن تعريف "النقرة" الكاملة على أنها الضغط على زر الفأرة للأسفل ثم تحريره للأعلى دون تحريك المؤشر، ويمكنك أيضًا إجراء نقرة من خلال استدعاء الدالة `pyautogui.mouseDown()` التي تضغط زر الفأرة للأسفل فقط، ثم استدعاء الدالة `pyautogui.mouseUp()` التي تحرّر الزر. تمتلك هاتان الدالتان وسطاء الدالة `click()` نفسها، ولكن تُعَد الدالة `click()` مجرد دالة مغلِّفة ملائمة لهاتين الدالتين.

تنقر الدالة `pyautogui.doubleClick()` نقرتين باستخدام زر الفأرة الأيسر، بينما تنقر الدالتان `pyautogui.rightClick()` و `pyautogui.middleClick()` نقرة واحدة باستخدام زري الفأرة الأيمن والأوسط على التوالي.

20.5.2 سحب Dragging الفأرة

يعني السحب تحريكَ الفأرة أثناء الضغط باستمرار على أحد أزرارها، فمثلًا يمكنك نقل الملفات بين المجلدات من خلال سحب أيقونات المجلدات، أو يمكنك نقل المواعيد في تطبيق التقويم من خلال سحبها باستخدام الفأرة.

توفر وحدة `PyAutoGUI` الدالتين `pyautogui.dragTo()` و `pyautogui.drag()` لسحب مؤشر الفأرة إلى موقع جديد أو موقع متعلق بموقعه الحالي. تستخدم الدالتان `dragTo()` و `drag()` وسطاء الدالتين `moveTo()` و `move()` نفسها وهي: حركة الإحداثي x أو الحركة أفقيًا وحركة الإحداثي y أو الحركة عموديًا ومدة زمنية اختيارية. لا يطبّق نظام ماك السحب تطبيقًا صحيحًا عندما تتحرك الفأرة بسرعة كبيرة، لذا يوصَى بتمرير وسيط الكلمات المفتاحية `duration`.

لنجـرّب هـذه الـدوال، لـذا افتح تطـبيق رسـمٍ مثـل تطـبيق الرسـام MS Paint على ويـنـدوز أو تطـبيق Paintbrush على نظام ماك ، أو تطبيق GNU Paint على نظام لينكس، حيث سنستخدم وحدة PyAutoGUI للرسم في هذه التطبيقات. إن لم يكن لديك تطبيق رسم، فيمكنك استخدام تطبيق sumopaint عبر الإنترنت.

أدخِل ما يلي في نافذة ملفٍ جديد في محرّرك واحفظه بالاسم spiralDraw.py مع وجـود مؤشـر الفـأرة على لوحة الرسم الخاصة بتطبيق الرسم وتحديد أداة القلم Pencil أو الفرشاة Brush:

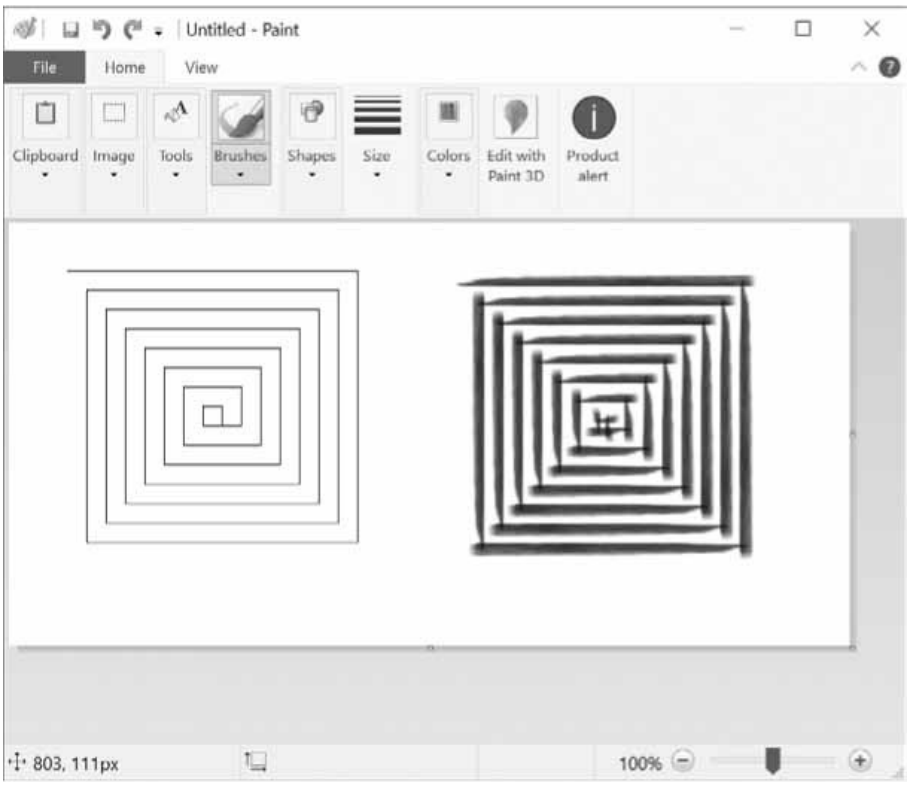
```
import pyautogui, time

❶ time.sleep(5)

❷ pyautogui.click()     # النقر لتنشيط النافذة

distance = 300

change = 20

while distance > 0:

 ❸ pyautogui.drag(distance, 0, duration=0.2)    # التحرك يمينًا

 ❹ distance = distance - change

 ❺ pyautogui.drag(0, distance, duration=0.2)    # التحرك للأسفل

 ❻ pyautogui.drag(-distance, 0, duration=0.2)   # التحرك يسارًا

  distance = distance - change

  pyautogui.drag(0, -distance, duration=0.2)   # التحرك للأعلى
```

سيكون هناك تأخير لمدة خمس ثوانٍ ❶ عند تشـغيل هـذا البرنـامج لتتمكّن من تحريـك مؤشـر الفـأرة على نافذة برنامج الرسم مـع تحديـد أداة القلم أو الفرشـاة، ثم سـيتحكّم برنـامج spiralDraw.py في الفـأرة وينقـر لتنشيط نافذة برنامج الرسم ❷.

النافذة النشطة هي النافذة التي تقبل حاليًا الإدخال من لوحـة المفـاتيح، وسـتؤثّر الإجـراءات الـتي تتخـذها مثل الكتابة أو سحب الفأرة على تلك النافذة، وتُعرَف النافذة النشطة أيضًا بالنافذة المُركَّزة أو النافذة الأمامية.

يرسم برنامج spiralDraw.py نمطًا حلزونيًا مربعًا مثل النمط الموجود على يسـار الشـكل التـالي بعـد أن يصبح برنامج الرسم نشطًا. يمكنك أيضًا إنشاء صورة حلزونية مربعة باسـتخدام الوحـدة Pillow الـتي ناقشـناها في الفصل السابق، ولكن يتيح لـك إنشـاء الصـورة من خلال التحكم في الفـأرة لرسـمها في برنـامج الرسـام MS Paint الاستفادةَ من أنماط الفرشاة المتنوعة لهذا البرنامج كمـا في الشـكل الموجـود على يمين الشـكل التـالي،

بالإضافة إلى ميزات متقدمة أخرى مثل التدرجات أو أداة التعبئة، حيث يمكنك تحديد إعدادات الفرشاة مسبقًا بنفسك أو جعل شيفرة بايثون الخاصة بك تحدّد هذه الإعدادات، ثم يمكنك تشغيل برنامج الرسم الحلزوني.



الشكل 109: نتائج مثال استخدام دالةمرسومة باستخدام فُرش برنامج الرسام المختلفة

يبدأ المتغير `distance` عند القيمة 200، لذلك يسحب استدعاء الدالة `drag()` الأول المؤشر بمقدار 200 بكسل إلى اليمين، ويستغرق ذلك 0.2 ثانية ❸ في التكرار الأول لحلقة `while`، ثم تُقلَّل قيمة المتغير `distance` إلى القيمة 195 ❹، ويسحب استدعاء الدالة `drag()` الثاني المؤشر بمقدار 195 بكسل للأسفل ❺. يسحب استدعاء الدالة `drag()` الثالث المؤشر بمقدار 195- أفقيًا (أي بمقدار 195 إلى اليسار) ❻، وتُقلَّل قيمة المتغير `distance` إلى 190، ويسحب استدعاء `drag()` الأخير المؤشر بمقدار 190 بكسل للأعلى.

تُسحَب الفأرة إلى اليمين والأسفل واليسار والأعلى في كل تكرار، وتكون قيمة المتغير `distance` أصغر قليلًا مما كانت عليه في التكرار السابق.

يمكنك تحريك مؤشر الفأرة لرسم شكل حلزوني مربع من خلال تكرار هذه الشيفرة البرمجية.

يمكنك رسم هذا الحلزوني يدويًا (أو باستخدام الفأرة)، ولكن يجب أن تعمل ببطء لتكون دقيقًا جدًا، ولكن يمكن للوحدة `PyAutoGUI` إنجاز ذلك في بضع ثوانٍ.

> **ملاحظة**: لا تستطيع الوحدة PyAutoGUI حاليًا إرسال نقرات الفأرة أو ضغطات المفاتيح إلى برامج معينة مثل برامج مكافحة الفيروسات (لمنع الفيروسات من تعطيل البرنامج) أو ألعاب الفيديو على نظام ويندوز (التي تستخدم طريقة مختلفة لتلقي دخل الفأرة ولوحة المفاتيح). يمكنك التحقق من توثيق الوحدة PyAutoGUI على موقعها الرسمي لمعرفة ما إذا كانت هذه الميزات مدعومة في نظامك.

20.5.3 التمرير بالفأرة

دالة الوحدة `PyAutoGUI` الأخيرة الخاصة بالفأرة هي الدالة `scroll()` التي نمرّر إليها وسيطًا نوعه عدد صحيح يمثّل عدد الوحدات التي تريد تمرير الفأرة عبرها للأعلى أو للأسفل، حيث يختلف حجم هذه الوحدة باختلاف نظام التشغيل والتطبيق، لذا يجب أأن تجرّب لمعرفة مقدار التمرير في حالتك، ويُطبَّق التمرير عند الموضع الحالي لمؤشر الفأرة.

يؤدي تمرير عدد صحيح موجب إلى التمرير للأعلى، بينما يؤدي تمرير عدد صحيح سالب إلى التمرير للأسفل.

شغّل ما يلي في الصدفة التفاعلية للمحرّر Mu أثناء وجود مؤشر الفأرة على نافذة هذا المحرّر:

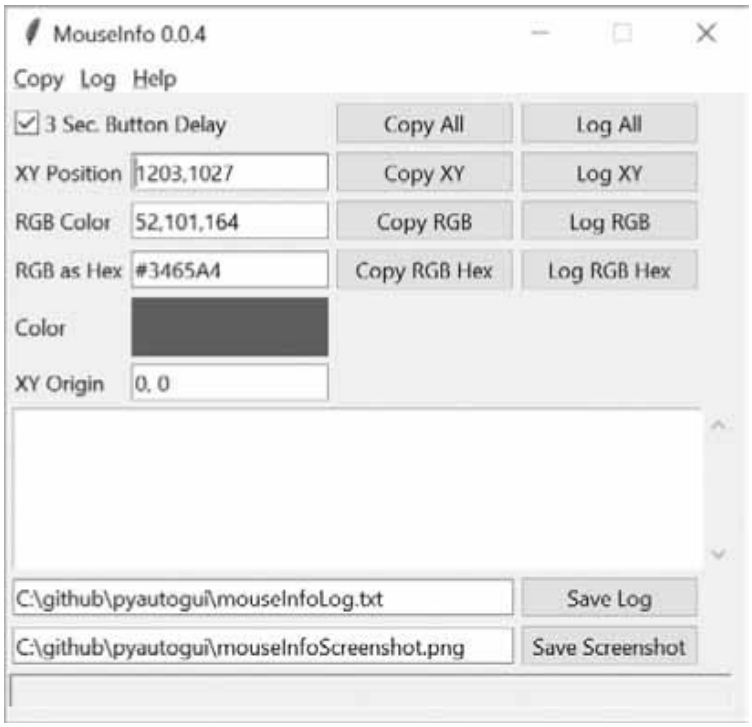
```
>>> pyautogui.scroll(200)
```

سترى أن برنامج Mu يُمرَّر للأعلى إذا كان مؤشر الفأرة على حقل نصي يمكن تمريره للأعلى.

20.6 تخطيط حركات الفأرة

إحدى صعوبات كتابة برنامج يؤتمت عملية النقر على الشاشة هي العثور على إحداثيات x و y للأشياء التي ترغب في النقر عليها، ولكن يمكن أن تساعدك الدالة `pyautogui.mouseInfo()` في هذا الأمر، حيث يُفترَض استدعاء هذه الدالة من الصدفة التفاعلية، وليس كجزء من برنامجك.

تشغِّل هذه الدالة تطبيقًا صغيرًا اسمه MouseInfo المُضمَّن مع الوحدة `PyAutoGUI`، وتبدو نافذة هذا التطبيق كما يلي:



الشكل 110: نافذة التطبيق MouseInfo

أدخِل الآن ما يلي في الصدفة التفاعلية:

```
>>> import pyautogui
>>> pyautogui.mouseInfo()
```

يؤدي ذلك إلى ظهور نافذة تطبيق MouseInfo، حيث توفر لك هذه النافذة معلومات حول الموضع الحالي لمؤشر الفأرة، بالإضافة إلى لون البكسل الموجود تحت مؤشر الفأرة كمجموعة RGB مكونة من ثلاثة أعداد صحيحة وكقيمة ست عشرية، حيث يظهر اللون نفسه في مربع اللون Color الموجود في النافذة.

يمكنك تسجيل معلومات الإحداثيات أو البكسلات من خلال النقر على أحد أزرار النسخ `Copy` أو التسجيل `Log` الثمانية، حيث ستنسخ أزرار `Copy All` و `Copy XY` و `Copy RGB` و `Copy RGB Hex` معلوماتها الخاصة في الحافظة Clipboard، وستكتب الأزرار `Log All` و `Log XY` و `Log RGB` و `Log RGB Hex` معلوماتها الخاصة في الحقل النصي الكبير من هذه النافذة، ويمكنك حفظ النص الموجود في هذا الحقل لتسجيل النص بالنقر على زر الحفظ `Save Log`.

لاحظ تحديد مربع الاختيار `3 Sec. Button Delay` افتراضيًا، مما يتسبب في تأخير لمدة 3 ثوانٍ بين النقر على زر النسخ `Copy` أو التسجيل `Log` وحدوث النسخ أو التسجيل، ويمنحك ذلك وقتًا قصيرًا للنقر على الزر ثم تحريك الفأرة إلى الموضع المطلوب. قد يكون من الأسهل إلغاء تحديد مربع الاختيار `3 Sec. Button Delay`، وتحريك الفأرة إلى موضعٍ معين، ثم الضغط على المفاتيح من F1 إلى F8 لنسخ موضع الفأرة أو

تسجيله. يمكنك إلقاء نظرة على قوائم النسخ والتسجيل الموجودة أعلى نافذة تطبيق MouseInfo لمعرفة المفاتيح المرتبطة بهذه الأزرار.

ألغِ مثلًا تحديد مربع الاختيار `3 Sec. Button Delay`، ثم حرّك الفأرة على الشاشة أثناء الضغط على الزر F6، ولاحظ كيفية تسجيل إحداثيات x و y للفأرة في الحقل النصي الكبير في منتصف النافذة، حيث يمكنك لاحقًا استخدام هذه الإحداثيات في سكربتات `PyAutoGUI` الخاصة بك.

اطّلع على توثيق تطبيق MouseInfo الكامل لمزيدٍ من المعلومات.

20.7 العمل مع الشاشات

ليس من الضروري أن تنقر أو تكتب برامجُك لأتمتة واجهة المستخدم الرسومية دون رؤية ما يحدث، إذ تحتوي الوحدة `PyAutoGUI` على ميزات لقطة الشاشة التي يمكنها إنشاء ملف صورة بناءً على محتويات الشاشة الحالية، ويمكن لهذه الدوال أيضًا إعادة كائن `Image` الخاص بالوحدة `Pillow` لمظهر الشاشة الحالي. اطّلع على الفصل السابق وثبّت الوحدة `pillow` قبل الاستمرار في هذا القسم.

يجب أن تثبّت برنامج `scrot` على الحواسيب التي تعمل على نظام لينكس لاستخدام دوال لقطة الشاشة في الوحدة `PyAutoGUI`، لذا شغّل الأمر `sudo apt install scrot` في نافذة الطرفية لتثبيت هذا البرنامج. إذا كنت تستخدم نظام تشغيل ويندوز أو ماك، فانتقل إلى الخطوة التالية من هذا القسم.

20.7.1 الحصول على لقطة الشاشة

يمكنك التقاط لقطات شاشة في لغة بايثون من خلال استدعاء الدالة `pyautogui.screenshot()`، لذا أدخِل ما يلي في الصدفة التفاعلية:

```
>>> import pyautogui
>>> im = pyautogui.screenshot()
```

سيحتوي المتغير `im` على كائن `Image` للقطة الشاشة، وبالتالي يمكنك الآن استدعاء التوابع مع كائن `Image` في المتغير `im` مثل أيّ كائن `Image` آخر. اطّلع على الفصل السابق للحصول على مزيدٍ من التفاصيل حول كائنات `Image`.

20.7.2 تحليل لقطة الشاشة

لنفترض أن إحدى الخطوات في برنامجك لأتمتة واجهة المستخدم الرسومية هي النقر على زر رمادي، حيث يمكنك التقاط لقطة شاشة قبل استدعاء التابع `click()` وإلقاء نظرة على البكسل الذي سينقر سكربتك عليه، فإن لم يكن لون هذا البكسل رماديًا مثل الزر الرمادي، فهذا يعني أن برنامجك يعلم أن هناك خطأً ما، وبالتالي قد

تتحرك النافذة بطريقة غير متوقعة، أو قد يوقِف مربع حوار منبثق الزر. يمكن لبرنامجك عندها رؤية أنه لا ينقر على الشيء الصحيح ويوقِف نفسه بدلًا من الاستمرار وإحداث فوضى من خلال النقر على الشيء الخطأ.

يمكنك الحصول على قيمة لون RGB لبكسل معين على الشاشة باستخدام الدالة `pixel()`. أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import pyautogui
>>> pyautogui.pixel((0, 0))
(176, 176, 175)
>>> pyautogui.pixel((50, 200))
(130, 135, 144)
```

مرّر مجموعة من الإحداثيات مثل (0, 0) أو (50, 200) إلى الدالة `pixel()` التي ستخبرك بلون البكسل عند تلك الإحداثيات في صورتك. القيمة المُعادة من الدالة `pixel()` هي مجموعة RGB مكونة من ثلاثة أعداد صحيحة تمثّل مقدار اللون الأحمر والأخضر والأزرق في البكسل، ولا توجد قيمة رابعة تمثّل قيمة ألفا Alpha، لأن صور لقطة الشاشة معتمة Opaque تمامًا.

تعيد الدالة `pixelMatchesColor()` الخاصة بوحدة `PyAutoGUI` القيمة `True` إذا تطابق البكسل الموجود عند إحداثيات x و y المُعطاة على الشاشة مع اللون المُعطَى. يُعَد الوسيطان الأول والثاني أعدادًا صحيحة تمثّل إحداثيات x و y، والوسيط الثالث هو مجموعة مكونة من ثلاثة أعداد صحيحة تمثّل لون RGB الذي يجب أن يتطابق مع البكسل الموجود على الشاشة.

أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import pyautogui
❶ >>> pyautogui.pixel((50, 200))
(130, 135, 144)
❷ >>> pyautogui.pixelMatchesColor(50, 200, (130, 135, 144))
True
❸ >>> pyautogui.pixelMatchesColor(50, 200, (255, 135, 144))
False
```

استخدمنا الدالة `pixel()` للحصول على مجموعة RGB التي تمثّل لون البكسل عند إحداثيات مُحـدَّدة ❶، وسنمرّر الآن الإحداثيات نفسها ومجموعة RGB إلى الدالة `pixelMatchesColor()` ❷، والتي يجب أن تعيد القيمة `True`.

نغيّـر بعـد ذلـك قيمـةً من مجموعـة RGB ونسـتدعي الدالـة `pixelMatchesColor()` مـرةً أخـرى مـع الإحداثيات نفسها ❸، حيث يجب أن تعيد القيمة `False`.

يمكن أن يكون استدعاء هذه الدالة مفيدًا عنـدما تكـون برامجـك لأتمتـة واجهـة المسـتخدم الرسـومية على وشك استدعاء التابع `click()`

. لاحظ أن اللون عند الإحداثيات المحددة يجب أن يكـون متطابقًـا تمامًـا، حيث إذا كـان مختلفًـا قليلًا مثـل (255, 255, 254) بدلًا من (255, 255, 255)، فستعيد الدالة `pixelMatchesColor()` القيمة `False`.

20.8 التعرف على الصور

إن لم تكن على معرفة مسبقة بالمكان الذي يجب أن تنقـر عليـه الوحـدة `PyAutoGUI`، فيمكنـك اسـتخدام ميزة التعرّف على الصور، لذا أعطِ وحدة `PyAutoGUI` صورةً لما تريد النقر عليه ودعه يكتشف الإحداثيات.

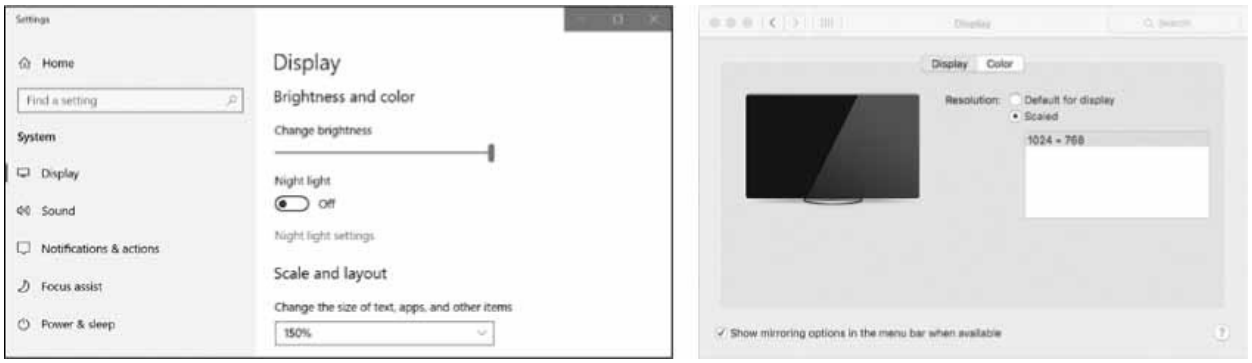
إذا التقطتَ مسبقًا لقطة شاشة للحصول على صورة زر الإرسـال Submit في الملـف `submit.png` مثلًا، فستعيد الدالة `locateOnScreen()` إحداثيات مكان وجود تلك الصورة.

لنتعرّف على كيفية عمل هذه الدالة، لذا حاول التقاط لقطة شاشة لمنطقة صـغيرة من شاشـتك، ثم احفـظ الصـورة وأدخِـل مـا يلي في الصـدفة التفاعليـة مـع وضـع اسـم ملـف لقطـة الشاشـة الخاصـة بـك مكـان `'submit.png'`:

```
>>> import pyautogui
>>> b = pyautogui.locateOnScreen('submit.png')
>>> b
Box(left=643, top=745, width=70, height=29)
>>> b[0]
643
>>> b.left
643
```

يُعَـد كـائن `Box` مجموعـةً مسـماة تعيـدها الدالـة `locateOnScreen()` ولـه إحـداثي x للحافـة اليسـرى وإحداثي y للحافة العلوية والعرض والارتفاع لمكان العثور على الصـورة الأول على الشاشـة. إذا طبّقتَ ذلـك على حاسوبك باستخدام لقطة شاشتك، فستكون القيمة المُعادة مختلفة عن القيمة الموضحة في مثالنا.

إذا تعذر العثور على الصورة على الشاشة، فستعيد الدالة `locateOnScreen()` القيمة `None`. لاحظ أن الصورة الموجودة على الشاشة يجب أن تتطابق تمامًا مع الصورة المُقدَّمة للتعرّف عليها، حيث إذا كانت الصورة مختلفة ببكسل واحد، فسترفع الدالة `locateOnScreen()` الاستثناء `ImageNotFoundException`. إذا غيّرتَ دقة الشاشة، فقد لا تتطابق الصور من لقطات الشاشة السابقة مع الصور الموجودة على شاشتك الحالية، لذا يمكنك تغيير القياسات في إعدادات العرض لنظام تشغيلك كما هو موضّح في الشكل التالي:



الشكل 111: إعدادات قياسات العرض في نظام ويندوز 10 ونظام ماك

إذا عُثِر على الصورة في عدة أماكن على الشاشة، فستعيد الدالة `locateAllOnScreen()` كائن `Generator` الذي يمكنك تمريره إلى التابع `list()` لإعادة قائمة من المجموعات المكونة من أربعة أعداد صحيحة، حيث ستوجد مجموعة واحدة مكونة من أربعة أعداد صحيحة لكل موقع توجد فيه الصورة على الشاشة.

تابع مثال الصدفة التفاعلية من خلال إدخال ما يلي مع وضع ملف الصورة الخاص بك مكان `'submit.png'`:

```
>>> list(pyautogui.locateAllOnScreen('submit.png'))
[(643, 745, 70, 29), (1007, 801, 70, 29)]
```

تمثل كل مجموعة من هذه المجموعات المكونة من أربعة أعداد صحيحة منطقةً من الشاشة، حيث تظهَر الصورة في موقعين في المثال السابق. إذا وُجِدت صورتُك في منطقةٍ واحدة فقط، فسيؤدي استخدام التابع `list()` والدالة `locateAllOnScreen()` إلى إعادة قائمة تحتوي على مجموعة واحدة فقط.

نحصل على المجموعة المكونة من أربعة أعداد صحيحة التي تمثّل الصورة التي تريد تحديدها، ثم يمكننا النقر على مركز هذه المنطقة من خلال تمرير هذه المجموعة إلى التابع `click()`. لندخل الآن مثلًا ما يلي في الصدفة التفاعلية:

```
>>> pyautogui.click((643, 745, 70, 29))
```

يمكنك أيضًا تمرير اسم ملف الصورة مباشرةً إلى التابع `click()` كما يلي:

```
>>> pyautogui.click('submit.png')
```

تقبل الدالتان `moveTo()` و `dragTo()` أيضًا وسطاء لاسم ملف الصورة. تذكّر أن الدالة `locateOnScreen()` ترفع استثناءً إن لم تتمكّن من العثور على الصورة على الشاشة، لذا يجب أن تستدعيها من داخل تعليمة `try` كما يلي:

```
try:
  location = pyautogui.locateOnScreen('submit.png')
except:
  print('Image could not be found.')
```

سيؤدي استثناء عدم العثور على الصورة على الشاشة إلى تعطّل برنامجك إن لم تستخدم تعليمات `try` و`except`، لذا من الجيد استخدام تعليمات `try` و `except` عند استدعاء الدالة `locateOnScreen()` لأنك لا تستطيع التأكّد من أن برنامجك سيعثر على الصورة دائمًا.

20.9 الحصول على معلومات النافذة

يُعَد التعرّف على الصور طريقة ضعيفة للعثور على الأشياء التي تظهر على الشاشة، حيث إذا كان هناك بكسل واحد بلون مختلف، فلن تتمكن الدالة `pyautogui.locateOnScreen()` من العثور على الصورة، لذا إذا كنت بحاجة إلى العثور على مكان وجود نافذة معينة على الشاشة، فمن الأسرع والأكثر موثوقية استخدام ميزات النافذة الخاصة بوحدة `PyAutoGUI`.

> **ملاحظة**: تعمل ميزات النافذة الخاصة بوحدة PyAutoGUI على نظام تشغيل ويندوز فقط، ولا تعمل على نظام تشغيل ماك أو لينكس ابتداءً من الإصدار 0.9.46، وتأتي هذه الميزات من احتواء الوحدة PyAutoGUI على الوحدة PyGetWindow.

20.9.1 الحصول على النافذة النشطة

النافذة النشطة على شاشتك هي النافذة الموجودة حاليًا في المقدمة والتي تقبل الإدخال من لوحة المفاتيح. إذا كنت تكتب حاليًا شيفرة برمجية في المحرّر Mu، فإن نافذته هي النافذة النشطة، حيث ستُنشَّط نافذة واحدة فقط من بين جميع النوافذ التي تظهر على شاشتك في كل مرة. استدعِ الدالة `pyautogui.getActiveWindow()` في الصدفة التفاعلية للحصول على كائن `Window` أو كائن `Win32Window` عند التشغيل على نظام ويندوز.

يمكنك استرداد أيٍّ من سمات الكائن `Window` التي تمثّل حجمه وموضعه وعنوانه بعد الحصول عليه وهذه السمات هي:

- `left` و `right` و `top` و `bottom`: عدد صحيح واحد يمثّل الإحداثي x أو y لطرف النافذة.
- `topleft` و `topright` و `bottomleft` و `bottomright`: مجموعة مسماة مكونة من عددين صحيحين يمثّلان إحداثيات (x, y) لزاوية النافذة.
- `midleft` و `midright` و `midleft` و `midright`: مجموعة مسماة مكونة من عددين صحيحين يمثلان إحداثيات (x, y) لمنتصف طرف النافذة.
- `width` و `height`: عدد صحيح واحد يمثّل أحد أبعاد النافذة بالبكسل.
- `size`: مجموعة مسماة مكونة من عددين صحيحين يمثّلان عرض وارتفاع النافذة (width, height).
- `area`: عدد صحيح واحد يمثل مساحة النافذة بالبكسل.
- `center`: مجموعة مسماة مكونة من عددين صحيحين يمثلان إحداثيات (x, y) لمركز النافذة.
- `centerx` و `centery` : عدد صحيح واحد يمثل إحداثي x أو y لمركز النافذة.
- `box`: مجموعة مسماة مكونة من أربعة أعداد صحيحة لقياسات يسار وأعلى وعرض وارتفاع النافذة `(left, top, width, height)`.
- `title`: سلسلة من النص الموجود في شريط العنوان أعلى النافذة.

أدخِل مثلًا ما يلي في الصدفة التفاعلية من أجل الحصول على معلومات موضع النافذة وحجمها وعنوانها من كائن `Window`:

```
>>> import pyautogui
>>> fw = pyautogui.getActiveWindow()
>>> fw
Win32Window(hWnd=2034368)
>>> str(fw)
'<Win32Window left="500", top="300", width="2070", height="1208",
title="Mu 1.0.1 - test1.py">'
>>> fw.title
'Mu 1.0.1 - test1.py'
>>> fw.size
(2070, 1208)
>>> fw.left, fw.top, fw.right, fw.bottom
(500, 300, 2070, 1208)
>>> fw.topleft
(256, 144)
```

```
>>> fw.area
2500560
>>> pyautogui.click(fw.left + 10, fw.top + 20)
```

يمكنك الآن استخدام هذه السمات لحساب الإحداثيات الدقيقة في النافذة، فمثلًا إذا كنت تعلم أن الزر الذي تريد النقر عليه يقع دائمًا على بعد 10 بكسلات على اليمين و20 بكسلًا للأسفل من الزاوية العلوية اليسرى للنافذة، وأن الزاوية العلوية اليسرى للنافذة تقع عند إحداثيات الشاشة (500, 300)، فسيؤدي استدعاء التابع `pyautogui.click(310, 520)` (أو `pyautogui.click(fw.left + 10, fw.top + 20)` إذا احتوى المتغير `fw` على كائن `Window` الخاص بالنافذة) إلى النقر على هذا الزر، وبالتالي لن تضطر إلى الاعتماد على الدالة `locateOnScreen()` الأبطأ والأقل موثوقية للعثور على الزر.

20.9.2 طرق أخرى للحصول على النافذة

تُعَد الدالة `getActiveWindow()` مفيدةً للحصول على النافذة النشطة في وقت استدعاء الدالة، ولكن ستحتاج إلى استخدام بعض الدوال الأخرى للحصول على كائنات `Window` للنوافذ الأخرى على الشاشة، حيث تعيد الدوال الأربع التالية قائمةً بكائنات `Window`، وإن لم تتمكّن من العثور على أيّ نوافذ، فستعيد قائمةً فارغة:

- `pyautogui.getAllWindows()`: تعيد قائمةً بكائنات `Window` لكل نافذة مرئية على الشاشة.
- `pyautogui.getWindowsAt(x, y)`: تعيد قائمةً بكائنات `Window` لكل نافذة مرئية تتضمن النقطة (x, y).
- `pyautogui.getWindowsWithTitle(title)`: تعيد قائمةً بكائنات `Window` لكل نافذة مرئية تتضمن السلسلة النصية `title` في شريط العنوان الخاص بها.
- `pyautogui.getActiveWindow()`: تعيد كائن `Window` للنافذة التي تتلقّى تركيز لوحة المفاتيح في الوقت الحالي.
- تحتوي الوحدة `PyAutoGUI` أيضًا على الدالة `pyautogui.getAllTitles()` التي تعيد قائمةً بالسلاسل النصية لكل نافذة مرئية.

20.9.3 معالجة النوافذ

يمكن لسمات النوافذ أن تفعل أكثر من مجرد إخبارك بحجم النافذة وموضعها، إذ يمكنك أيضًا ضبط قيمها لتغيير حجم النافذة أو تحريكها، فمثلًا أدخِل ما يلي في الصدفة التفاعلية:

```
>>> import pyautogui
>>> fw = pyautogui.getActiveWindow()
❶ >>> fw.width # الحصول على العرض الحالي للنافذة
1669
❷ >>> fw.topleft # الحصول على الموضع الحالي للنافذة
(174, 153)
❸ >>> fw.width = 1000 # تغيير حجم العرض
❹ >>> fw.topleft = (800, 400) # تحريك النافذة
```

نستخدم أولًا سمات كائن `Window` لمعرفة معلومات حول حجم النافـذة ❶ وموضـعها ❷. يجب أن تتحـرك النافذة ❹ وتصبح أضيق ❸ كما في الشكل التالي بعد استدعاء هذه الدوال في المحرّر Mu:

الشكل 112: نافذة المحرّر Mu قبل وبعد باستخدام سمات كائن Window

يمكنك أيضًا اكتشاف وتغيير حالات تصغير النافذة وتكبيرها وتنشيطها، لذا جرّب إدخال ما يلي ضمن الصدفة التفاعلية:

```
>>> import pyautogui
>>> fw = pyautogui.getActiveWindow()
❶ >>> fw.isMaximized # تعيد القيمة True عند تكبير النافذة
False
❷ >>> fw.isMinimized # تعيد القيمة True عند تصغير النافذة
False
❸ >>> fw.isActive # تعيد القيمة True إذا كانت النافذة نشطة
True
❹ >>> fw.maximize() # تكبير النافذة
>>> fw.isMaximized
True
❺ >>> fw.restore() # التراجع عن إجراء التصغير/التكبير
❻ >>> fw.minimize() # تصغير النافذة
>>> import time
>>> # الانتظار 5 ثوانٍ أثناء تنشيط نافذة مختلفة:
❼ >>> time.sleep(5); fw.activate()
❽ >>> fw.close() # سيؤدي ذلك إلى إغلاق النافذة التي تكتب فيها
```

تحتوي السمات `isMaximized` ❶ و `isMinimized` ❷ و `isActive` ❸ على قيمٍ منطقية تشير إلى ما إذا كانت النافذة حاليًا في حالة التكبير أو التصغير أو التنشيط أم لا، بينما تغير التوابع `maximize()` ❹ و`minimize()` ❻ و `activate()` ❼ و `restore()` ❺ حالة النافذة، وسيعيد التابع `restore()` النافذة إلى حجمها وموضعها السابق بعد تكبير النافذة أو تصغيرها باستخدام التابعين `maximize()` و `minimize()`.

يغلق التابع `close()` النافذة، ولكن كن حذرًا عند استخدام هذا التابع، لأنه قد يتجاوز أي مربعات حوار للرسائل التي تطلب منك حفظ عملك قبل الخروج من التطبيق.

يمكن العثور على التوثيق الكامل لميزة التحكم في النوافذ الخاصة بوحدة PyAutoGUI على موقعها الرسمي. يمكنك أيضًا استخدام هذه الميزات بصورة منفصلة عن الوحدة PyAutoGUI مع الوحدة PyGetWindow الذي يمكنك الاطلاع على توثيقها على موقعها الرسمي.

20.10 التحكم بلوحة المفاتيح

تحتوي الوحدة PyAutoGUI أيضًا على دوال لإرسال ضغطات مفاتيح افتراضية إلى حاسوبك، والتي تمكّنك من ملء الاستمارات أو إدخال نصٍ في التطبيقات.

20.10.1 إرسال سلسلة نصية من لوحة المفاتيح

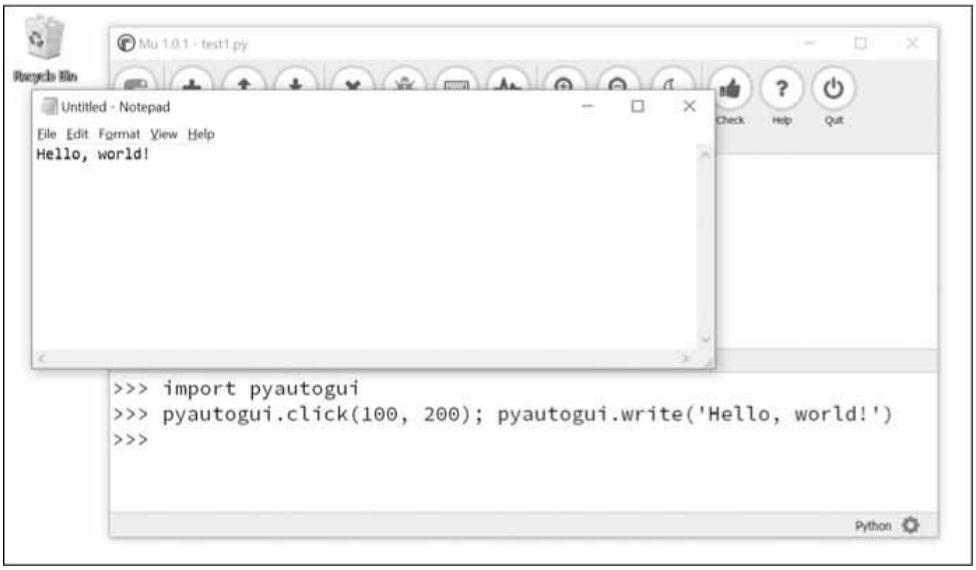
ترسل الدالة pyautogui.write() ضغطات مفاتيح افتراضية إلى الحاسوب، حيث يعتمد ما تفعله هذه الضغطات على النافذة النشطة والحقل النصي المُركَّز عليه، لذا قد ترغب أولًا في إرسال نقرة بالفأرة إلى الحقل النصي الذي تريده للتأكد من التركيز عليه.

لنستخدم لغة بايثون لكتابة النص "Hello, world!" في نافذة محرر الملفات. افتح أولًا نافذة جديدة في محرّر ملفاتك وَضَعها في الزاوية العلوية اليسرى من شاشتك بحيث تنقر وحدة PyAutoGUI في المكان المناسب للتركيز عليها، ثم أدخِل ما يلي في الصدفة التفاعلية:

```
>>> pyautogui.click(100, 200); pyautogui.write('Hello, world!')
```

لاحظ كيف أننا وضعنا أمرين على السطر نفسه، وفصلنا بينهما بفاصلة منقوطة، مما يمنع الصدفة التفاعلية من مطالبتك بالإدخال بين تشغيل هاتين التعليمتين، ويمنعك من التركيز على نافذة جديدة عن طريق الخطأ بين الاستدعائين click() و write() الذي يمكن أن يفسد مثالنا.

سترسل شيفرة بايثون أولًا نقرة افتراضية بالفأرة إلى الإحداثيات (100, 200)، والتي يجب أن تنقر على نافذة محرّر الملفات لنقل التركيز إليها، وسيرسل استدعاء الدالة write() النص "Hello, world!" إلى النافذة، مما يجعلها تبدو كما في الشكل التالي، وبهذا تكون قد أصبحت لديك الآن شيفرة برمجية يمكن أن تكتب نيابةً عنك.



الشكل 113: استخدام وحدة PyAutogGUI للنقر على نافذة محرّر الملفات وكتابة النص فيها

ستكتب الدالة `write()` افتراضيًا السلسلة النصية الكاملة مباشرةً، ولكن يمكنك تمرير وسيطٍ ثانٍ اختياري لإضافة توقف قصير بين كل محرف والآخر، وهذا الوسيط هو عدد صحيح أو عشري يمثل عدد الثواني للتوقف مؤقتًا، فمثلًا سينتظر الاستدعاء `pyautogui.write('Hello, world!', 0.25)` ربع ثانية بعد كتابة الحرف H، وربع ثانية أخرى بعد كتابة الحرف e، وإلخ.

قد يكون تأثير الآلة الكاتبة التدريجي مفيدًا للتطبيقات الأبطأ التي لا يمكنها معالجة ضغطات المفاتيح بسرعة كافية للتماشي مع الوحدة `PyAutoGUI`.

ستحاكي أيضًا الوحدة `PyAutoGUI` الضغط باستمرار على مفتاح `SHIFT` آليًا بالنسبة للمحارف A أو ! .

20.10.2 أسماء المفاتيح

لا يُعَد تمثيل كافة المفاتيح باستخدام محارف نصية مفردة أمرًا سهلًا مثل تمثيل المفتاح SHIFT أو مفتاح السهم الأيسر بمحرف واحد، لذا تمثل وحدة `PyAutoGUI` مفاتيح لوحة المفاتيح هذه بقيم سلاسل نصية قصيرة، فمثلًا نمثّل مفتاح ESC باستخدام السلسلة النصية `'esc'` ونمثّل مفتاح ENTER باستخدام السلسلة النصية `'enter'`.

يمكن تمرير قائمة بالسلاسل النصية لهذه المفاتيح إلى الدالة `write()` بدلًا من تمرير وسيط سلسلة نصية واحدة، فمثلًا يضغط الاستدعاء التالي على المفتاح A ثم على المفتاح B ثم على مفتاح السهم الأيسر مرتين ويضغط أخيرًا على المفتاحين X و Y:

```
>>> pyautogui.write(['a', 'b', 'left', 'left', 'X', 'Y'])
```

يؤدي الضغط على مفتاح السهم الأيسر إلى تحريك مؤشر لوحة المفاتيح، لذا سينتج عن ذلك الخـرج XYab. يوضّح الجدول الآتي السلاسـل النصـية لمفـاتيح لوحـة المفـاتيح الخاصـة بوحـدة PyAutoGUI، والـتي يمكنـك تمريرها إلى الدالة `write()` لمحاكاة الضغط على أيّ مجموعة من المفاتيح.

يمكنك أيضًا الاطلاع على قائمة `pyautogui.KEYBOARD_KEYS` لرؤية جميع السلاسل النصـية لمفـاتيح لوحة المفاتيح المحتملة التي ستقبلها وحدة PyAutoGUI. تشير السلسلة النصية `'shift'` إلى مفتاح SHIFT الأيسر وهي تكافئ السلسـلة النصـية `'shiftleft'`، وينطبـق الأمـر نفسـه على السلاسـل النصـية `'ctrl'` و`'alt'` و `'win'` التي تشير جميعها إلى مفتاح الجهة اليسرى من لوحة المفاتيح.

يوضح الجدول الآتي قيم سمات `PyKeyboard`:

السلسلة النصية لمفتاح لوحة المفاتيح	معناها
`'a'` و `'b'` و `'c'` و `'A'` و `'B'` و `'C'` و `'1'` و `'2'` و `'3'` و `'!'` و `'@'` و `'#'` وإلخ	مفاتيح المحارف المفردة
`'enter'` (أو `'return'` أو `'\n'`)	مفتاح ENTER
`'esc'`	مفتاح ESC
`'shiftleft'` و `'shiftright'`	مفتاحا SHIFT الأيسر والأيمن
`'altleft'` و `'altright'`	مفتاحا ALT الأيسر والأيمن
`'ctrlleft'` و `'ctrlright'`	مفتاحا CTRL الأيسر والأيمن
`'tab'` (أو `'\t'`)	مفتاح TAB
`'backspace'` و `'delete'`	مفتاح BACKSPACE ومفتاح DELETE
`'pageup'` و `'pagedown'`	مفتاح PAGE UP ومفتاح PAGE DOWN
`'home'` و `'end'`	مفتاح HOME ومفتاح END
`'up'` و `'down'` و `'left'` و `'right'`	مفاتيح الأسهم للأعلى وللأسفل وإلى اليسار وإلى اليمين
`'f1'` و `'f2'` و `'f3'` وإلخ	المفاتيح من F1 إلى F12
`'volumemute'` و `'volumedown'` و `'volumeup'`	مفاتيح كتم الصوت وخفض مستوى الصوت ورفع مستوى الصوت. لا تحتوي بعض لوحات المفاتيح على هذه المفاتيح، ولكن سيبقى نظام تشغيل حاسوبك قادرًا على فهم محاكاة هذه الضغطات للمفاتيح
`'pause'`	مفتاح PAUSE
`'capslock'` و `'numlock'` و `'scrolllock'`	مفتاح CAPS LOCK ومفتاح NUM LOCK ومفتاح SCROLL LOCK
`'insert'`	مفتاح INS أو INSERT

`'printscreen'`	مفتاح PRTSC أو PRINT SCREEN
`'winleft'` و `'winright'`	مفتاح WIN الأيسر والأيمن على نظام ويندوز
`'command'`	مفتاح Command على نظام ماك ⌘
`'option'`	مفتاح OPTION على نظام ماك

الجدول 28: قيم سمات PyKeyboard

20.10.3 الضغط على لوحة المفاتيح وتحريرها

ترسل الدالتان `pyautogui.keyDown()` و `pyautogui.keyUp()` ضغطات مفاتيح افتراضية وتحريرها إلى الحاسوب مثل الدالتين `mouseDown()` و `mouseUp()`، ونمرّر إلى هاتين الدالتين سلسلة نصية لمفاتيح لوحة المفاتيح (اطّلع على الجدول السابق) كوسيطٍ لها. توفّر وحدة `PyAutoGUI` الدالة `pyautogui.press()` التي تستدعي هاتين الدالتين لمحاكاة ضغطة كاملة على المفاتيح.

شغّل الشيفرة البرمجية التالية التي ستكتب محرف إشارة الدولار $ الذي نحصل عليه من خلال الضغط على مفتاح SHIFT مع الضغط على الرقم 4:

```
>>> pyautogui.keyDown('shift'); pyautogui.press('4');
pyautogui.keyUp('shift')
```

تضغط التعليمة السابقة على مفتاح SHIFT، ثم تضغط على (وتحرر) الرقم 4، ثم تحرّر مفتاح SHIFT. إذا أردتَ كتابة سلسلة نصية في حقل نصي، فستكون الدالة `write()` أكثر ملاءمة، ولكن ستكون الدالة `press()` الطريقة الأبسط بالنسبة للتطبيقات التي تأخذ أوامرًا ذات مفتاح واحد.

20.10.4 مجموعات مفاتيح التشغيل السريع Hotkey أو الاختصارات

مفاتيح التشغيل السريع أو الاختصارات هي مجموعة من الضغطات على المفاتيح لاستدعاء بعض وظائف التطبيق، فمفتاح التشغيل السريع الشائع لنسخ تحديدٍ مثلًا هو `CTRL-C` في نظامي تشغيل ويندوز ولينكس لينكس أو `⌘-C` في نظام تشغيل ماك.

يضغط المستخدم مع الاستمرار على مفتاح CTRL، ثم يضغط على المفتاح C، ثم يحرّر المفتاحين C وCTRL، حيث يمكننا تطبيق ذلك باستخدام الدالتين `keyDown()` و `keyUp()` الخاصتين بالوحدة `PyAutoGUI` من خلال إدخال ما يلي:

```
pyautogui.keyDown('ctrl')
pyautogui.keyDown('c')
pyautogui.keyUp('c')
pyautogui.keyUp('ctrl')
```

يُعَد ذلك معقدًا إلى حدٍ ما، لذا استخدم الدالة `pyautogui.hotkey()` بدلًا من ذلك، حيث تأخذ هذه الدالة عدة وسطاء تمثّل السلسلة النصية لمفاتيح لوحة المفاتيح، وتضغط عليها بالترتيب، ثم تحررها بالترتيب العكسي، إذ ستكون الشيفرة البرمجية الخاصية بمثال `CTRL-C` ببساطة كما يلي:

```
pyautogui.hotkey('ctrl', 'c')
```

تُعَد هذه الدالة مفيدة خاصةً لمجموعات مفاتيح التشغيل السريع الأكبر حجمًا، فمثلًا تعرض مجموعة مفاتيح التشغيل السريع `Ctrl-Alt-Shift-S` لوحة الأنماط Style في برنامج وورد Word، حيث يمكنك استخدام الاستدعاء `hotkey('ctrl', 'alt', 'shift', 's')` فقط بدلًا من إجراء ثمانية استدعاءات لدوال مختلفة (أربعة استدعاءات للدالة `keyDown()` وأربعة استدعاءات للدالة `keyUp()`).

20.11 إعداد سكربتات أتمتة واجهة المستخدم الرسومية

تُعَد سكربتات أتمتة واجهة المستخدم الرسومية طريقةً رائعة لأتمتة المهام المملة، ولكنها قد تكون صعبة التحقيق، حيث إذا كان هناك نافذة في مكان خاطئ على سطح المكتب أو ظهرت بعض النوافذ المنبثقة بطريقة غير متوقعة، فقد ينقر السكربت الخاص بك على الأشياء الخاطئة على الشاشة، لذا إليك بعض النصائح لإعداد سكربتات أتمتة واجهة المستخدم الرسومية:

- استخدم دقة الشاشة نفسها في كل مرة تشغّل فيها السكربت حتى لا يتغير موضع النوافذ.
- يجب تكبير نافذة التطبيق التي ينقر عليها السكربت الخاص بك بحيث تكون الأزرار والقوائم في المكان نفسه في كل مرة تشغّل فيها السكربت.
- أضِف فترات توقف كافية أثناء انتظار تحميل المحتوى، إذ لا تريد أن يبدأ السكربت بالنقر قبل أن يصبح التطبيق جاهزًا.
- استخدم الدالة `locateOnScreen()` للعثور على الأزرار والقوائم التي يمكنك النقر عليها بدلًا من الاعتماد على إحداثيات XY. إن لم يتمكّن السكربت الخاص بك من العثور على الشيء الذي يريد النقر عليه، فأوقِف البرنامج بدلًا من السماح له بمواصلة النقر عشوائيًا.
- استخدم الدالة `getWindowsWithTitle()` للتأكّد من وجود نافذة التطبيق التي تعتقد أن السكربت الخاص بك ينقر عليها، واستخدم التابع `activate()` لوضع تلك النافذة في المقدمة.
- استخدم الوحدة `logging` التي تحدثنا عنها في الفصل الحادي عشر للاحتفاظ بملفٍ يسجّل ما فعله السكربت الخاص بك، وبالتالي إذا أوقفتَ السكربت في منتصف العملية، فيمكنك تعديله للمتابعة من مكان توقف هذا السكربت.

- أضِف أكبر عدد ممكن من عمليات التحقق إلى السكربت الخاص بك، فمثلًا يمكن أن يفشل السكربت إذا ظهرت نافذة منبثقة غير متوقعة أو إذا فقدَ حاسوبك اتصاله بالإنترنت.
- قد ترغب في الإشراف على السكربت عندما يبدأ لأول مرة للتأكد من أنه يعمل بصورة صحيحة.

قد ترغب أيضًا في التوقف مؤقتًا في بداية السكربت الخاص بك حتى يتمكّن المستخدم من إعداد النافذة التي سينقر عليها السكربت، حيث تحتوي وحدة `PyAutoGUI` على الدالة `sleep()` التي تعمل بطريقة مماثلة للدالة `time.sleep()`، ولكنها توفّر عليك الاضطرار إلى إضافة التعليمة `import time` إلى السكربتات الخاصة بك، وتوجد أيضًا الدالة `countdown()` التي تطبع أرقام العد التنازلي لمنح المستخدم إشارة مرئية إلى أن السكربت سيستأنف عمله قريبًا.

أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import pyautogui
>>> pyautogui.sleep(3) # إيقاف البرنامج مؤقتًا لمدة 3 ثوانٍ
>>> pyautogui.countdown(10) # العد التنازلي لمدة 10 ثوانٍ
10 9 8 7 6 5 4 3 2 1
>>> print('Starting in ', end=''); pyautogui.countdown(3)
Starting in 3 2 1
```

يمكن أن تساعد هذه النصائح في جعل سكربتات أتمتة واجهة المستخدم الرسومية أسهل في الاستخدام وأكثر قدرة على الاستعادة من الظروف غير المتوقعة.

20.12 مراجعة لدوال وحدة PyAutoGUI

يغطي هذا الفصل العديد من الدوال المختلفة، لذا سنوضّح فيما يلي مرجعًا موجزًا سريعًا لهذه الدوال:

- `moveTo(x, y)`: تحرك مؤشر الفأرة إلى إحداثيات `x` و `y` المُحدَّدة.
- `move(xOffset, yOffset)`: تحرك مؤشر الفأرة بالنسبة إلى موضعه الحالي.
- `dragTo(x, y)`: تحرك مؤشر الفأرة أثناء الضغط المستمر على زر الفأرة الأيسر.
- `drag(xOffset, yOffset)`: تحرّك مؤشر الفأرة نسبة إلى موضعه الحالي أثناء الضغط المستمر على زر الفأرة الأيسر.
- `click(x, y, button)`: تحاكي النقر (بزر الفأرة الأيسر افتراضيًا).
- `rightClick()`: تحاكي النقر بالزر الأيمن.

- `middleClick()`: تحاكي النقر بالزر الأوسط.
- `doubleClick()`: تحاكي النقر المزدوج على الزر الأيسر.
- `mouseDown(x, y, button)`: تحاكي الضغط على الزر المُحدَّد في الموضع `x, y`.
- `mouseUp(x, y, button)`: تحاكي تحرير الزر المُحدَّد في الموضع `x, y`.
- `scroll(units)`: تحاكي عجلة التمرير في الفأرة، حيث نمرّر وسيطًا موجبًا للتمرير للأعلى، ونمرّر وسيطًا سالبًا للتمرير للأسفل.
- `write(message)`: تكتب المحارف الموجودة في سلسلة الرسالة النصية المُحدَّدة.
- `write([key1, key2, key3])`: تكتب سلاسل نصية لمفاتيح لوحة المفاتيح المُحدَّدة.
- `press(key)`: تضغط على السلسلة النصية لمفتاحٍ محدَّد من لوحة المفاتيح.
- `keyDown(key)`: تحاكي الضغط على مفتاح لوحة المفاتيح المُحدَّد.
- `keyUp(key)`: تحاكي تحرير مفتاح لوحة المفاتيح المُحدَّد.
- `hotkey([key1, key2, key3])`: تحاكي الضغط على السلاسل النصية لمفاتيح لوحة المفاتيح المُحدَّدة بالترتيب ثم تحرّرها بترتيب عكسي.
- `screenshot()`: تعيد لقطة الشاشة بوصفها كائن `Image`. اطّلع على الفصل السابق للحصول على معلومات حول كائنات `Image`.
- `getActiveWindow()` و `getAllWindows()` و `getWindowsAt()` و `getWindowsWithTitle()`: تعيد هذه الدوال كائنات `Window` التي يمكنها تغيير حجم نوافذ التطبيقات وإعادة تموضعها على سطح المكتب.
- `getAllTitles()`: تعيد قائمةً بالسلاسل النصية لشريط عنوان كلّ نافذةٍ على سطح المكتب.

20.13 اختبارات كابتشا Captcha وأخلاقيات استخدام الحواسيب

تُعَد اختبارات كابتشا Captcha اختصارًا للعبارة الإنجليزية "Completely Automated Public Turing test to tell Computers and Humans Apart" أو "اختبار تورينج العام الآلي بالكامل للتمييز بين الحواسيب والبشر"، وهي الاختبارات الصغيرة التي تطلب منك كتابة حروف موجودة في صورة غير واضحة أو النقر على صور صنابير إطفاء الحرائق مثلًا.

يسهُل على البشر اجتياز هذه الاختبارات، ولكن يكاد يكون من المستحيل على البرامج حلها بالرغم من أننا نجدها مزعجة. يمكنك أن ترى بعد قراءة هذا الفصل مدى سهولة كتابة سكربت يمكنه التسجيل في مليارات

حسابات البريد الإلكتروني المجانية مثلًا أو إغراق المستخدمين برسائل مزعجة، لذا تعمل اختبارات كابتشا على تخفيف ذلك من خلال المطالبة بخطوة لا يمكن إلا للبشر اجتيازها.

لا تطبق جميع مواقع الويب اختبارات كابتشا، وقد تكون عرضةً لإساءة الاستخدام من المبرمجين غير الأخلاقيين، إذ يُعَد تعلّم البرمجة مهارة قوية ومهمة، ولكن قد تميل إلى إساءة استخدام هذه القوة لتحقيق مكاسب شخصية أو حتى لمجرد التفاخر، فلا يُعَد الباب المفتوح مبررًا للتعدّي على ممتلكات الآخرين، لذا تقع مسؤولية برامجك على عاتقك الشخصي بوصفك مبرمجًا.

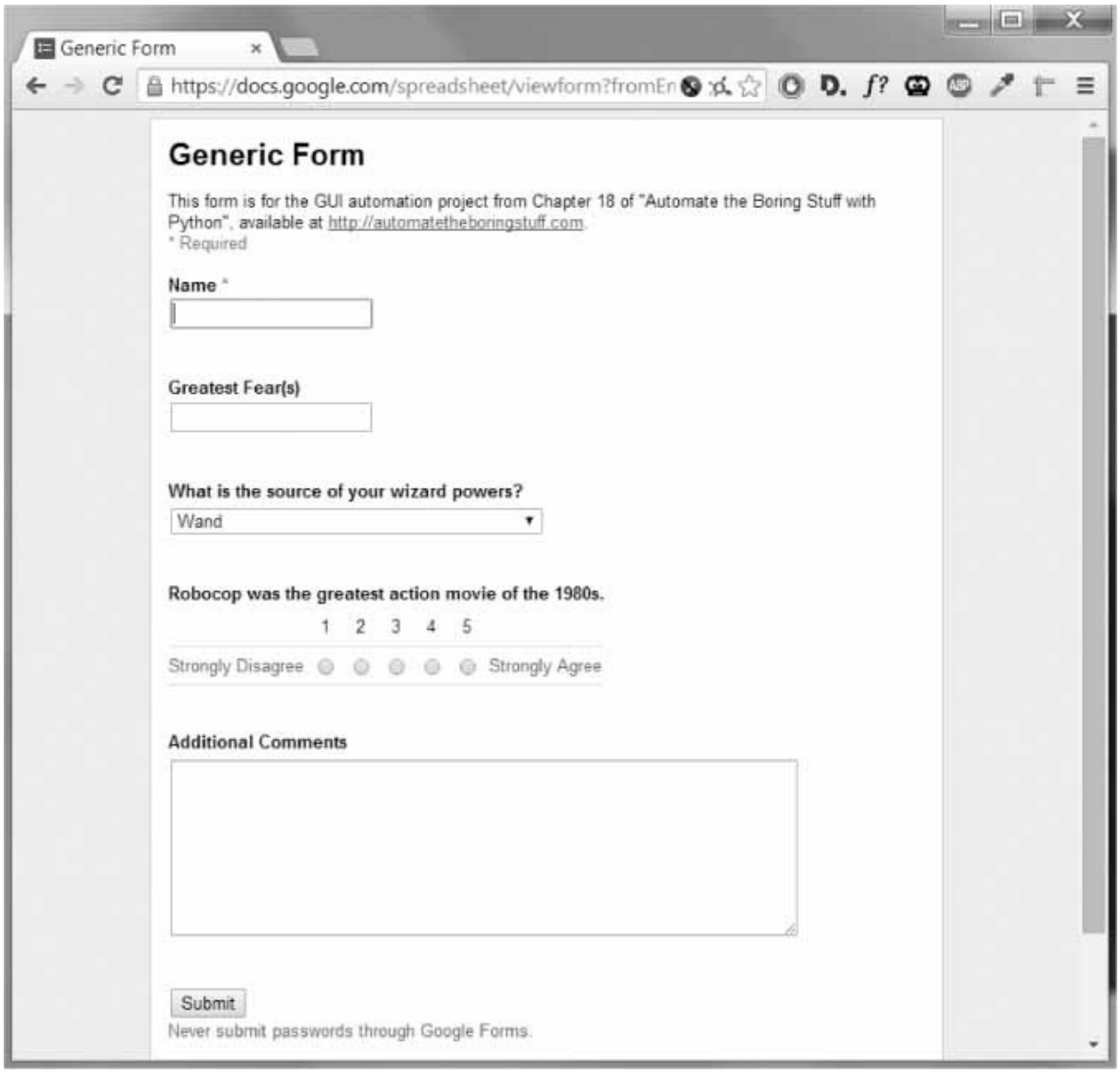
لا يُعَد التحايل على الأنظمة لإحداث ضررٍ أو انتهاك الخصوصية أو الحصول على ميزة غير عادلة معيارًا للذكاء، لذا نأمل أن تركز على عملك دون الضرر بالآخرين.

20.14 تطبيق عملي: ملء الاستمارات آليًا

يُعَد ملء الاستمارات مهمةً روتينية مملةً جدًا، إذًا لنفترض مثلًا أن لديك كمية هائلة من البيانات في جدول بيانات، ويجب أن تعيد كتابتها في واجهة استمارة تطبيق آخر دون وجود شخص آخر لإنجاز ذلك نيابةً عنك.

تحتوي بعض التطبيقات على ميزة الاستيراد التي تسمح لك برفع جدول بيانات يحتوي على المعلومات، ولكن قد لا توجد طريقة أخرى سوى النقر والكتابة دون اهتمام لساعات متواصلة في بعض الأحيان، لذا لنحاول إيجاد وسيلة لأتمتة هذه المهمة المملة.

استمارة هذا المشروع هي استمارة على مستندات جوجل Google Docs، والتي يمكنك العثور عليها على autbor.com، وتبدو كما يلي:



الشكل 114: الاستمارة المستخدمَة في هذا المشروع

إليك الخطوات العامة لما يجب أن يفعله برنامجك:

1. النقر على الحقل النصي الأول من الاستمارة.
2. التنقل عبر الاستمارة، وكتابة المعلومات في كل حقل.
3. النقر على زر الإرسال Submit.
4. تكرار العملية مع المجموعة التالية من البيانات.

وبالتالي يجب أن تطبّق شيفرتك البرمجية الخطوات التالية:

1. استدعاء الدالة `pyautogui.click()` للنقر على الاستمارة وزر الإرسال Submit.

2. استدعاء الدالة `pyautogui.write()` لإدخال النص في الحقول.
3. معالجة الاستثناء `KeyboardInterrupt` حتى يتمكّن المستخدم من الضغط على الاختصار `CTRL-C` من أجل الإنهاء.

افتح نافذة جديدة في محرّرك لإنشاء ملف جديد واحفظه بالاسم `formFiller.py`.

20.14.1 الخطوة الأولى: معرفة الخطوات ملء الاستمارة

يجب أولًا معرفة ضغطات المفاتيح ونقرات الفأرة التي ستملأ الاستمارة قبل كتابة الشيفرة البرمجيـة. يمكن أن يساعدك التطبيق الذي يُشغّله اسـتدعاء الدالـة `pyautogui.mouseInfo()` في معرفـة إحـداثيات الفـأرة المُحدَّدة، ويجب معرفة إحداثيات الحقل النصي الأول فقط، ثم يمكنك الضـغط على مفتـاح TAB لنقـل التركـيز إلى الحقل التالي بعد النقر على الحقل الأول، مما يوفّر عليك الاضطرار إلى معرفة إحداثيات x و y من أجل النقـر على كل حقل.

إليك خطوات إدخال البيانات في الاستمارة:

1. انقل تركيز لوحة المفاتيح على حقل الاسم Name بحيث يؤدي الضغط على المفاتيح إلى كتابة نص في الحقل.
2. اكتب اسمًا، ثم اضغط على مفتاح TAB.
3. اكتب خوفك الأكبر في الحقل Greatest Fear ثم اضغط على مفتاح TAB.
4. اضغط على مفتاح السهم للأسفل عـددًا صـحيحًا من المـرات لتحديـد مصـدر قـواك السـحرية Wizard Power Source، إذ ستضغط مرة واحدة لخيار العصا السحرية wand ومرتين لخيار التعويذة amulet وثلاث مرات لخيار الكرة البلورية crystal ball وأربع مرات لخيار المال money، ثم اضغط على مفتـاح TAB. لاحظ أنه يجب أن تضغط على مفتاح السهم للأسفل مرة أخرى لكل خيار في نظام ماك macOS، وقد تحتاج إلى الضغط على مفتاح ENTER أيضًا بالنسبة لبعض المتصفحات.
5. اضغط على مفتاح السهم الأيمن لتحديد إجابة سؤال RoboCop، واضغط عليه مـرة واحـدة للخيـار 2 أو مرتين للخيار 3 أو ثلاث مرات للخيار 4 أو أربع مـرات للخيـار 5 أو اضـغط على مفتـاح المسـافة لتحديـد الخيار 1 المُحدَّد افتراضيًا، ثم اضغط على مفتاح TAB.
6. اكتب تعليقًا إضافيًا في الحقل Additional Comments، ثم اضغط على مفتاح TAB.
7. اضغط على مفتاح ENTER للنقر على زر الإرسال Submit.
8. سينقلك المتصفح إلى صفحة أخرى بعد إرسـال الاسـتمارة، حيث يجب اتبـاع رابـط في هـذه الصـفحة للعودة إلى صفحة الاستمارة.

قد تعمل المتصفحات الأخرى على أنظمة تشغيل مختلفة بطريقة مختلفة قليلًا عن الخطوات التي ذكرناها، لذا تأكد من أن هذه المجموعات من ضغطات المفاتيح تعمل على حاسوبك قبل تشغيل البرنامج.

20.14.2 الخطوة الثانية: إعداد الإحداثيات

حمّل مثال الاستمارة التي نزّلتها (الشكل السابق) في المتصفح من خلال الانتقال إلى autbor.com، واجعل شيفرتك البرمجية كما يلي:

```
#! python3
# formFiller.py - ملء الاستمارة آليًا
import pyautogui, time
# منح المستخدم فرصةً لإنهاء السكربت
# الانتظار حتى تحميل صفحة الاستمارة
# ملء حقل الاسم Name
# ملء حقل مخاوفك الكبرى Greatest Fear(s)
# ملء حقل مصدر قواك السحرية Source of Wizard Powers
# ملء الحقل RoboCop
# ملء حقل التعليقات الإضافية Additional Comments
# الضغط على زر الإرسال Submit
# الانتظار حتى تحميل صفحة الاستمارة
# النقر على رابط إرسال رد آخر
```

تحتاج الآن البيانات التي تريد إدخالها فعليًا في هذه الاستمارة، حيث قد تأتي هذه البيانات في العالم الحقيقي من جدول بيانات أو ملف نص عادي أو من موقع ويب، وقد تتطلب شيفرة برمجية إضافية لتحميلها في البرنامج، ولكننا سنكتب كل هذه البيانات ضمن متغير في مثالنا، لذا أضِف ما يلي إلى برنامجك:

```
#! python3
# formFiller.py - ملء الاستمارة آليًا
--snip--
formData = [{'name': 'Alice', 'fear': 'eavesdroppers', 'source':
'wand',
```

```
        'robocop': 4, 'comments': 'Tell Bob I said hi.'},
        {'name': 'Bob', 'fear': 'bees', 'source': 'amulet', 'robocop': 4,
        'comments': 'n/a'},
        {'name': 'Carol', 'fear': 'puppets', 'source': 'crystal ball',
        'robocop': 1, 'comments': 'Please take the puppets out of the
        break room.'},
        {'name': 'Alex Murphy', 'fear': 'ED-209', 'source': 'money',
        'robocop': 5, 'comments': 'Protect the innocent. Serve the public
        trust. Uphold the law.'},
              ]
--snip--
```

تحتوي القائمة formData على أربعة قواميس لأربعة أسماء مختلفة، ويحتوي كل قاموس على أسماء الحقول النصية كمفاتيحٍ له والردود كقيمٍ له. أخيرًا، نضبط المتغير PAUSE الخاص بوحدة PyAutoGUI للانتظار لمدة نصف ثانية بعد كل استدعاء دالة، ونذكّر المستخدم بالنقر على المتصفح لجعله النافذة النشطة. أضِف ما يلي إلى برنامجك بعد تعليمة إسناد قيمٍ إلى القائمة formData:

```
pyautogui.PAUSE = 0.5
print('Ensure that the browser window is active and the form is
loaded!')
```

20.14.3 الخطوة الثالثة: البدء في كتابة البيانات

سنكرّر حلقة for على كلٍّ من القواميس الموجودة في قائمة formData، ونمرّر القيم الموجودة في القاموس إلى دوال وحدة PyAutoGUI التي ستكتب فعليًا في الحقول النصية.

أضِف الشيفرة البرمجية التالية إلى برنامجك:

```
#! python3
# formFiller.py - ملء الاستمارة آليًا
--snip--
for person in formData:
  # منح المستخدم فرصةً لإنهاء السكربت
```

```
    print('>>> 5-SECOND PAUSE TO LET USER PRESS CTRL-C <<<')
❶ time.sleep(5)
--snip--
```

يحتوي السكربت على توقف مؤقت لمدة خمس ثوانٍ ❶ كميزة أمانٍ صغيرة، مما يمنح المستخدم فرصةً للضغط على Ctrl-C (أو تحريك مؤشر الفأرة إلى الزاوية العلوية اليسرى من الشاشة لرفع استثناء FailSafeException) لإيقاف تشغيل البرنامج في حالة أنه يعمل شيئًا غير متوقع. أضِف ما يلي بعد الشيفرة البرمجية التي تنتظر إعطاء الصفحة وقتًا للتحميل:

```
#! python3
# formFiller.py - ملء الاستمارة آليًا
--snip--
❶ print('Entering %s info...' % (person['name']))
❷ pyautogui.write(['\t', '\t'])
  # ملء حقل الاسم Name
❸ pyautogui.write(person['name'] + '\t')
  # ملء حقل مخاوفك الكبرى Greatest Fear(s)
❹ pyautogui.write(person['fear'] + '\t')
--snip--
```

نضيف استدعاء الدالة print() من حينٍ لآخر لعرض حالة البرنامج في نافذته الطرفية لإعلام المستخدم بما يحدث ❶.

حصلت الاستمارة على وقتها الكافي للتحميل، لذا نستدعي الدالة pyautogui.write(['\t', '\t']) للضغط على مفتاح TAB مرتين والتركيز على حقل الاسم Name ❷، ثم نستدعي الدالة write() مرة أخرى لإدخال السلسلة النصية في person['name'] ❸. نضيف المحرف '\t' إلى نهاية السلسلة النصية التي نمرّرها إلى الدالة write() لمحاكاة الضغط على مفتاح TAB، مما ينقل تركيز لوحة المفاتيح إلى الحقل التالي وهو Greatest Fear(s). يؤدي استدعاءٌ آخر للدالة write() إلى كتابة السلسلة النصية في person['fear'] ضمن هذا الحقل ثم ينتقل إلى الحقل التالي في الاستمارة ❹.

20.14.4 الخطوة الرابعة: التعامل مع قوائم التحديد وأزرار الاختيار

تُعَد القائمة المنسدلة لسؤال "القوى السحرية Wizard Powers" وأزرار الاختيار الخاصة بحقل RoboCop أصعب في التعامل من الحقول النصية، حيث يمكنك النقر على هذه الخيارات باستخدام الفأرة من خلال معرفة إحداثيات x و y لكل خيار ممكن، ولكن من الأسهل استخدام مفاتيح الأسهم في لوحة المفـاتيح لإجـراء التحديـد بدلًا من ذلك.

أضِف ما يلي إلى برنامجك:

```
#! python3
# formFiller.py - ملء الاستمارة آليًا
--snip--
  # Source of Wizard Powers ملء حقل مصدر قواك السحرية

❶ if person['source'] == 'wand':

  ❷ pyautogui.write(['down', '\t'] , 0.5)

  elif person['source'] == 'amulet':

    pyautogui.write(['down', 'down', '\t'] , 0.5)

  elif person['source'] == 'crystal ball':

    pyautogui.write(['down', 'down', 'down', '\t'] , 0.5)

  elif person['source'] == 'money':

    pyautogui.write(['down', 'down', 'down', 'down', '\t'] , 0.5)


  # RoboCop ملء الحقل

❸ if person['robocop'] == 1:

  ❹ pyautogui.write([' ', '\t'] , 0.5)

  elif person['robocop'] == 2:

    pyautogui.write(['right', '\t'] , 0.5)

  elif person['robocop'] == 3:

    pyautogui.write(['right', 'right', '\t'] , 0.5)
```

```
    elif person['robocop'] == 4:
      pyautogui.write(['right', 'right', 'right', '\t'] , 0.5)
    elif person['robocop'] == 5:
      pyautogui.write(['right', 'right', 'right', 'right', '\t'] , 0.5)
  --snip--
```

تذكر أنك كتبتَ شيفرة برمجية لمحاكاة الضغط على مفتاح TAB بعد ملء حقل المخاوف الكبرى Greatest Fear(s)، وبالتالي سيؤدي الضغط على مفتاح السهم للأسفل إلى الانتقال إلى العنصر التالي في قائمة التحديد بعد التركيز على القائمة المنسدلة. يجب أن يرسل برنامجك عددًا من ضغطات مفاتيح السهم للأسفل قبل الانتقال إلى الحقل التالي اعتمادًا على القيمة الموجودة في `person['source']`، حيث إذا كانت قيمة مفتاح `'source'` الموجودة في قاموس هذا المستخدم هي `'wand'` ❶، فسنحاكي الضغط على مفتاح السهم للأسفل مرة واحدة (لتحديد القيمة Wand) والضغط على مفتاح TAB ❷. إذا كانت القيمة الموجودة في مفتاح `'source'` هي `'amulet'`، فسنحاكي الضغط على مفتاح السهم للأسفل مرتين والضغط على مفتاح TAB، وينطبق الشيء نفسه بالنسبة للإجابات المُحتمَلة الأخرى. يضيف الوسيط `0.5` في استدعاءات الدالة `write()` توقفًا مؤقتًا لمدة نصف ثانية بين المفاتيح حتى لا يتحرك البرنامج بسرعة كبيرة في الاستمارة.

يمكن تحديد أزرار الاختيار الخاصة بسؤال RoboCop باستخدام مفاتيح الأسهم إلى اليمين، أو إذا أردتَ تحديد الخيار الأول ❸، فاضغط على على شريط المسافة فقط ❹.

20.14.5 الخطوة الخامسة: إرسال الاستمارة ثم الانتظار

يمكنك ملء حقل التعليقات الإضافية Additional Comments باستخدام الدالة `write()` من خلال تمرير `person['comments']` كوسيطٍ لها. يمكنك كتابة مفتاح `'\t'` إضافي لنقل تركيز لوحة المفاتيح إلى الحقل التالي أو إلى زر الإرسال Submit. سيؤدي استدعاء الدالة `pyautogui.press('enter')` إلى محاكاة الضغط على مفتاح ENTER وإرسال الاستمارة بعد التركيز على زر الإرسال Submit، ثم سينتظر برنامجك خمس ثوانٍ حتى تحميل الصفحة التالية.

ستحتوي الصفحة الجديدة بعد تحميلها على رابط إرسال ردٍ آخر الذي سيوجّه المتصفح إلى صفحة استمارة جديدة فارغة، حيث خزّنا إحداثيات هذا الرابط بوصفها مجموعةً في المتغير `submitAnotherLink` في الخطوة الثانية، لذا مرّر هذه الإحداثيات إلى الدالة `pyautogui.click()` للنقر على هذا الرابط.

يمكن لحلقة `for` الخارجية الخاصة بالسكربت الاستمرار إلى التكرار التالي وإدخال معلومات الشخص التالي في الاستمارة عندما تكون الاستمارة الجديدة جاهزة.

أكمِل برنامجك بإضافة الشيفرة البرمجية التالي:

```
#! python3
# formFiller.py - ملء الاستمارة آليًا
--snip--
  # Additional Comments ملء حقل التعليقات الإضافية
  pyautogui.write(person['comments'] + '\t')
  # Enter الضغط على زر الإرسال Submit من خلال الضغط على مفتاح
  time.sleep(0.5) # الانتظار لمدة 5 ثواني حتى يتم تفعيل الزر
  pyautogui.press('enter')
  # الانتظار حتى تحميل صفحة الاستمارة
  print('Submitted form.')
  time.sleep(5)
  # النقر على رابط إرسال رد آخر
  pyautogui.click(submitAnotherLink[0], submitAnotherLink[1])
```

سيُدخِل البرنامج المعلومات الخاصة بكل شخص بعد انتهاء حلقة `for` الرئيسية، حيث يوجد في مثالنا أربعة أشخاص للإدخال فقط، ولكن إذا كان لديك 4000 شخص، فستوفر كتابة برنامج لإنجاز ذلك عليك الكثير من الوقت والكتابة.

20.15 عرض مربعات الرسائل

تميل جميع البرامج التي كتبناها حتى الآن إلى استخدام خرجٍ يحتوي على نصٍ عادي باستخدام الدالة `print()` ودخلٍ يحتوي على نصٍ عادي باستخدام الدالة `input()`، ولكن ستستخدم برامج `PyAutoGUI` سطح المكتب بأكمله، إذ يُحتمَل فقدان النافذة النصية التي يعمل فيها برنامجك سواء كانت نافذة المحرّر Mu أو نافذة طرفية Terminal عندما ينقر برنامج `PyAutoGUI` الخاص بك ويتفاعل مع النوافذ الأخرى، مما يصعّب الحصول على الدخل والخرج من المستخدم عند إخفاء نوافذ المحرّر Mu أو نافذة الطرفية تحت نوافذ أخرى.

يمكن حل هذه المشكلة باستخدام وحدة `PyAutoGUI` التي تقدّم مربعات رسائل منبثقة لتقديم إشعارات للمستخدم وتلقي الدخل منه، إذ توجد أربع دوال لمربعات الرسائل وهي:

- `pyautogui.alert(text)`: تعرض النص `text` وتحتوي على زر موافقة OK واحد.
- `pyautogui.confirm(text)`: تعرض النص `text` وتحتوي على زر موافقة OK وزر إلغاء Cancel، وتعرض إما `'OK'` أو `'Cancel'` اعتمادًا على الزر الذي نقرنا عليه.

- `pyautogui.prompt(text)`: تعرض النص `text` وتحتوي على حقل نصي ليكتب المستخدم فيه، والذي تعيده كسلسلة نصية.
- `pyautogui.password(text)`: تماثل الدالة `prompt()`، ولكنها تعرض علامات نجمية على النص المُدخَل حتى يتمكّن المستخدم من إدخال معلومات حساسة مثل كلمة المرور.

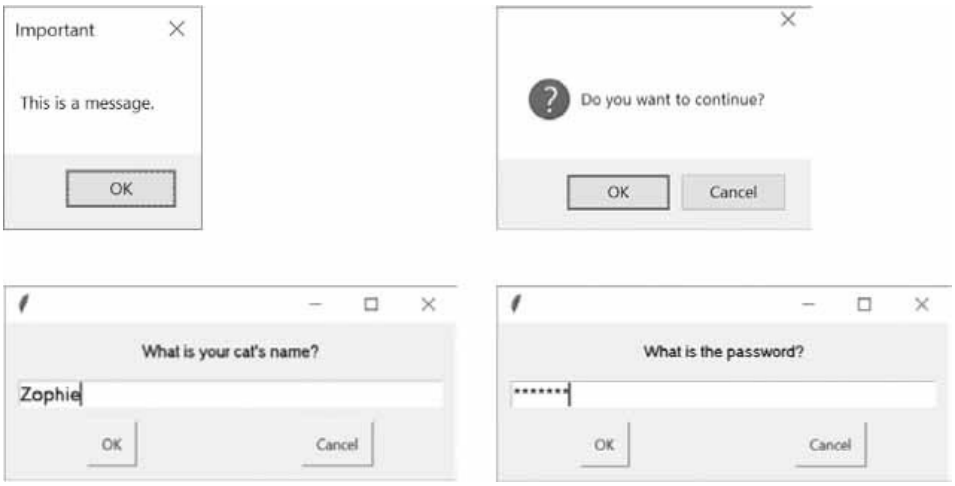
تحتوي هذه الدوال أيضًا على معاملٍ ثانٍ اختياري يقبل قيمة سلسلة نصية لاستخدامها بوصفها عنوانًا في شريط العنوان الخاص بمربع الرسالة.

لن تعود هذه الدوال حتى ينقر المستخدم على الزر الموجود عليها، لذلك يمكن استخدامها أيضًا لإدخال فترات توقف مؤقت في برامج PyAutoGUI الخاصة بك.

أدخِل مثلًا ما يلي في الصدفة التفاعلية:

```
>>> import pyautogui
>>> pyautogui.alert('This is a message.', 'Important')
'OK'
>>> pyautogui.confirm('Do you want to continue?') # اضغط على زر الإلغاء
'Cancel'
>>> pyautogui.prompt("What is your cat's name?")
'Zophie'
>>> pyautogui.password('What is the password?')
'hunter2'
```

تبدو مربعات الرسائل المنبثقة التي تنتجها السطور السابقة كما يلي:



الشكل 115: مربعات الرسائل المنبثقة

النوافـذ من أعلى اليسـار إلى أسـفل اليمين هي: النوافـذ الـتي تنشـئها الـدوال `alert()` و `confirm()` و`prompt()` و`password()`

يمكن استخدام هذه الدوال لتقديم إشعارات أو طرح أسئلة على المستخدم أثنـاء تفاعـل بـاقي البرنـامج مـع الحاسوب من خلال الفأرة ولوحة المفـاتيح. اطّلـع على التوثيـق الكامـل عـبر الإنـترنت للوحـدة `PyMsgBox` على موقعها الرسمي.

20.16 أسئلة للتدريب

1.كيف يمكنك تشغيل ميزة الفشل الآمن الخاص بالوحدة `PyAutoGUI` لإيقاف البرنامج؟

2.ما هي الدالة التي تعيد دقة Resolution الشاشة الحالية؟

3.ما هي الدالة التي تعيد إحداثيات الموضع الحالي لمؤشر الفأرة؟

4.ما هو الفرق بين الدالتين `pyautogui.moveTo()` و `pyautogui.move()`؟

5.ما هي الدوال التي يمكن استخدامها لسحب الفأرة؟

6.ما هو استدعاء الدالة الذي سيكتب محارف العبارة `"Hello, world!"`؟

7.كيف يمكنك الضغط على مفاتيح خاصة مثل مفتاح السهم إلى اليسار في لوحة المفاتيح؟

8.كيف يمكنك حفظ محتويات الشاشة الحالية في ملف صورة بالاسم `screenshot.png`؟

9.ما هي الشيفرة البرمجية التي ستضبط توقفًا مؤقتًا لمـدة ثـانيتين بعـد كـل اسـتدعاء لـدوال الوحـدة `PyAutoGUI`؟

10.إذا أردتَ أتمتـة النقـرات وضـغطات المفـاتيح في متصـفح الـويب، فهـل يجب اسـتخدام الوحـدة `PyAutoGUI` أو الأداة `Selenium`؟

11.ما الذي يجعل الوحدة `PyAutoGUI` عرضةً للخطأ؟

12.كيف يمكنك العثور على حجم النوافذ على الشاشة التي تتضمن النص `Notepad` في عنوانها؟

13.كيف يمكنك جعل متصفح فايرفوكس Firefox مثلًا نشطًا وأمام جميع النوافذ الأخرى على الشاشة؟

20.17 مشاريع للتدريب

حاول كتابة البرامج التي تؤدي المهام التي سنوضّحها فيما يلي لكسب خبرة عملية أكبر.

20.17.1 برنامج لإبقاء الحالة "مشغول" على برنامج المراسلة الفورية

تحدّد العديد من برامج المراسلة الفورية ما إذا كنت في وضع السكون أو كنت بعيدًا عن حاسوبك من خلال اكتشاف عدم وجود حركة للفأرة خلال فترة زمنية معينة مثل 10 دقائق. قد تكون بعيدًا عن حاسوبك ولكنك لا تريد أن يرى الآخرون حالة المراسلة الفورية الخاصة بك وهي في وضع السكون، لذا اكتب برنامجًا لتحريك مؤشر الفأرة قليلًا كل 10 ثوانٍ، حيث يجب أن تكون الحركة صغيرة وغير متكررة بدرجة كافية حتى لا تعترض طريقك إذا أردتَ استخدام حاسوبك أثناء تشغيل السكربت.

20.17.2 استخدام الحافظة Clipboard لقراءة حقل نصي

يمكنك إرسال ضغطات المفاتيح إلى الحقول النصية في التطبيق باستخدام الدالة `pyautogui.write()`، ولكن لا يمكنك استخدام وحدة `PyAutoGUI` وحدها لقراءة النص الموجود فعليًا ضمن الحقل النصي، لذا يمكن أن تساعد الوحدة `Pyperclip` في ذلك. استخدم الوحدة `PyAutoGUI` للحصول على نافذة محرّر نصوص مثل المحرّر Mu أو المفكرة Notepad، وإحضارها إلى مقدمة الشاشة من خلال النقر عليها، ثم النقر داخل الحقل النصي، وإرسال مفتاح التشغيل السريع `CTRL-A` أو `⌘-A` لتحديد الكل وإرسال مفتاح التشغيل السريع `Ctrl-C` أو `⌘-C` للنسخ إلى الحافظة، ويمكن لسكربت بايثون بعد ذلك قراءة نص الحافظة من خلال تشغيل التعليمتين `import pyperclip` و `pyperclip.paste()`.

اكتب برنامجًا يتبع هذا الإجراء لنسخ النص من الحقول النصية في النافذة، لذا استخدم الدالة `pyautogui.getWindowsWithTitle('Notepad')` (أو أي محرّر نصوص تختاره) للحصول على كائن `Window`. يمكن لسمات `top` و `left` لكائن `Window` أن تخبرك بمكان هذه النافذة، وسيضمن التابع `activate()` وجودها في مقدمة الشاشة. يمكنك بعد ذلك النقر على الحقل النصي الرئيسي لمحرّر النصوص من خلال إضافة 100 أو 200 بكسل مثلًا إلى قيم السمات `top` و `left` باستخدام التابع `pyautogui.click()` لنقل تركيز لوحة المفاتيح إلى هذا الحقل، ثم استدعِ الدالتين `pyautogui.hotkey('ctrl', 'a')` و `pyautogui.hotkey('ctrl', 'c')` لتحديد النص بأكمله ونسخه إلى الحافظة. أخيرًا، استدعِ الدالة `pyperclip.paste()` لاسترداد النص من الحافظة ولصقه في برنامج بايثون. يمكنك بعد ذلك استخدام هذه السلسلة النصية كما تريد، ولكن مرّرها إلى الدالة `print()` حاليًا.

لاحظ أن دوال النافذة الخاصة بوحدة `PyAutoGUI` تعمل فقط على نظام ويندوز بدءًا من الإصدار 1.0.0 من وحدة `PyAutoGUI`، ولن تعمل على نظام ماك macOS أو لينكس.

20.17.3 بوت المراسلة الفورية

تستخدم برامج المراسلة الفورية بروتوكولاتٍ خاصة تصعّب كتابة وحدات بايثون التي يمكنها التفاعل مع هذه البرامج، ولكن لا يمكن لهذه البروتوكولات الخاصة منعك من كتابة أداة أتمتة لواجهة المستخدم الرسومية.

يحتوي تطبيق واتس أب Whatsapp على شريط بحث يتيح لك إدخال اسم مستخدم في قائمة أصدقائك وفتح نافذة مراسلة عند الضغط على مفتاح ENTER، حيث ينتقل تركيز لوحة المفاتيح إلى النافذة الجديدة آليًا، وتمتلك تطبيقات المراسلة الفورية الأخرى طرقًا مشابهة لفتح نوافذ الرسائل الجديدة. اكتب برنامجًا يرسل رسالة إشعار آليًا إلى مجموعة مختارة من الأشخاص في قائمة أصدقائك، وقد يضطر برنامجك إلى التعامل مع حالات استثنائية مثل ظهور نافذة الدردشة في إحداثيات مختلفة على الشاشة، أو ظهور مربعات التأكيد التي تقاطع رسائلك. يجب على برنامجك التقاط لقطات شاشة لتوجيه تفاعل واجهة المستخدم الرسومية واعتماد طرقٍ لاكتشاف متى لا تُرسَل ضغطات المفاتيح الافتراضية.

> **ملاحظة**: قد ترغب في إعداد بعض الحسابات التجريبية الوهمية حتى لا ترسل رسائل غير مرغوب فيها إلى أصدقائك الحقيقيين عن طريق الخطأ أثناء كتابة هذا البرنامج.

20.18 الخلاصة

تتيح لك أتمتة واجهة المستخدم الرسومية باستخدام وحدة `pyautogui` التفاعل مع التطبيقات الموجودة على حاسوبك من خلال التحكم في الفأرة ولوحة المفاتيح، حيث يُعَد هذا النهج مرنًا بما يكفي لفعل أيّ شيء يمكن للمستخدم تطبيقه، ولكن يتمثّل جانبه السلبي في أن هذه البرامج لا تستطيع رؤية ما تنقر عليه أو تكتبه. حاول التأكد من أن برامج أتمتة واجهة المستخدم الرسومية عند كتابتها ستتعطّل بسرعة عند إعطائها تعليمات سيئة، إذ قد يكون تعطل البرنامج أمرًا مزعجًا، ولكنه أفضل بكثير من استمرار البرنامج مع وجود الخطأ.

يمكنك تحريك مؤشر الفأرة على الشاشة ومحاكاة نقرات الفأرة وضغطات المفاتيح واختصارات لوحة المفاتيح باستخدام وحدة `PyAutoGUI` التي يمكنها أيضًا التحقق من الألوان على الشاشة، ويمكنها أن تزوّد برنامج أتمتة واجهة المستخدم الرسومية الخاص بك بفكرة كافية عن محتويات الشاشة لمعرفة خروجه عن المسار الصحيح أم لا، ويمكنك إعطاء الوحدة `PyAutoGUI` لقطة شاشة والسماح لها بمعرفة إحداثيات المنطقة التي تريد النقر عليها.

يمكنك الجمع بين جميع ميزات `PyAutoGUI` لأتمتة أيّ مهمة متكررة على حاسوبك. قد تكون مشاهدة مؤشر الفأرة يتحرك من تلقاء نفسه ورؤية النص يظهر على الشاشة آليًا أمرًا مملًا للغاية، ولكن يوجد شعور معين بالرضا يأتي من رؤية كيف أنقذك ذكاؤك من إنجاز المهام المملة.

دورة تطوير التطبيقات
باستخدام لغة بايثون

احترف البرمجة وتطوير التطبيقات مع أكاديمية حسوب
والتحق بسوق العمل فور انتهائك من الدورة

التحق بالدورة الآن

الملحق 1: تثبيت الوحدات الخارجية ببايثون

كتب العديد من المطورين وحداتهم الخاصة، مما أدى إلى توسيع قدرات لغة بايثون Python إلى ما هو أبعد مما توفره المكتبة المعيارية للوحدات المُدمَجة مع بايثون، والطريقة الأساسية لتثبيت هذه الوحدات الخارجية هي استخدام الأداة `Pip` في بايثون، حيث تنزِّل هذه الأداة وحدات بايثون وتثبّتها بطريقة آمنة على حاسوبك من موقع الويب الخاص بمؤسسة برمجيات بايثون، ويُعَد PyPI أو Python Package Index متجر التطبيقات المجاني لوحدات بايثون.

> **ملاحظة**: يمكنك تثبيت جميع الوحدات المطلوبة مع الإصدارات المستخدمة في هذا الكتاب من الفصول من خلال تثبيت الوحدة `automateboringstuff` باستخدام الأداة Pip، لذا شغّل الأمر `pip install --user automateboringstuff` من موجّه الأوامر أو نافذة الطرفية Terminal.

الأداة Pip

تُثبَّت الأداة `Pip` تلقائيًا مع الإصدار 3.4 من بايثون والإصدارات الأحدث على نظامي تشغيل ويندوز Windows وماك macOS، ولكن يجب أن تثبّتها تثبيتًا منفصلًا على نظام لينكس Linux. يمكنك معرفة ما إذا كانت الأداة `Pip` مُثبَّتة مسبقًا على نظام لينكس من خلال تشغيل الأمر `which pip3` في نافذة الطرفية Terminal، حيث إذا كانت مثبَّتة، فسترى موقع الأداة `pip3` معروضًا، وإلّا فلن يُعرض أيّ شيء. يمكنك تثبيت الأداة `pip3` على نظام أوبنتو Ubuntu أو نظام دبيان Debian Linux من خلال فتح نافذة طرفية جديدة وإدخال الأمر `sudo apt install python3-pip`، ويمكنك تثبيت الأداة `pip3` على نظام فيدورا لينكس Fedora Linux من خلال إدخال الأمر `sudo dnf install python3-pip` في نافذة الطرفية، ويجب أن تدخِل كلمة مرور المدير الخاصة بحاسوبك.

تُشغَّل الأداة `pip` من نافذة الطرفية (وتُسمَّى أيضًا سطر الأوامر)، وليس من صدفة بايثون التفاعلية Interactive Shell. شغّل برنامج "موجّه الأوامر Command Prompt" من قائمة "ابدأ Start" في نظام تشغيل ويندوز، وشغّل الطرفية من أيقونة سبوت لايت Spotlight على نظام ماك macOS، وشغّل الطرفية من أيقونة Ubuntu Dash أو اضغط على الاختصار `Ctrl-Alt-T` في نظام أوبنتو لينكس Ubuntu Linux.

إن لم يكن مجلد `pip` موجودًا في متغير البيئة `PATH`، فيجب تغيير المجلد في نافذة الطرفية باستخدام الأمر `cd` قبل تشغيل الأداة `pip`.

إذا أردتَ معرفة اسم المستخدم الخاص بك، فشغّل الأمر `echo %USERNAME%` على نظام ويندوز أو الأمر `whoami` على نظامي ماك macOS ولينكس، ثم شغّل الأمر `cd pip folder`، حيث يكون مجلد `pip` على نظام ويندوز في:

```
C:\Users\<USERNAME>\AppData\Local\Programs\Python\Python39\Scripts
```

ويوجد هذا المجلد على نظام ماك macOS في:

```
/Library/Frameworks/Python.framework/Versions/3.9/bin/
```

ويوجد هذا المجلد على نظام لينكس في:

```
/home/<USERNAME>/.local/bin/
```

وبذلك ستكون في المجلد الصحيح لتشغيل الأداة `pip`.

تثبيت الوحدات الخارجية

يُسمَّى الملف القابل للتنفيذ للأداة `pip` بالاسم `pip` على نظام ويندوز ويسمّى `pip3` على نظامي ماك macOS ولينكس. مرّر الأمر `install` متبوعًا باسم الوحدة التي تريد تثبيتها من سطر الأوامر، فمثلًا يمكنك إدخال الأمر `pip install --user MODULE` في نظام ويندوز، حيث `MODULE` هو اسم الوحدة.

قد تكون التغييرات المستقبلية على هذه الوحدات الخارجية غير متوافقة مع الإصدارات السابقة، لذا نوصي بتثبيت الإصدارات المحدَّدة المستخدمة في هذا الكتاب من الفصول كما سنوضّح لاحقًا، حيث يمكنك إضافة الخيار `-U MODULE==VERSION` إلى نهاية اسم الوحدة لتثبيت إصدار محدّد، ولاحظ وجود إشارتي مساواة في خيار سطر الأوامر، فمثلًا يثبّت الأمر `pip install --user -U send2trash==1.5.0` الإصدار 1.5.0 من الوحدة `send2trash`.

يمكنك تثبيت جميع الوحدات التي تغطيها هذا الكتاب من الفصول من خلال تنزيل ملفات المتطلبات "requirements" لنظام تشغيلك من موقع nostarch وتشغيل أحد الأوامر التالية:

- على نظام ويندوز:

```
pip install --user -r automate-win-requirements.txt --user
```

- على نظام ماك macOS:

```
pip3 install --user -r automate-mac-requirements.txt --user
```

- على نظام لينكس:

```
pip3 install --user -r automate-linux-requirements.txt --user
```

تحتوي القائمة التالية على الوحدات الخارجية المستخدمة في هـذا الكتـاب مـع إصـداراتها المختلفـة، حيث يمكنك إدخال هذه الأوامر إدخالًا منفصلًا إذا أردتَ تثبيت عددٍ محدّد من هذه الوحدات على حاسوبك:

- `pip install --user send2trash==1.5.0`
- `pip install --user requests==2.21.0`
- `pip install --user beautifulsoup4==4.7.1`
- `pip install --user selenium==3.141.0`
- `pip install --user openpyxl==2.6.1`
- `pip install --user PyPDF2==1.26.0`
- `pip install --user python-docx==0.8.10` (تثبيت `python-docx` وليس`docx`)
- `pip install --user imapclient==2.1.0`
- `pip install --user pyzmail36==1.0.4`
- `pip install --user twilio`
- `pip install --user ezgmail`
- `pip install --user ezsheets`
- `pip install --user pillow==9.2.0`
- `pip install --user pyobjc-framework-Quartz==5.2` (على نظام ماك فقط)
- `pip install --user pyobjc-core==5.2` (على نظام ماك فقط)
- `pip install --user pyobjc==5.2` (على نظام ماك فقط)

- `pip install --user python3-xlib==0.15` (على نظام لينكس فقط)
- `pip install --user pyautogui`

> **ملاحظة**: يمكن أن يستغرق تثبيت الوحدة `pyobjc` في نظام ماك 20 دقيقة أو أكثر، لذا لا تنزعج إذا استغرق الأمر بعض الوقت، ويجب عليك أيضًا تثبيت الوحدة `pyobjc-core` أولًا، مما سيؤدي إلى تقليل وقت التثبيت الإجمالي.

يمكنك اختبار نجاح تثبيت الوحدة بعد تثبيتها من خلال تشغيل الأمر `import ModuleName` في الصدفة التفاعلية، وإذا لم تُعرَض أيّ رسائل خطأ، فسنفترض نجاح تثبيت هذه الوحدة. إذا كانت الوحدة مثبتة مسبقًا، ولكنك ترغب في ترقيتها إلى أحدث إصدار متاح على موقع PyPI، فشغّل الأمر `pip install --user -U MODULE` أو `pip3 install --user -U MODULE` على نظامي ماك macOS ولينكس، حيث يثبّت الخيار `--user` الوحدة في مجلد المستخدم الخاص بك، مما يؤدي إلى تجنب أخطاء الأذونات المحتملة التي قد تواجهها عند محاولة التثبيت لجميع المستخدمين.

تُجرِي الإصدارات الأحدث من وحدات `Selenium` و `OpenPyXL` تغييرات غير متوافقة مع الإصدارات السابقة المُستخدَمة في هذا الكتاب، بينما تتفاعل وحدات `Twilio` و `EZGmail` و `EZSheets` مع الخدمات عبر الإنترنت، ولكن قد يُطلَب منك تثبيت أحدث إصدار من هذه الوحدات باستخدام الأمر:

```
pip install --user -U
```

> **ملاحظة**: اقترحنا سابقًا استخدام الأمر `sudo` إذا واجهتَ أخطاء في الأذونات أثناء تشغيل أمر pip كما يلي: `sudo pip install module`، ولكنها تُعَد ممارسة سيئة، لأنها تثبّت وحدات مع تثبيت مايثون الذي يستخدمه نظام تشغيلك، إذ قد يشغّل نظام تشغيلك سكربتات بايثون لتنفيذ المهام المتعلقة بالنظام، وإذا ثبّتَ وحدات مع تثبيت بايثون بحيث تتعارض هذه الوحدات مع وحداته الحالية، فقد يؤدي ذلك إلى إنتاج أخطاء يصعب إصلاحها، لذا لا تستخدم الأمر `sudo` أبدًا عند تثبيت وحدات بايثون.

تثبيت وحدات للمحرّر Mu

يمتلك محرّر Mu بيئة بايثون الخاصة به، وهي منفصلة عن البيئة التي تمتلكها عمليات تثبيت بايثون النموذجية، إذ يمكنك تثبيت الوحدات بحيث تستخدمها في السكربتات التي يشغّلها المحرّر Mu من خلال إظهار لوحة المدير Admin Panel عند النقر على رمز الترس في الزاوية اليمنى السفلية للمحرّر Mu، ثم انقر على تبويب الحزم الخارجية Third Party Packages في النافذة التي ستظهر واتبع الإرشادات الخاصة بتثبيت الوحدات في هذا التبويب. لا تزال القدرة على تثبيت الوحدات في المحرّر Mu ميزة حديثة وقيد التطوير، لذا قد تتغير هذه التعليمات.

إن لم تتمكّن من تثبيت الوحدات باستخدام لوحة المدير، فيمكنك أيضًا فتح نافـذة الطرفيـة وتشـغيل الأداة pip الخاصة بالمحرّر Mu. يجب أن تستخدم خيار سطر الأوامـر `--target` الخـاص بـالأداة `pip` لتحديـد مجلـد الوحدة الخـاص بـالمحرّر Mu، وهـذا المجلـد هـو `C:\Users\<USERNAME>\AppData\Local\Mu\pkgs` على نظــام وينــدوز، و`/Applications/mu-editor.app/Contents/Resources/app_packages` على نظام ماك macOS، ولكنك لا تحتاج في نظـام لينكس إلى إدخـال الوسـيط `--target` ، لـذا شـغّل الأمـر `pip3` فقط.

شغّل الأوامر التالية مثلًا بعد تنزيل ملف المتطلبات لنظام تشغيلك من موقع nostarch:

على نظام ويندوز:

```
pip install –r automate-win-requirements.txt --target "C:
\Users\USERNAME
\AppData\Local\Mu\pkgs"
```

على نظام ماك macOS:

```
pip3 install –r automate-mac-requirements.txt --target /Applications/
mu-editor.app/Contents/Resources/app_packages
```

على نظام لينكس:

```
pip3 install --user –r automate-linux-requirements.txt
```

إذا أردتَ تثبيت بعض الوحدات فقط، فيمكنك تشغيل الأمر `pip` (أو `pip3`) العادي وإضافة الوسيط:

```
--target
```

هل تطمح لبيع منتجاتك الرقمية عبر الإنترنت؟

استثمر مهاراتك التقنية وأطلق منتجًا رقميًا
يحقق لك دخلًا عبر بيعه على متجر بيكاليكا

أطلق منتجك الآن

الملحق 2: تشغيل البرامج في بايثون

إذا كان برنامجك مفتوحًا في المحرّر Mu، فيُعَد تشغيله أمرًا بسيطًا من خلال الضغط على مفتاح F5 أو النقر على زر التشغيل Run الموجود أعلى النافذة، وهي طريقة سهلة لتشغيل البرامج أثناء كتابتها، ولكن يمكن أن يكون فتح المحرّر Mu لتشغيل برامجك النهائية عبئًا عليك، لذا توجد طرق أكثر ملاءمة سنوضّحها في هذا الملحق لتنفيذ سكربتات بايثون اعتمادًا على نظام التشغيل الذي تستخدمه.

تشغيل البرامج من نافذة الطرفية Terminal

إذا فتحتَ نافذة طرفية مثل موجّه الأوامر على نظام ويندوز Windows أو الطرفية Terminal على نظامي ماك macOS ولينكس Linux، فسترى نافذةً فارغة لإدخال أوامر نصية فيها. يمكنك تشغيل برامجك من الطرفية، ولكن إن لم تكن معتادًا على ذلك، فسيكون استخدام حاسوبك عبر هذه الطرفية (التي تُسمَّى سطر الأوامر أيضًا) أمرًا صعبًا، إذ لن تقدّم الطرفية أيّ تلميحات حول ما يُفترض أن تفعله على عكس واجهة المستخدم الرسومية GUI.

يمكنك فتح نافذة طرفية على نظام ويندوز من خلال النقر على زر "ابدأ أو Start"، ثم أدخل إلى موجّه الأوامر، واضغط على مفتاح ENTER. انقر على أيقونة Spotlight في الجزء العلوي الأيمن في نظام ماك macOS، واكتب Terminal، ثم اضغط على مفتاح ENTER. يمكنك في نظام أوبنتو لينكس Ubuntu Linux الضغط على مفتاح WIN لإظهار أيقونة Dash، ثم اكتب Terminal، واضغط على مفتاح ENTER، وسيفتح اختصار لوحة المفاتيح Ctrl-Alt-T أيضًا نافذة الطرفية على نظام أوبنتو.

تحتوي الصدفة التفاعلية على سطر الأوامر >>>، وتعرض الطرفية أيضًا هذا السطر لإدخال الأوامر. سيكون المسار التالي هو المسار الكامل للمجلد الذي تتواجد فيه حاليًا على نظام ويندوز:

```
C:\Users\Al>your commands go here
```

يعرض سطر الأوامر على نظام ماك macOS اسم حاسوبك ونقطتين ومجلد العمل الحالي (مع تمثيل مجلد المنزل home directory بالرمز ~ اختصارًا) واسم المستخدم الخاص بك متبوعًا بإشارة الدولار الأمريكي ($) كما يلي:

```
Als-MacBook-Pro:~ al$ your commands go here
```

يشبه سطر الأوامر في نظام أوبنتو لينكس سطرَ الأوامر في نظام ماك macOS باستثناء أنه يبدأ باسم المستخدم والإشارة @ كما يلي:

```
al@al-VirtualBox:~$ your commands go here
```

ملاحظة: يمكن تخصيص سطور الأوامر، ولكن لن نوضّح ذلك في هذا الملحق.

إذا أدخلتَ أمرًا مثل الأمر python على نظام ويندوز أو الأمر python3 على نظامي ماك macOS ولينكس، فستتحقق الطرفية من وجود برنامج يحمل هذا الاسم في المجلد الذي تتواجد فيه حاليًا، وإذا لم تجده هناك، فستتحقق من المجلدات المُدرَجة في متغير البيئة PATH، حيث يمكن عَدّ متغيرات البيئة بأنها متغيرات على مستوى نظام التشغيل بأكمله، إذ تحتوي على عددٍ من إعدادات النظام. يمكنك رؤية القيمة المُخزَّنة في متغير البيئة PATH من خلال تشغيل الأمر echo %PATH% على نظام ويندوز والأمر echo $PATH على نظامي ماك macOS ولينكس. إليك مثال على نظام ماك macOS:

```
Als-MacBook-Pro:~ al$ echo $PATH
/Library/Frameworks/Python.framework/Versions/3.9/bin:/usr/local/
bin:/usr/
bin:/bin:/usr/sbin:/sbin
```

يوجد ملف البرنامج python3 على نظام ماك macOS، بالتحديد في المجلد

```
/Library/Frameworks/Python.framework/Versions/3.9/bin
```

لذلك لا حاجة لإدخال المسار

```
/Library/Frameworks/Python.framework/Versions/3.9/bin/python3
```

يمكنك التبديل إلى هذا المجلد أولًا لتشغيل البرنامج، حيث يمكنك إدخال الأمر python3 من أيّ مجلد، وستجده الطرفية في أحد مجلدات متغير البيئة PATH، إذ تُعَد إضافة مجلد البرنامج إلى متغير البيئة PATH اختصارًا مناسبًا.

إذا أردتَ تشغيل برنامج `.py`، فيجب أن تدخِل الأمر `python` (أو `python3`) متبوعًا باسم ملف `.py`، مما يؤدي إلى تشغيل بايثون التي تشغّل بدورها الشيفرة البرمجية التي تجدها في ملف `.py`، وستعود إلى موجّه الطرفية بعد انتهاء برنامج بايثون.

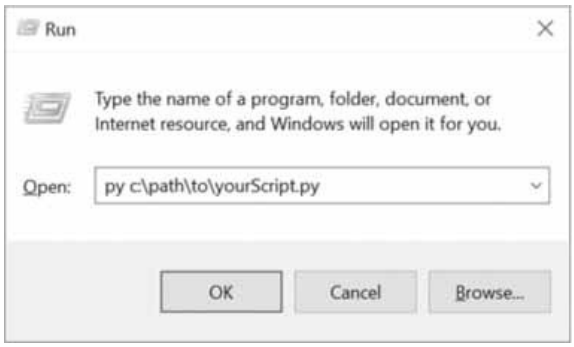
سيبدو برنامج بسيط يظهِر الرسالة "Hello, world!" مثلًا في نظام ويندوز كما يلي:

```
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Al>python hello.py
Hello, world!

C:\Users\Al>
```

يؤدي تشغيل الأمر `python` (أو `python3`) بدون أيّ اسم ملف إلى أن تشغّل بايثون الصدفة التفاعلية.

تشغيل برامج بايثون على نظام ويندوز

توجد عدة طرق أخرى يمكنك من خلالها تشغيل برامج بايثون على نظام ويندوز بدلًا من فتح نافذة الطرفية لتشغيل سكربتات بايثون، حيث يمكنك الضغط على الاختصار `WIN-R` لفتح مربع حوار التشغيل Run وإدخال `py C:\path\to\your\pythonScript.py` كما هو موضح في الشكل التالي. يكون البرنامج `py.exe` مثبّتًا على المسار `C:\Windows\py.exe` الموجود في متغير البيئة `PATH`، وتُعَد كتابة امتداد الملف `.exe` أمرًا اختياريًا عند تشغيل البرامج.



الشكل 116: مربع حوار التشغيل Run في نظام ويندوز

الجانب السلبي لهذه الطريقة هو أنه يجب أن تدخِل المسار الكامل للسكربت الخاص بك، وإذا شغّلنا سكربت بايثون من مربع الحوار، فستُفتح نافذة طرفية جديدة لعرض خرج هذا السكربت، وستُغلق هذه النافذة تلقائيًا عند انتهاء البرنامج، وبالتالي قد تفوّت بعض الخرج. يمكنك حل هذه المشاكل من خلال إنشاء سكربت دفعي Batch Script، والذي هو ملف نصي صغير له امتداد الملف `.bat` ويمكنه تشغيل أوامر طرفية متعددة،

وهو يشبه إلى حد كبير سكربت الصدفة Shell Script في نظامي ماك macOS ولينكس. يمكنك استخدام محرّر نصوص مثل المفكرة Notepad لإنشاء هذه الملفات.

ننشئ ملف دفعي من خلال إنشاء ملف نصي جديد يحتوي على سطر واحد كما يلي:

```
@py.exe C:\path\to\your\pythonScript.py %*
@pause
```

ضع مسار برنامجك المطلق مكان المسار السابق، واحفظ هذا الملف مع امتداد الملف `.bat` مثل الملف `pythonScript.bat`. تمنع الإشارة `@` الموجودة في بداية كل أمر عرضَه في نافذة الطرفية، ويوجّه الرمز `%*` أيّ وسطاء مُدخَلة في سطر الأوامر بعد اسم الملف الدفعي إلى سكربت بايثون، ويقرأ برنامج بايثون بدوره وسطاء سطر الأوامر من قائمة `sys.argv`. سيمنعك هذا الملف الدفعي من الاضطرار إلى كتابة المسار المطلق الكامل لبرنامج بايثون في كل مرة تريد تشغيله فيها، وسيضيف `@pause` عبارة الضغط على أيّ مفتاح للمتابعة `"Press any key to continue..."` بعد نهاية سكربت بايثون لمنع اختفاء نافذة البرنامج بسرعة كبيرة. ننصحك بوضع جميع الملفات الدفعية وملفات `.py` في مجلد واحد موجود فعليًا في متغير البيئة `PATH` مثل المجلد `C:\Users\<USERNAME>`.

لن تحتاج من خلال إعداد ملف دفعي لتشغيل سكربت بايثون إلى فتح نافذة طرفية وكتابة مسار الملف الكامل واسم سكربت بايثون الخاص بك، إذ يمكنك فقط الضغط على الاختصار `WIN-R` وإدخال `pythonScript` (اسم الملف `pythonScript.bat` الكامل ليس ضروريًا)، ثم اضغط على مفتاح `ENTER` لتشغيل السكربت الخاص بك.

تشغيل برامج بايثون على نظام ماك macOS

يمكنك إنشاء سكربت صدفة لتشغيل سكربتات بايثون الخاصة بك في نظام ماك macOS من خلال إنشاء ملف نصي له امتداد الملف `.command`. أنشئ ملفًا جديدًا في محرّر النصوص مثل المحرّر TextEdit وأضِف المحتوى التالي:

```
#!/usr/bin/env bash
python3 /path/to/your/pythonScript.py
```

احفظ هذا الملف مع امتداد الملف `.command` في مجلدك الرئيسي (مثل المجلد `/Users/al`)، واجعل سكربت الصدفة قابلًا للتنفيذ من خلال تشغيل الأمر `chmod u+x yourScript.command` في نافذة الطرفية. ستتمكّن الآن من النقر على أيقونة `Spotlight` (أو الضغط على الاختصار `⌘-SPACE`) وإدخال `yourScript.command` لتشغيل سكربت الصدفة، والذي بدوره سيشغّل سكربت بايثون الخاص بك.

تشغيل برامج بايثون على نظام أوبنتو لينكس

يتطلب تشغيل سكربتات بايثون في أوبنتو لينكس من قائمة Dash إعدادًا إضافيًا. لنفترض أن لدينا السكربت `/home/al/example.py`، إذ قد يكون سكربت بايثون الخاص بك موجودًا في مجلد مختلف وباسم ملف مختلف، ونريد تشغيله من قائمة Dash. استخدم أولًا محرر نصوص مثل المحرّر `gedit` لإنشاء ملف جديد مع المحتوى التالي:

```
[Desktop Entry]
Name=example.py
Exec=gnome-terminal -- /home/al/example.sh
Type=Application
Categories=GTK;GNOME;Utility;
```

احفظ هذا الملف في المجلد `/home/<al>/.local/share/applications` (مع وضع اسم مستخدمك مكان `al`) بالاسم `example.desktop`. إن لم يُظهِر محرّر النصوص مجلد `.local` لأن المجلدات التي تبدأ بنقطة تُعَد مجلدات مخفية، فيجب أن تحفظه في مجلدك الرئيسي (مثل `/home/al`) وتفتح نافذة طرفية لنقل الملف باستخدام الأمر:

```
mv /home/al/example.desktop /home/al/.local/share/applications
```

في حال ما إذا كان الملف `example.desktop` متواجدًا على مستوى هذا المجلد: `/home/al/.local/share/applications`، فستتمكن من الضغط على مفتاح `Windows` في لوحة المفاتيح لإظهار قائمة Dash وكتابة `example.py` (أو أي شيء تضعه في الحقل `Name`)، مما يؤدي إلى فتح نافذة طرفية جديدة أو برنامج `gnome-terminal` على وجه التحديد الذي يشغّل سكربت الصدفة `/home/al/example.sh` الذي سننشئه لاحقًا.

أنشئ ملفًا جديدًا مع المحتوى التالي في محرّر النصوص:

```
#!/usr/bin/env bash
python3 /home/al/example.py
bash
```

احفظ هذا الملف في الملف `/home/al/example.sh` الذي هو سكربت صدفة، والذي هو سكربت يشغّل سلسلةً من أوامر الطرفية، حيث يشغّل سكربت الصدفة سكربت بايثون `/home/al/example.py` ثم يشغّل برنامج صدفة باش Bash Shell. إن لم نضع أمر باش الموجود في السطر الأخير، فستُغلَق نافذة الطرفية بعد انتهاء سكربت بايثون وستفقد أيّ نصٍ يعرضه استدعاء الدالة `print()` على الشاشة.

يجب أن تضيف أذونات التنفيذ إلى سكربت الصدفة، لذا شغّل الأمر التالي من نافذة الطرفية:

```
al@ubuntu:~$ chmod u+x /home/al/example.sh
```

أعـــددنا الملفين `example.desktop` و `example.sh`، وســـتتمكّن الآن من تشـــغيل الســـكربت `example.py` من خلال الضغط على مفتـاح `Windows` وإدخـال `example.py` أو أي اسـم تضـعه في الحقـل `Name` الخاص بالملف `example.desktop`.

تشغيل برامج بايثون مع تعطيل التأكيدات Assertions

يمكنك تعطيل تعليمات `assert` في برامج بايثون الخاصة بك للحصول على تحسين بسيط في الأداء، لـذا ضمّن مفتاح `-O` بعد الأمر `python` أو `python3` وقبل اسم ملف `.py`. عند تشـغيل بـايثون من الطرفيـة، ممـا يؤدي إلى تشغيل نسخة مُحسَّنة من برنامجك تتخطى عمليات التحقق من التأكيد.

دورة تطوير التطبيقات
باستخدام لغة بايثون

مميزات الدورة

- بناء معرض أعمال قوي بمشاريع حقيقية
- وصول مدى الحياة لمحتويات الدورة
- تحديثات مستمرة على الدورة مجانًا
- شهادة معتمدة من أكاديمية حسوب
- إرشادات من المدربين على مدار الساعة
- من الصفر دون الحاجة لخبرة مسبقة

اشترك الآن

الملحق 3: إجابات الأسئلة التدريبية

يحتوي هذا الملحق على إجابات الأسئلة التدريبية الموجودة في نهاية كل فصل من هذا الكتاب، لذا نوصي بشدة بأخذ الوقت الكافي لحل هذه المسائل، إذ تُعَد البرمجة أكبر من مجرد حفظ الصيغ البرمجية وقائمة أسماء الدوال، وكلما مارستها أكثر، ستكتسب منها المزيد كما هو الحال عند تعلم لغة أجنبية. توجد العديد من المواقع التي تحتوي على مسائل برمجية أيضًا، حيث يمكنك العثور على قائمة منها على موقع nostarch.

لا يوجد برنامج واحد صحيح عندما يتعلق الأمر بالمشاريع التدريبية، إذ يمكنك عَدّ البرنامج صحيحًا طالما أنه ينفّذ ما يطلبه المشروع، ولكن إذا أردتَ الاطلاع على أمثلةٍ للمشاريع المكتملة، فهي متوفرة على رابط "تنزيل الملفات المُستخدمة في الكتاب" على موقع nostarch.

إجابات أسئلة الفصل الأول

1. المعاملات هي `+` و `-` و `*` و `/`، والقيم هي `'hello'` و `-88.8` و `5`.
2. المتغير هو `spam`، والسلسلة النصية هي `'spam'`، حيث تبدأ السلاسل النصية وتنتهي بعلامات الاقتباس دائمًا.
3. أنواع البيانات الثلاثة المقدّمة في هذا الفصل هي الأعداد الصحيحة والأعداد العشرية والسلاسل النصية.
4. التعبير Expression هو مزيج قيم ومعاملات، وتُقيَّم جميع التعبيرات (أو تُختصَر) إلى قيمة واحدة.
5. يُقيَّم التعبير إلى قيمة واحدة على عكس التعليمة Statement.
6. ضُبِط المتغير `beef` على القيمة 20، ولا يؤدي التعبير `beef + 1` إلى إعادة إسناد القيمة في المتغير `beef`، إذ قد يحتاج ذلك إلى تعليمة الإسناد: `beef = beef + 1`.

7. يُقيَّم كلا التعبيرين إلى السلسلة النصية `'spamspamspam'`.
8. لا يمكن أن تبدأ أسماء المتغيرات بعدد.
9. ستُقيَّم الدوال `int()` و `float()` و `str()` على النسخة التي تمثّل عددًا صحيحًا وعددًا عشريًا وسلسلة نصية للقيمة المُمرَّرة إلى هذه الدوال.
10. يتسبّب التعبير في حدوث خطأ لأن القيمة `99` هي عدد صحيح، ويمكن دمج السلاسل النصية فقط مع سلاسل نصية أخرى باستخدام المعامل `+`، والطريقة الصحيحة هي `'I have eaten ' + str(99) + ' burritos.'`.

إجابات أسئلة الفصل الثاني

1. `True` و `False` مع استخدام الحرف الكبير T و F وباقي حروف الكلمة هي حروف صغيرة.
2. `and` و `or` و `not`.
3. `True and True` هي `True`.
   - `True and False` هي `False`.
   - `False and True` هي `False`.
   - `False and False` هي `False`.
   - `True or True` هي `True`.
   - `True or False` هي `True`.
   - `False or True` هي `True`.
   - `False or False` هي `False`.
   - `not True` هي `False`.
   - `not False` هي `True`.
4. `False`
   - `False`
   - `True`
   - `False`

- `False`
- `True`

5. `==` و `!=` و `<` و `>` و `<=` و `>=`

6. `==` هو المعامل الذي يقارن قيمتين ويقيّمهما إلى قيمة منطقية Boolean، بينما `=` هو معامل الإسناد الذي يخزّن قيمةً في متغير.

7. الشرط هو تعبير يُستخدَم في تعليمة التحكم في التدفق والذي يُقيَّم إلى قيمة منطقية.

8. تمثّل الكتل الثلاث كل شيء ضمن تعليمة `if` والسطرين `print('beef')` و `print('meat')`.

```
print('eggs')
if spam > 5:
print('beef')
else:
print('meat')
print('spam')
```

9. الشيفرة البرمجية هي:

```
if spam == 1:
print('Hello')
elif spam == 2:
print('Howdy')
else:
print('Greetings!')
```

10. اضغط على الاختصار `CTRL-C` لإيقاف برنامج عالق في حلقة لا نهائية.

11. ستنقل التعليمة `break` التنفيذ إلى خارج الحلقة وبعدها مباشرةً، وستنقل التعليمة `continue` التنفيذ إلى بداية الحلقة.

12. تفعل جميع هذه الدوال الشيء نفسه. تتراوح قيم الاستدعاء `range(10)` من القيمة `0` إلى القيمة `10` (دون تضمينها)، ويخبر الاستدعاء `range(0, 10)` الحلقة صراحةً بأن تبدأ من القيمة `0`، ويخبر الاستدعاء `range(0, 10, 1)` الحلقة صراحةً بزيادة المتغير بمقدار 1 في كل تكرار.

13. الشيفرة البرمجية هي:

```
for i in range(1, 11):
```

```
print(i)
```

وأيضًا:

```
i = 1
while i <= 10:
print(i)
i = i + 1
```

14. يمكن استدعاء هذه الدالة باستخدام `spam.beef()`.

إجابات أسئلة الفصل الثالث

1. تقلّل الدوال من الحاجة إلى تكرار الشيفرة البرمجية، مما يجعل البرامج أقصر وأسهل في القراءة والتعديل.
2. تُنفَّذ الشيفرة البرمجية الموجودة في الدالة عند استدعائها وليس عند تعريفها.
3. تعرّف التعليمة `def` (أو تنشئ) دالة.
4. تتكون الدالة من التعليمة `def` والشيفرة البرمجية الموجودة في التعليمة `def` الخاصة بها. ينقل استدعاء الدالة تنفيذ البرنامج إليها، ويُقيَّم استدعاؤها إلى القيمة التي تعيدها هذه الدالة.
5. يوجد نطاق عام واحد، ويُنشَأ نطاق محلي عند استدعاء الدالة.
6. يُدمَّر النطاق المحلي وتُهمَل جميع المتغيرات الموجودة فيه عندما تعود دالة.
7. القيمة المُعادة هي القيمة التي يُقيَّم استدعاء الدالة إليها، ويمكن استخدام القيمة المُعادة بوصفها جزءًا من التعبير مثل أي قيمة أخرى.
8. إن لم توجَد تعليمة إعادة `return` للدالة، فستكون قيمتها المُعادة هي `None`.
9. تجبر التعليمة `global` المتغير في الدالة على الإشارة إلى المتغير العام.
10. نوع بيانات القيمة `None` هو `NoneType`.
11. تستورد التعليمة `import` وحدةً باسم `areallyourpetsnamederic`، وهي ليست وحدة حقيقية.
12. يمكن استدعاء هذه الدالة باستخدام `spam.beef()`.
13. ضع سطر الشيفرة البرمجية الذي قد يسبّب خطأً في التعليمة `try`.

14. توجد الشيفرة البرمجية التي يُحتمَل أن تسبّب خطأ في التعليمة `try`، وتوجد الشيفرة البرمجية التي تُنفَّذ في حالة حدوث خطأ في التعليمة `except`.

إجابات أسئلة الفصل الرابع

1. قيمة القائمة الفارغة، وهي قيمة القائمة التي لا تحتوي على عناصر، وهي مشابهة لقيمة السلسلة النصية الفارغة `''`.
2. `'spam[2] = 'hello'` (لاحظ أن القيمة الثالثة في القائمة موجودة في الفهرس `2` لأن الفهرس الأول هو `0`).
3. الإجابات كالآتي:
   - `'d'` (لاحظ أن `'3' * 2` هي السلسلة النصية `'33'`، والتي نمرّرها إلى الدالة `int()` قبل تقسيمها على العدد 11، وتُقيَّم في النهاية إلى القيمة 3، حيث يمكن استخدام التعابير في أيّ مكان نستخدم القيم فيه).
   - `'d'` (تُحسَب الفهارس السالبة من النهاية).
   - `['a', 'b']`
4. الإجابات كالآتي:
   - 1
   - `[3.14, 'cat', 11, 'cat', True, 99]`
   - `[3.14, 11, 'cat', True]`
5. معامل دمج القوائم هو `+`، ومعامل التكرار هو `*`، ونستخدم الشيء نفسه مع السلاسل النصية.
6. يضيف التابع `append()` قيمًا إلى نهاية القائمة فقط، بينما يضيفها التابع `insert()` في أيّ مكان من القائمة.
7. تُعَد التعليمة `del` وتابع القائمة `remove()` طريقتين لإزالة القيم من القائمة.
8. يمكن تمرير كلٍّ من القوائم والسلاسل النصية إلى الدالة `len()`، وتحتوي على فهارس وأقسام، ويمكن استخدامها في حلقات `for`، ويمكن دمجها أو تكرارها، وتُستخدَم مع المعاملات `in` و `not in`.
9. تُعَد القوائم قابلة للتغيير، إذ يمكن إضافة قيم إليها أو إزالتها أو تغييرها، بينما تُعَد المجموعات Tuples غير قابلة للتغيير، إذ لا يمكن تعديلها أبدًا. تُكتَب المجموعات باستخدام الأقواس `(` و `)`، بينما تستخدم القوائم الأقواس المربعة `[` و `]`.

10. `(42,)` (تُعَد الفاصلة في النهاية إجبارية).

11. الدالتان `tuple()` و `list()` على التوالي.

12. تحتوي على مراجع إلى قيم القائمة.

13. تنشئ الدالة `copy.copy()` نسخة سطحية Shallow Copy من القائمة، بينما تنشئ الدالة `copy.deepcopy()` نسخة عميقة Deep Copy من القائمة، أي أن `copy.deepcopy()` هي التي ستكرّر أي قوائم موجودة ضمن القائمة فقط.

إجابات أسئلة الفصل الخامس

1. قوسان معقوصان: `{}`.

2. `{'foo': 42}`.

3. تكون العناصر المُخزَّنة في القاموس غير مرتبة، بينما تكون العناصر الموجودة في القائمة مرتبة.

4. تحصل على خطأ `KeyError`.

5. لا يوجد فرق، حيث يتحقق المعامل `in` من وجود قيمة بوصفها مفتاحًا في القاموس.

6. تتحقق التعليمة `'cat' in spam` من وجود المفتاح `'cat'` في القاموس، بينما تتحقق `'cat' in spam.values()` من وجود القيمة `'cat'` لأحد المفاتيح في `spam`.

7. `spam.setdefault('color','black')`

8. `pprint.pprint()`

إجابات أسئلة الفصل السادس

1. تمثل محارف الهروب Escape محارفًا في قيم السلاسل النصية التي قد يكون من الصعب أو من المستحيل كتابتها في الشيفرة البرمجية.

2. يمثّل `\n` سطرًا جديدًا، ويمثّل `\t` مفتاح الجدولة Tab.

3. يمثل محرف الهروب `\\` محرف الخط المائل العكسي.

4. ليست علامة الاقتباس المفردة في `Howl's` خطأ لأنك استخدمتَ علامات الاقتباس المزدوجة لتحديد بداية السلسلة النصية ونهايتها.

5. تسمح السلاسل النصية متعددة الأسطر باستخدام الأسطر الجديدة في السلاسل النصية بدون محرف الهروب `\n`.

6. نقيّم التعابير كما يلي:
   - `'e'`
   - `'Hello'`
   - `'Hello'`
   - `'lo, world!`
7. نقيّم التعابير كما يلي:
   - `'HELLO'`
   - `True`
   - `'hello'`
8. نقيّم التعابير كما يلي:
   - `['Remember,', 'remember,', 'the', 'fifth', 'of', 'November.']`
   - `'There-can-be-only-one.'`
9. توابع السلاسل النصية `rjust()` و `ljust()` و `center()` على التوالي.
10. يزيل التابعان `lstrip()` و `rstrip()` المسافات البيضاء من النهايتين اليسرى واليمنى للسلسلة النصية على التوالي.

إجابات أسئلة الفصل السابع

1. تعيد الدالة `re.compile()` كائنات `Regex`.
2. تُستخدَم السلاسل النصية الخام دون الحاجة للهروب من محارف الخطوط المائلة العكسية.
3. يعيد التابع `search()` كائنات `Match`.
4. يعيد التابع `group()` سلاسلًا نصية للنص المطابق.
5. المجموعة 0 هي النص المطابق بأكمله، وتغطي المجموعة 1 المجموعة الأولى من الأقواس، وتغطي المجموعة 2 المجموعة الثانية من الأقواس.
6. يمكن الهروب من النقاط والأقواس باستخدام خط مائل عكسي مثل: `\.` و `\(` و `\)`.
7. إن لم يكن للتعبير النمطي Regex أيّ مجموعات Groups، فستُعاد قائمة من السلاسل النصية، وإذا كان للتعبير النمطي مجموعات، فستُعاد قائمة من مجموعات Tuples السلاسل النصية.

8. يدل المحرف `|` على المطابقة "إما، أو" بين مجموعتين.
9. يمثّل المحرف `?` إما "تطابق صفر أو واحد من المجموعة السابقة" أو يمكن استخدامه للإشارة إلى المطابقة غير الشرهة.
10. يطابق `+` واحدًا أو أكثر، ويطابق `*` صفرًا أو أكثر.
11. يطابق `{3}` تمامًا ثلاث نسخ من المجموعة السابقة، ويطابق `{3,5}` بين ثلاث وخمس نسخ.
12. تطابق أصناف المحارف المختصرة `\d` و `\w` و `\s` مع رقم واحد أو كلمة أو محرف مسافة على التوالي.
13. تطابق أصناف المحارف المختصرة `\D` و `\W` و `\S` مع محرف واحد ليس رقمًا أو كلمة أو محرف مسافة على التوالي.
14. يجري `.*` مطابقة شرهة، ويجري `.*?` مطابقة غير شرهة.
15. إما `[0-9a-z]` أو `[a-z0-9]`.
16. يؤدي تمرير `re.I` أو `re.IGNORECASE` كوسيط ثانٍ إلى التابع `re.compile()` إلى جعل المطابقة غير حساسة لحالة الأحرف.
17. يتطابق المحرف `.` مع أيّ محرف باستثناء محرف السطر الجديد. إذا مرّرنا `re.DOTALL` كوسيطٍ ثانٍ إلى التابع `re.compile()`، فستتطابق النقطة مع محارف السطر الجديد أيضًا.
18. يعيد استدعاء الدالة `sub()` السلسلة النصية `X drummers, X pipers, five rings, X' hens'`.
19. يتيح الوسيط `re.VERBOSE` إضافة مسافة بيضاء وتعليقات إلى السلسلة النصية التي نمرّرها إلى التابع `re.compile()`.
20. ينشئ الاستدعاء `re.compile(r'^\d{1,3}(,\d{3})*$')` هذا التعبير النمطي، ولكن يمكن أن تنتج سلاسل نصية لتعابير نمطية أخرى تعبيرًا نمطيًا مشابهًا.
21. `re.compile(r'[A-Z][a-z]*\sWatanabe')`
22. `re.compile(r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats| baseballs)\.', re.IGNORECASE)`

إجابات أسئلة الفصل الثامن

1. لا، فالوحدة `PyInputPlus` هي وحدة خارجية ولا تأتي مع مكتبة Python المعيارية.

2. يؤدي ذلك اختياريًا إلى جعل شيفرتك البرمجية أقصر في الكتابة، حيث يمكنك كتابة `pyip.inputStr()` بدلًا من `pyinputplus.inputStr()`.
3. تعيد الدالة `inputInt()` قيمة عددية صحيحة int، بينما تعيد الدالة `inputFloat()` قيمة عددية عشرية، وهذا هو الفرق بين إعادة القيمتين 4 و 4.0.
4. استدعِ الدالة `pyip.inputint(min=0, max=99)`.
5. قائمة من السلاسل النصية للتعابير النمطية المسموح بها أو المرفوضة صراحةً.
6. سترفع الدالة الاستثناء `RetryLimitException`.
7. تعيد الدالة القيمة `'hello'`.

إجابات أسئلة الفصل التاسع

1. تتعلق المسارات النسبية بمجلد العمل الحالي.
2. تبدأ المسارات المطلقة بالمجلد الجذر مثل `/` أو `C:\`.
3. يُقيَّم في نظام ويندوز Windows بأنه `WindowsPath('C:/Users/Al')`، ويُقيَّم في أنظمة التشغيل الأخرى بأنه نوع مختلف من كائن `Path` ولكن مع المسار نفسه.
4. يؤدي التعبير `'Al' / 'C:/Users'` إلى حدوث خطأ، إذ لا يمكنك استخدام المعامل `/` لضم سلسلتين نصيتين.
5. تعيد الدالة `os.getcwd()` مجلد العمل الحالي، وتغيّر الدالة `os.chdir()` مجلد العمل الحالي.
6. المجلد `.` هو المجلد الحالي، والمجلد `..` هو المجلد الأب.
7. `C:\beef\eggs` هو اسم المجلد `dir`، بينما `spam.txt` هو الاسم الأساسي أو اسم الملف.
8. تمثّل السلسلة النصية `'r'` وضع القراءة، وتمثّل السلسلة النصية `'w'` وضع الكتابة، وتمثّل السلسلة النصية `'a'` وضع الإلحاق.
9. يُمسَح الملف الموجود مسبقًا والمفتوح في وضع الكتابة ويُكتَب فوقه بالكامل.
10. يعيد التابع `read()` محتويات الملف بالكامل بوصفها قيمة سلسلة نصية واحدة، ويعيد التابع `readlines()` قائمة من السلاسل النصية، إذ تكون كل سلسلة نصية سطرًا من محتويات الملف.
11. تشبه القيمة `Shelf` قيمة القاموس، إذ تحتوي على مفاتيح وقيم مع التوابع `keys()` و `values()` التي تعمل بطريقة مشابهة لتوابع القاموس وتحمل الأسماء نفسها.

إجابات أسئلة الفصل العاشر

1. تنسخ الدالة `shutil.copy()` ملفًا واحدًا، بينما تنسخ الدالة `shutil.copytree()` مجلدًا كاملًا مع جميع محتوياته.
2. تُستخدَم الدالة `shutil.move()` لإعادة تسمية الملفات ونقلها.
3. تنقل دوال الوحدة `send2trash` الملف أو المجلد إلى سلة المحذوفات، بينما تحذف دوال الوحدة `shutil` الملفات والمجلدات نهائيًا.
4. تكافئ الدالة `zipfile.ZipFile()` الدالة `open()`، حيث يكون الوسيط الأول هو اسم الملف والوسيط الثاني هو وضع فتح الملف المضغوط ZIP (للقراءة أو الكتابة أو الإلحاق).

إجابات أسئلة الفصل الحادي عشر

1. `assert spam >= 10, 'The spam variable is less than 10.'`
2. إما `assert eggs.lower() != beef.lower() 'The eggs and beef variables are the same!'` أو `assert eggs.upper() != beef.upper(), 'The eggs and beef variables are the same!'`.
3. `assert False, 'This assertion always triggers.'`
4. يمكنك استدعاء الدالة `logging.debug()`، ولكن يجب أن يكون لديك أولًا السطران التاليان في بداية البرنامج:

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -
%(levelname)s -  %(message)s')
```

5. يمكنك إرسال رسائل التسجيل Logging إلى ملف بالاسم `programLog.txt` باستخدام الدالة `logging.debug()`، ولكن يجب أن يكون لديك أولًا السطران التاليان في بداية البرنامج:

```
import logging
>>> logging.basicConfig(filename='programLog.txt',
level=logging.DEBUG,
format=' %(asctime)s -  %(levelname)s -  %(message)s')
```

6. DEBUG و INFO و WARNING و ERROR و CRITICAL.
7. `logging.disable(logging.CRITICAL)`

8. يمكنك تعطيل رسائل التسجيل دون إزالة استدعاءات دوال التسجيل، ويمكنك تعطيل رسائل التسجيل ذات المستوى الأدنى انتقائيًا، ويمكنك إنشاء رسائل التسجيل، كما توفر رسائل التسجيل علامة زمنية.

9. ينقل الزر "Step In" منقح الأخطاء Debugger إلى استدعاء الدالة، وينفّذ الزر `Step Over` استدعاء الدالة بسرعة دون الدخول إليها، وينفّذ الزر "Step Out" بقية الشيفرة البرمجية بسرعة حتى يخرج من الدالة الموجود فيها حاليًا.

10. إذا نقرتَ على زر المتابعة "Continue"، فسيتوقّف منقح الأخطاء عندما يصل إلى نهاية البرنامج أو إلى سطرٍ له نقطة توقف Breakpoint.

11. نقطة التوقف هي إعدادٌ نضعه عند سطر من الشيفرة البرمجية ويؤدي إلى توقّف منقح الأخطاء مؤقتًا عندما يصل تنفيذ البرنامج إلى هذا السطر.

12. نضبط نقاط توقف في المحرّر Mu من خلال النقر على رقم السطر لإظهار نقطة حمراء بجانبه.

إجابات أسئلة الفصل الثاني عشر

1. تحتوي الوحدة `webbrowser` على التابع `open()` الذي يشغّل متصفح الويب مع عنوان URL محدّد، ويمكن للوحدة `requests` تنزيل الملفات والصفحات من الويب، وتحلّل الوحدة `BeautifulSoup` شيفرة HTML، ويمكن للوحدة `selenium` تشغيل المتصفح والتحكم فيه.

2. تعيد الدالة `requests.get()` كائن `Response` الذي يحتوي على سمةٍ Attribute هي السمة `text` التي تحتوي على المحتوى المُنزَّل بوصفه سلسلة نصية.

3. يرفع التابع `raise_for_status()` استثناءً عند وجود مشاكل في التنزيل، ولا يفعل أي شيء عند نجاح التنزيل.

4. تحتوي السمة `status_code` الخاصة بكائن `Response` على رمز حالة HTTP.

5. إذا فتحتَ الملف الجديد على حاسوبك في وضع الكتابة الثنائي `'wb'`، فاستخدم حلقة `for` التي تتكرر مع التابع `iter_content()` الخاص بكائن `Response` لكتابة أجزاء إلى الملف كما في المثال التالي:

```
saveFile = open('filename.html', 'wb')
for chunk in res.iter_content(100000):
saveFile.write(chunk)
```

6. يعرض المفتاح `F12` أدوات المطور في متصفح كروم، ويؤدي الضغط على الاختصار `CTRL-SHIFT-C` (في نظامي ويندوز ولينكس) أو الاختصار `⌘-OPTION-C` (في نظام macOS) إلى إظهار أدوات المطور في فايرفوكس Firefox.
7. انقر بزر الفأرة الأيمن على العنصر الموجود في الصفحة وحدّد خيار فحص العنصر Inspect Element من القائمة.
8. `'#main'`
9. `'.highlight'`
10. `'div div'`
11. `'button[value="favorite"]'`
12. `spam.getText()`
13. `linkElem.attrs`
14. نستورد الوحدة `selenium` باستخدام التعليمة `from selenium import webdriver`.
15. تعيد توابع `find_element_*` العنصر المطابق الأول بوصفه كائن `WebElement`، بينما تعيد توابع `find_elements_*` قائمة بجميع العناصر المطابقة بوصفها كائنات `WebElement`.
16. يحاكي التابعان `click()` و `send_keys()` نقرات الفأرة ومفاتيح لوحة المفاتيح على التوالي.
17. يؤدي استدعاء التابع `submit()` مع أيّ عنصر ضمن الاستمارة إلى إرسال الاستمارة.
18. تحاكي توابع كائن `WebDriver` التي هي `forward()` و `back()` و `refresh()` الأزرار المقابلة لها في المتصفح.

إجابات أسئلة الفصل الثالث عشر

1. تعيد الدالة `openpyxl.load_workbook()` كائن `Workbook`.
2. تحتوي السمة `sheetnames` على كائن `Worksheet`.
3. من خلال تشغيل `wb['Sheet1']`.
4. باستخدام `wb.active`.
5. `sheet['C5'].value` أو `sheet.cell(row=5, column=3).value`.
6. `sheet['C5'] = 'Hello'` أو `sheet.cell(row=5, column=3).value = 'Hello'`.

7. `cell.row` و `cell.column`.
8. تحتوي على القيمة الأعلى للعمود والصف في الورقة على التوالي بوصفها قيم عددية صحيحة.
9. `openpyxl.cell.column_index_from_string('M')`
10. `openpyxl.cell.get_column_letter(14)`
11. `sheet['A1':'F1']`
12. `wb.save('example.xlsx')`
13. يمكن ضبط الصيغة باستخدام الطريقة نفسها لضبط أيّ قيمة أخرى. اضبط السمة `value` الخاصة بالخلية على السلسلة النصية التي تمثّل نص الصيغة، وتذكّر أن الصيغ تبدأ بإشارة المساواة `=`.
14. مرّر القيمة `True` لوسيط الكلمة المفتاحية Keyword Argument الذي هو `data_only` عند استدعاء الدالة `load_workbook()`.
15. `sheet.row_dimensions[5].height = 100`
16. `sheet.column_dimensions['C'].hidden = True`
17. الأجزاء المُثبَّتة هي صفوف وأعمدة ستظهر دائمًا على الشاشة، وهي مفيدة للترويسات.
18. الإجابة هي كل من:
    - `openpyxl.chart.Reference()`
    - `openpyxl.chart.Series()`
    - `openpyxl.chart.BarChart()`
    - `chartObj.append(seriesObj)`
    - `.add_chart()`

إجابات أسئلة الفصل الرابع عشر

1. يمكن الوصول إلى جداول بيانات جوجل Google Sheets من خلال استخدام ملف الاعتماديات Credentials File والملف المفتاحي Token File لجداول بيانات جوجل والملف المفتاحي لجوجل درايف Google Drive.
2. تحتوي الوحدة `EZSheets` على كائنات `ezsheets.Spreadsheet` و `ezsheets.Sheet`.

3. استدعِ التابع `downloadAsExcel()` الخاص بكائن `Spreadsheet`.
4. استدعِ الدالة `ezsheets.upload()` ومرّر إليها اسم ملف إكسل Excel.
5. من خلال الوصول إلى `ss['Students']['B2']`.
6. استدعِ الدالة `ezsheets.getColumnLetterOf(999)`.
7. من خلال الوصول إلى الخاصيتين `rowCount` و `columnCount` لكائن `Sheet`.
8. استدعِ التابع `delete()` الخاص بكائن `Sheet`، ويكون هذا الحذف نهائيًا إذا مرّرتَ وسيط الكلمة المفتاحية `permanent=True` فقط.
9. تنشئ الدالة `createSpreadsheet()` والتابع `createSheet()` الخاص بكائن `Spreadsheet` كائنات `Spreadsheet` و `Sheet` على التوالي.
10. ستعمل وحدة `EZSheets` على تقييد استدعاءات التوابع الخاصة بك.

إجابات أسئلة الفصل الخامس عشر

1. كائن `File` الذي تعيده الدالة `open()`.
2. وضع القراءة الثنائي (`'rb'`) مع الدالة `PdfFileReader()` ووضع الكتابة الثنائي (`'wb'`) مع الدالة `PdfFileWriter()`.
3. يعيد استدعاء التابع `getPage(4)` كائن `Page` للصفحة 5، لأن الصفحة 0 هي الصفحة الأولى.
4. يخزّن المتغير `numPages` عددًا صحيحًا يمثّل عدد الصفحات في كائن `PdfFileReader`.
5. استدعِ الدالة `decrypt('swordfish')`.
6. التابعان `rotateClockwise()` و `rotateCounterClockwise()`، ونمرّر درجات التدوير بوصفها وسيطًا عدديًا صحيحًا.
7. `docx.Document('demo.docx')`
8. يحتوي المستند على فقرات متعددة، حيث تبدأ الفقرة في سطر جديد وتحتوي على كائنات `Run` متعددة، وكائنات `Run` هي مجموعات متجاورة من المحارف ضمن الفقرة.
9. استخدم السمة `doc.paragraphs`.
10. يحتوي كائن `Run` (وليس كائن الفقرة `Paragraph`) على هذه المتغيرات.

11. تجعل القيمة `True` كائن `Run` بخط عريض دائمًا، وتجعله القيمة `False` بخط غير عريض دائمًا، وذلك بغض النظر عن إعداد نمط الخط العريض. تجعل القيمة `None` كائن `Run` يستخدم إعداد نمط الخط العريض فقط.
12. من خلال استدعاء الدالة `docx.Document()`.
13. `doc.add_paragraph('Hello there!')`
14. الأعداد الصحيحة 0 و 1 و 2 و 3 و4

إجابات أسئلة الفصل السادس عشر

1. يمكن أن تحتوي جداول البيانات في إكسل على قيم لأنواع بيانات أخرى غير السلاسل النصية، إذ يمكن أن تحتوي الخلايا على خطوط أو أحجام أو إعدادات ألوان مختلفة، ويمكن أن يكون للخلايا عروض وارتفاعات مختلفة، ويمكن دمج الخلايا المتجاورة، ويمكنك تضمين الصور والرسوم البيانية.
2. تمرّر كائن `File` الذي تحصل عليه من استدعاء الدالة `open()`.
3. يجب فتح كائنات `File` في وضع القراءة الثنائي (`'rb'`) لكائنات `reader` وفي وضع الكتابة الثنائي (`'wb'`) لكائنات `writer`.
4. التابع `writerow()`.
5. يغيّر الوسيط `delimiter` السلسلة النصية المُستخدَمة لفصل الخلايا في صف واحد، ويغير الوسيط `lineterminator` السلسلة النصية المُستخدَمة لفصل الصفوف.
6. `json.loads()`
7. `json.dumps()`

إجابات أسئلة الفصل السابع عشر

1. توقيت يونيكس Unix Epoch هو اللحظة المرجعية التي تستخدمها العديد من برامج التاريخ والوقت، وهذه اللحظة هي 1 من الشهر الأول من عام 1970 وفق التوقيت العالمي المنسَّق UTC.
2. `time.time()`
3. `time.sleep(5)`
4. تعيد أقرب عدد صحيح إلى الوسيط المُمرَّر، فمثلًا تعيد الدالة `round(2.4)` القيمة 2.
5. يمثّل كائن `datetime` لحظة محددة من الزمن، ويمثل كائن `timedelta` مدة زمنية.

6. يعيد تشغيل `datetime.datetime(2019, 1, 7).weekday()` القيمة 0، وهذا يعني يوم الاثنين Monday، حيث تستخدم وحدة `datetime` القيمة 0 ليوم الاثنين والقيمة 1 ليوم الثلاثاء Tuesday وهكذا حتى الوصول إلى القيمة 6 ليوم الأحد Sunday.

7. `threadObj = threading.Thread(target=spam) threadObj.start()`

8. تأكّد من أن الشيفرة البرمجية التي نشغّلها في أحد الخيوط Thread لا تقرأ أو تكتب المتغيرات نفسها للشيفرة البرمجية التي نشغّلها في خيطٍ آخر.

إجابات أسئلة الفصل الثامن عشر

1. بروتوكول SMTP وبروتوكول IMAP على التوالي.

2. `smtplib.SMTP()` و`smtpObj.ehlo()` و`smptObj.starttls()` و`smtpObj.login()`.

3. `imapclient.IMAPClient()` و`imapObj.login()`.

4. قائمة بسلاسل نصية تمثّل الكلمات المفتاحية لبروتوكول IMAP مثل `'BEFORE <date>'` أو `'FROM <string>'` أو `'SEEN'`.

5. أسنِد قيمة عددية صحيحة كبيرة إلى المتغير `imaplib._MAXLINE` مثل القيمة `10000000`.

6. تقرأ الوحدة `pyzmail` رسائل البريد الإلكتروني التي نزّلناها.

7. تخبر ملفات `credentials.json` و `token.json` وحدة `EZGmail` بحساب جوجل الذي يجب استخدامه عند الوصول إلى جيميل Gmail.

8. تمثل الرسالة بريدًا إلكترونيًا واحدًا، بينما تمثّل المحادثة الصادرة والواردة التي تتضمن عدة رسائل بريد إلكتروني سلسلة محادثات Thread.

9. ضمّن النص `'has:attachment'` في السلسلة النصية التي تمررها إلى التابع `search()`.

10. ستحتاج إلى رقم معرّف SID لحساب Twilio ورقم مفتاح الاستيثاق Authentication Token ورقم هاتف Twilio الخاص بك.

إجابات أسئلة الفصل التاسع عشر

1. قيمة RGBA هي مجموعة مكونة من 4 أعداد صحيحة، وتتراوح قيمة كلٍّ منها من 0 إلى 255.حيث تقابل الأعداد الصحيحة الأربعة مقدار اللون الأحمر والأخضر والأزرق وألفا (الشفافية Transparency) في اللون.

2. يعيد استدعاء الدالة `ImageColor.getcolor('CornflowerBlue', 'RGBA')` قيمة RGBA لهذا اللون وهي `(100, 149, 237, 255)`.
3. المجموعة المربعة Box Tuple هي قيمة لمجموعة مكونة من أربعة أعداد صحيحة هي: الإحداثي x للحافة اليسرى والإحداثي y للحافة العلوية والعرض والارتفاع على التوالي.
4. `Image.open('zophie.png')`
5. `imageObj.size` هي مجموعة مكونة من عددين صحيحين هما: العرض والارتفاع.
6. `imageObj.crop((0, 50, 50, 50))`. لاحظ أنك تمرّر مجموعة مربعة إلى التابع `crop()`، وليس أربعة وسطاء منفصلة نوعها أعداد صحيحة.
7. استدعِ التابع `imageObj.save('new_filename.png')` الخاص بكائن `Image`.
8. تحتوي الوحدة `ImageDraw` على شيفرة برمجية للرسم على الصور.
9. تحتوي كائنات `ImageDraw` على توابع لرسم الأشكال مثل `point()` أو `line()` أو `rectangle()`، وتُعاد من خلال تمرير كائن `Image` إلى الدالة `ImageDraw.Draw()`.

إجابات أسئلة الفصل العشرين

1. حرّك الفأرة إلى الزاوية العلوية اليسرى من الشاشة أو الإحداثيات (0, 0).
2. تعيد الدالة `pyautogui.size()` مجموعة مؤلفة من عددين صحيحين لعرض الشاشة وارتفاعها.
3. تعيد الدالة `pyautogui.position()` مجموعة مؤلفة من عددين صحيحين لإحداثيات x و y لمؤشر الفأرة.
4. تحرّك الدالة `moveTo()` الفأرة إلى الإحداثيات المطلقة على الشاشة، بينما تحرّك الدالة `move()` الفأرة نسبةً إلى موضع الفأرة الحالي.
5. `pyautogui.dragTo()` و`pyautogui.drag()`.
6. `pyautogui.typewrite('Hello, world!')`
7. مرّر قائمة بالسلاسل النصية لمفاتيح لوحة المفاتيح إلى الدالة `pyautogui.write()` مثل `'left'` أو مرّر سلسلة نصية واحدة لمفتاح من لوحة المفاتيح إلى الدالة `pyautogui.press()`.
8. `pyautogui.screenshot('screenshot.png')`
9. `pyautogui.PAUSE = 2`

10. يجب أن تستخدم أداة `Selenium` للتحكم في متصفح الويب بدلًا من وحدة `PyAutoGUI`.

11. لا يمكن لوحدة `PyAutoGUI` أن ترى ما تنقر عليه أو ما تكتبه ولا يمكنها بسهولة معرفة فيما أنها تنقر وتكتب في النوافذ الصحيحة أم لا، إذ قد تؤدي النوافذ المنبثقة أو الأخطاء غير المتوقعة إلى إخراج السكربت عن المسار الصحيح وستطلب منك إيقاف تشغيله.

12. استدعِ الدالة `pyautogui.getWindowsWithTitle('Notepad')`.

13. من خلال تشغيل `w=pyatuogui.getWindowsWithTitle('Firefox')` ثم تشغيل `w.activate()`.

أحدث إصدارات أكاديمية حسوب