أكاديمية التعلم
Academy Of Learning

أساسيات البرمجة بلغة بايثون

دبلوم الأمن السيبراني

تحت إشراف
المؤسسة العامة للتدريب التقني والمهني
Technical and Vocational Training Corporation

❖ الأهداف التفصيلية للمقرر:

بنهاية هذا المقرر سيكون المتدرب قادراً وبكفاءة على أن:

- يثبت برنامج بايثون.
- يستخدم بيئة التطوير Python's IDLE.
- يعرف المتغيرات والثوابت.
- يكتب برامج باستخدام لغة البايثون.
- يطبق أساسيات لغة البرمجة بالبايثون.
- يستخدم جمل الشرط والتكرار في لغة البرمجة بايثون.
- يستخدم الدوال.
- يتعامل مع الصفوف والقوائم والمجموعات القواميس في لغة بايثون.
- يحدد مفاهيم البرمجة الكائنية OOP.
- يعالج الأخطاء التي تواجهه في لغة البرمجة بايثون.

❖ فهرس الكتاب

الفصل الأول : البدء مع البايثون

الفصل الثاني: البدء مع البايثون

الفصل الثالث: التعامل مع بنية البيانات

الفصل الأول

البدء مع بايثون

في هذا الفصل سنتعرف على المواضيع التالية:

- مقدمة البايثون تاريخها و أهميتها
- ماهي لغة البايثون ومميزاتها واستخداماتها
- مميزات البايثون بشكل موجز
- اذن استخدامات البايثون
- تثبيت بايثون علىWindows
- محررات الأكواد
- بيئة Python's IDLE
- فهم نافذةIDLE
- استخدام Python shell
- المكتبة القياسية
- قواعد بناء جملة بايثون وإضافة التعليقات
- التعامل مع بايثون وكتابة الأكواد
- تعريف المتغيرات وأنواع البيانات في بايثون

نهج المبرمج

مثل الساحر، يكون للمبرمج قوة غريبة، على سبيل المثال تستطيع تحويل جهاز إلى جهاز آخر وآلة حاسبة إلى آلة كاتبة أو رسم وقليل من السحر بعد تستطيع تحويل الأمير إلى ضفدع وذلك باستخدام لوحة الفاتيح لتدخل بعض التعويذات الغامضة مثل الساحر. وهو قادر على علاج التطبيقات السيئة.

لكن كيف يكون هذا ممكنا؟ وقد يبدو هذا متناقضا، كما لاحظنا سابقا، العلم الحقيقي هو في الواقع الذي لا يؤمن بأي سحر وأي هبة ولا تدخل خارق .فيظل باردًا و عنيداً يسعى وراء المنطق الغير مريح .

تفكير المبرمج يجمع بيت البنى الفكرية المعقدة، المماثلة لتلك التي قام بها العلماء الرياضيات والمهندسون والعلماء. كما في الرياضيات فإن البرمجة تستخدم لغات رسمية لوصف المنطق (أو الخوارزميات) مثل المهندس، الذي يصنع الاجهزة، فيجمع عناصر لتنفيذ آليات وتقييم أداءها. مثل العالم يلاحظ سلوك النظام المعقد ويصنع النماذج ويختبرها.

❖ **مقدمة البايثون تاريخها و أهميتها**

كعالم بيانات جديد، يبدأ مسارك بلغة البرمجة التي تحتاج إلى تعلمها. من بين جميع اللغات التي يمكنك االختيار من بينها، فإن بايثون Python هي الأكثر شعبية بالنسبة للعالم.

هي لغة برمجة عالية المستوى موجهة للكائنات وتستخدم لمجموعة واسعة من المشكلات ذات النطاق والتعقيد المتفاوتين. على عكس العديد من اللغات المماثلة، من السهل إتقانها وتعلمها وهي مثالية للمبتدئين. لكن هذه السهولة ليست السبب الوحيد لأهميتها، هي قوية بما يكفي حتى للمستخدمين المتقدمين. بالاضافة إلى ذلك، تعد بايثون أكثر أدوات علوم البيانات استخداماً، وهي مدرجة كشرط في معظم قوائم إعلانات وظائف علوم البيانات.

تاريخها

ابتكرها وطورها جايدو فان أواخر ثمانينات القرن الماضي في مركز العلوم والحاسب الألي بأمستردام. – تم الإعلان عنها لأول مره عام ١٩٩١ – تم كتابة نواتها بلغة البرمجة سي وقد سميت بايثون نسبة إلي فرقه مسرحية ببريطانيا كانت تسمي مونتي بايثون.

❖ ما هي لغة البايثون ومميزاتها واستخداماتها

تكتب بايثون باللغة العربية وهي لغة برمجة عالية المستوى ابتكرها Guido Van Rossum أثناء عمله في مركز أبحاث Informatica & Centrum Wiskunde عام 1986، عام 1991 تم نشر أول إصدار منها لتصبح في متناول الجميع.

استمر تطوير هذه اللغة وإضافة الكثير من المزايا عليها في كل إصدار جديد منها إلى يومنا هذا حتى أصبحت إحدى أهم لغات العصر والتي يمكن استخدامها لبناء برامج سطح المكتب، تطبيقات الويب، الألعاب، سكريبتات..إلخ

بايثون تعمل على جميع وأهم أنظمة التشغيل مثل Windows, Mac OS, Linux, Unix إلخ.. وتعتبر من أشهر لغات البرمجة على الإطلاق.

- **شعار لغة بايثون**

- **مميزات لغة بايثون بالنسبة للمطورين**

١. لها شعبية هائلة وهناك الكثير من المراجع لمن يريد تعلمها.

٢. بسيطة وتعلمها سهل جداً مقارنةً مع غيرها من اللغات.

٣. إذا أنشأت برنامجاً باستخدام لغة بايثون فإنه يعمل على أي نظام في العالم وهذا من أهم ما يدفعك لتعلمها.

٤. في وقتنا الحالي، تعتبر من أكثر اللغات طلباً في سوق العمل، أي إذا كنت تريد دخول سوق العمل فلغة بايثون توفر لك الكثير من الفرص.

٥. إحدى أهم اللغات التي يستخدمها المهمتين بمجال أمن المعلومات والاختراق الأخلاقي.

- **مميزات لغة بايثون عن باقي لغات البرمجة**

١. **مادياً**

لن تدفع أي مبلغ لتعمل على لغة بايثون، فهي مصدر مفتوح ومجانية وستبقى مجانية مدى الحياة.

٢. **تقنياً**

تستطيع البرمجة بها حتى ولو كان حاسوبك ضعيفاً أو قديماً.

٣. **سهولة القراءة والتعديل**

تعلمها سهل جداً ويمكن قراءة وتعديل الكود المكتوب فيها بسهولة.

٤. **العمل على أكثر من منصة**

البرنامج الذي تبنيه بواسطة لغة بايثون يعمل على كما على أهم أنظمة التشغيل مثل Windows, Mac OS, Linux, Unix.

٥. **كائنية التوجه**

تدعم مفهوم الكلاس، الكائن، التغليف، الوراثة إلخ..

٦. **تعدد المهام**

بايثون توفر لك تقنية الـ Multithreading والتي تسمح لك بجعل برنامجك قادراً على تنفيذ عدة أوامر مع بعض وبنفس الوقت.

٧. **قواعد البيانات**

بايثون توفر إنترفيسات جاهزة للتعامل مع أهم قواعد البيانات.

٨. **واجهة المستخدم**

يمكن بناء تطبيقات فيها واجهة مستخدم فيها.

٩. **التعامل مع لغات برمجة أخرى**

يمكنك التعامل مع لغات برمجة أخرى مثل (C, C++, Java) ضمن برنامجك المكتوب في الأساس بلغة بايثون.

- **تعامل لغات البرمجة الأخرى مع لغة بايثون**

أغلب لغات البرمجة تدعم التعامل مع لغة بايثون، أي أنهم يتيحون لك استخدام كود مكتوب بلغة بايثون في برامج مكتوبة في الأساس بلغات برمجة أخرى مثل (C, C++, Java)

- **المناهج التعليمية**

العديد من المعاهد والجامعات سواء كانت أجنبية أو عربية أصبحت تدرسها للطلاب.

- **مميزات بايثون بشكل موجز**

البساطة : بايثون هي واحدة من أسهل اللغات للبدء بها. أيضا ، هذه البساطة ال تحد من الميزات التي تحتاجها.

- **المكتبات والأطر :** نظرًا لشعبيتها ، تمتلك بايثون المئات من المكتبات والأطر المختلفة التي تساعد بشكل كبير في عملية التطوير الخاصة بك وتوفر الكثير من الوقت. بصفتك عالم بيانات ، ستجد أن العديد من هذه المكتبات تركز على علم البيانات والتعلم آللي.
- **مجتمع هائل :** أحد أسباب شهرة بايثون هو أنها تضم مجتمعًا كبيرًا من المهندسين والعلماء. قد تعتقد أن هذا ال ينبغي أن يكون أحد ألسباب الرئيسية الختيار بايثون ، لكن العكس هو الصحيح. إذا لم تستخدم آراء ودعم الخبراء آلخرين ، فسيكون مسار التعلم الخاص بك صعبًا
- **اهميتها في التعلم العميق :** تحتوي بايثون على العديد من الحزم مثل keras و Tensorflowو PyTorch التي تساعد علماء البيانات على تطوير خوارزميات التعلم العميق.
- **تمثيل مرئي أفضل للبيانات :** التمثيل المرئي للبيانات هو مفتاح لعلماء البيانات ألنه يساعدهم على فهم البيانات بشكل أفضل. يمكن أن تساعدك بايثون في الرسوم التوضيحية المذهلة مع مكتبات مثل ggplot و Matplotlib و NetworkX وما إلى ذلك.

- **استخدامات لغة بايثون**

تُستخدم لغة بايثون في كل المجالات، فهي لغة برمجة متعددة الأغراض، **ومن مجالات استخدامها:** تحليل البيانات، والروبوتات، وتعلم الآلة، وتطبيقاتREST ، وتطوير المواقع والألعاب، والرسوم ثلاثية الأبعاد، والأتمتة وبرمجة الأنظمة المدمجة، والكثير من المجالات الأخرى التي لا يسعنا حصرها هنا.

تستخدم الكثير من المواقع والشركات العملاقة لغة بايثون، ومنها Spotify و Google و Amazon إضافة إلى Facebook التي تستخدم بايثون لمعالجة الصور، وفي كل يوم تتحول شركات جديدة إلى استخدام بايثون، مثل Instagram التي قررت مؤخرًا استخدامها وفضلتها على PHP ،ُستخدم بايثون أيضًا من قبل بعض الجهات العلمية والبحثية، مثل وكالة الفضاء الأمريكية ناسا، والتي لها مستودع خاص بالمشاريع المُطورة ببايثون.

- **اذن استخدامات البايثون:**

تطوير الويب: باستخدام أطر مثل Django و Flask.

تحليل البيانات والعلوم: باستخدام مكتبات مثل Pandas و NumPy و Matplotlib.

التعلم الآلي: باستخدام مكتبات مثل TensorFlow و Scikit-learn.

البرمجة النصية وأتمتة المهام.

تطوير الألعاب: باستخدام مكتبات مثل Pygame

❖ تثبيت بايثون على ويندوز

يجب أن تملك جهازاً عليه نظام ويندوز ١٠ متصل بالشبكة مع صلاحيات مدير administrative access .

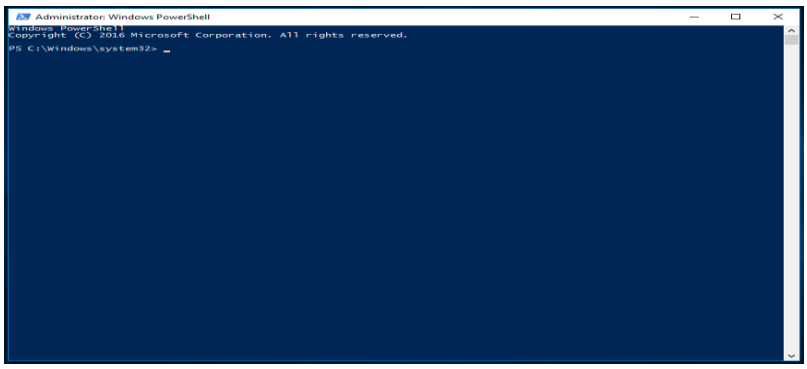
- **الخطوة الأولى: فتح وإعداد PowerShell**

سنجري معظم أطوار التثبيت والإعدادات عبر سطر الأوامر، والذي هو طريقةٌ غيرُ رسوميةٍ للتعامل مع الحاسوب، فبدلاً من الضغط على الأزرار، ستكتب نصاً وتعطيه للحاسوب لينفذه، وسيُظهِر لك ناتجاً نصياً أيضاً، يمكن أن يساعدك سطر الأوامر على تعديل أو أتمتة مختلف المهام التي تنجزها على الحاسوب يومياً، وهو أداةٌ أساسيةٌ لمطوري البرمجيات.

PowerShell هي برنامج من ميكروسوفت يوفر واجهة سطر الأوامر، يمكن إجراء المهام الإدارية عبر تنفيذ الأصناف cmdlets، والتي تُنطَق"command-lets" ، وهي أصناف متخصصة من الإطار NET ،يمكنها تنفيذ العمليات جُعِلت PowerShell مفتوحة المصدر منذ أغسطس ٢٠١٦، وصارت متوفرة الآن عبر ويندوز و أنظمة يونكس (بما في ذلك ماك ولينكس).

ستعثر على PowerShell بالنقر الأيمن على أيقونة Start في الركن الأيسر السفلي من الشاشة عندما تنبثق القائمة، انقر على"Search" ، ثم اكتب "PowerShell" في شريط البحث، عند تقديم خيارات لك انقر بالزر الأيمن على تطبيق سطح المكتب "Windows PowerShell" اختر "Run as Administrator" عندما يظهر مربع حوار يسألك"Do you want to allow this app to make changes to your PC?" ، انقر على"Yes".

بمجرد إتمام ذلك، سترى واجهة نصية تبدو كما يلي:



يمكننا تبديل مجلد النظام عن طريق كتابة الأمر التالي:

```
cd ~
```

بعد ذلك سننتقل إلى المجلد PS C:\Users\Sammy

لمتابعة عملية التثبيت، سنعد بعض الأذونات من خلال PowerShell تم إعداد PowerShell لتعمل في الوضع الأكثر أماناً بشكل افتراضي، هناك عدة مستويات للأذونات، والتي يمكنك إعدادها باعتبارك مديراً (administrator)

- **Restricted** تمثل سياسة التنفيذ الافتراضية، وبموجب هذا الوضع، لن تتمكن من تنفيذ السكربتات، وستعمل PowerShell كصدفة تفاعلية (interactive shell) وحسب.
- **AllSigned** ستمكّنك من تنفيذ جميع السكربتات وملفات الإعداد المُوقّعة من قبل جهة موثوقة، مما يعني أنه من المحتمل أن تعرّض جهازك لخطر تنفيذ سكربتات ضارة إن كانت موقعة من قبل جهة موثوقة.
- **RemoteSigned** ستمكّنك من تنفيذ السكربتات وملفات الإعداد المُنزّلة من الشبكة، والمُوقعة من قبل جهة موثوقة، مما يعني أنه من المحتمل أن تعرّض جهازك لخطر تنفيذ سكربتات ضارة إن كانت تلك السكربتات الموثوقة ضارة.
- **Unrestricted** تسمح بتنفيذ جميع السكربتات وملفات الإعداد المُنزّلة من الشبكة بمجرد أن تؤكد أنك تدرك أنّ الملف مُنزّل من الشبكة، في هذه الحالة التوقيعات الرقمية غير لازمة، مما يعني أنه من المحتمل أن تعرّض جهازك لخطر تنفيذ سكربتات غير موثوقة منزلة من الشبكة قد تكون ضارة.

سنستخدم سياسة التنفيذ RemoteSigned لتعيين الإذن للمستخدم الحالي، وهكذا سنسمح لبرنامج PowerShell بقبول السكربتات المُنزّلة التي نثق بها، ودون خفض كل دفاعاتنا وجعل الأذونات هشة كما هو الحال مع سياسة التنفيذ Unrestricted **سنكتب في PowerShell**

```
Set-ExecutionPolicy -Scope CurrentUser
```

ستطالبك PowerShell بتحديد سياسة التنفيذ، وبما أننا نريد استخدام RemoteSigned ، **فسنكتب:**

```
RemoteSigned
```

بمجرد الضغط على الزر enter، ستُسأل عما إن كنت تريد تغيير سياسة التنفيذ، اكتب الحرف y لاختيار "نعم"، واعتماد التغييرات، يمكننا التحقق من نجاح العملية عن طريق طلب الأذونات الحالية في الجهاز عبر كتابة:

```
Get-ExecutionPolicy -List
```

ستحصل على مخرجات مشابهة لما يلي:

```
        Scope ExecutionPolicy
                                                  ----- ---------------
MachinePolicy       Undefined
   UserPolicy       Undefined
      Process       Undefined
  CurrentUser    RemoteSigned
 LocalMachine       Undefined
```

هذا يؤكد أنّ المستخدم الحالي يمكنه تنفيذ السكربتات الموثوقة التي تم تنزيلها من الشبكة، يمكننا الآن تنزيل الملفات التي سنحتاج إليها لإعداد بيئة برمجة بايثون.

الخطوة الثانية: تثبيت Chocolatey

مدير الحزم (package manager)هو مجموعة من أدوات البرمجيات التي تعمل على أتمتة عمليات التثبيت، بما في ذلك التثبيت الأولي للبرامج، وترقيتها، وإعدادها، وإزالتها عند الحاجة، تحفظ هذه الأدوات التثبيتات في موقع مركزي، ويمكنها صيانة جميع حزم البرامج على النظام وفق تنسيقات (formats) معروفة.

Chocolatey هي مدير حزم تعمل من سطر الأوامر، تم تصميمها لنظام ويندوز، وتحاكي apt-get الخاصة بلينكس، متوفرة كإصدار مفتوح المصدر، ويمكنها مساعدتك Chocolatey على تثبيت التطبيقات والأدوات بسرعة، سنستخدمها لتنزيل ما نحتاج إليه لبيئتنا التطويرية.

قبل تثبيت السكربت، دعنا نقرأه للتأكد من أنّ التغييرات التي سيجريها على الجهاز مقبولة، سنستخدم إطار العمل NET، لتنزيل وعرض السكربت Chocolatey في نافذة الطرفية، سننشئ كائنًا WebClient يُسمى script$ (يمكنك تسميته كما تريد طالما ستستخدم المحرف $ في البداية)، والذي يشارك إعدادات الاتصال بالشبكة مع المتصفح Internet Explorer:

```
$script = New-Object Net.WebClient
```

سنحصل على المخرجات التالية:

```
. .
DownloadFileAsync          Method     void DownloadFileAsync(uri ad-
dress, string fileName), void DownloadFileAsync(ur...
DownloadFileTaskAsync      Method     System.Threading.Tasks.Task
DownloadFileTaskAsync(string address, string fileNa...
DownloadString             Method     string DownloadString(string ad-
dress), string DownloadString(uri address) # التابع الذي سنستخدمه
DownloadStringAsync        Method     void DownloadStringAsync(uri ad-
dress), void DownloadStringAsync(uri address, Sy...
DownloadStringTaskAsync    Method     Sys-
tem.Threading.Tasks.Task[string] DownloadStringTaskAsync(string ad-
dress), Sy…
 . . .
```

عند النظر إلى المخرجات، يمكننا تحديد التابع DownloadString الذي يمكننا استخدامه لعرض محتوى السكربت والتوقيع في نافذة PowerShell **كما يلي:**

```
$script.DownloadString("https://chocolatey.org/install.ps1")
```

بعد مطالعة السكربت، يمكننا تثبيت Chocolatey عن طريق كتابة ما يلي في PowerShell

```
iwr https://chocolatey.org/install.ps1 -UseBasicParsing | iex
```

تسمح لنا iwr أو Invoke-WebRequest التي تخص cmdlet باستخراج البيانات من الشبكة، سيؤدي هذا إلى تمرير السكربت إلى iex أو Invoke-Expression، والذي سينفذ محتويات السكربت، وتنفيذ سكربت التثبيت لمدير الحزم Chocolatey.

اسمح لبرنامج PowerShell بتثبيت Chocolatey بمجرد تثبيته بالكامل، يمكننا البدء في تثبيت أدوات إضافية باستخدام الأمر choco.

إن احتجت إلى ترقية Chocolatey مستقبلاً، **يمكنك تنفيذ الأمر التالي:**

```
choco upgrade chocolatey
```

بعد تثبيت مدير الحزم، يمكننا متابعة تثبيت ما نحتاجه لبيئة البرمجة خاصةً.

- **المرحلة الثالثة: تثبيت محرر النصوص nano (اختياري)**

سنثبّت الآن nano ، وهو محرر نصوص يستخدم واجهة سطر الأوامر، والذي يمكننا استخدامه لكتابة البرامج مباشرة داخل PowerShell هذه ليست خطوة إلزامية، إذ يمكنك بدلاً من ذلك استخدام محرر نصوص بواجهة مستخدم رسومية مثل Notepad، لكن ميزة nano أنه سيُعوِّدك على استخدام PowerShell.

دعنا نستخدم Chocolatey لتثبيت nano

```
choco install -y nano
```

الخيار y-يعني أنَّك توافق على تنفيذ السكربت تلقائيًا دون الحاجة إلى تأكيد.

بعد تثبيتnano، سنكون قادرين على استخدام الأمر nano لإنشاء ملفات نصية جديدة، وسنستخدمه بعد حين لكتابة أول برامجنا في بايثون.

- **المرحلة الرابعة: تثبيت بايثون 3**

مثلما فعلنا مع nano أعلاه، سنستخدم Chocolatey لتثبيت بايثون 3:

```
choco install -y python3
```

ستثبّت PowerShell الآن بايثون ٣، مع عرض بعض المخرجات أثناء العملية، بعد اكتمال العملية، سترى المخرجات التالية:

```
Environment Vars (like PATH) have changed. Close/reopen your shell to
 See the changes (or in powershell/cmd.exe just type 'refreshenv').
The install of python3 was successful.
 Software installed as 'EXE', install location is likely default.

Chocolatey installed 1/1 packages. 0 packages failed.
 See the log for details
(C:\ProgramData\chocolatey\logs\chocolatey.log).
```

بعد الانتهاء من التثبيت، ستحتاج إلى التحقق من أنّ بايثون مثبّتة وجاهزة للعمل، لرؤية التغييرات، استخدم الأمر refreshenv أو أغلق PowerShell ثم أعد فتحها بصلاحيات مدير النظام، **ثم تحقق من إصدار بايثون على جهازك:**

```
python -V
```

ستحصل على مخرجات في نافذة الطرفية والتي ستريك إصدار بايثون المثبّت.

```
Python 3.7.0
```

سيتم تثبيت، إلى جانب بايثون الأداة pip، وهي أداةٌ تعمل مع لغة بايثون تُثبّت وتدير الحزم البرمجية التي قد نحتاج إلى استخدامها في تطوير مشاريعنا.

سنحدّث pip عبر الأمر التالي:

```
python -m pip install --upgrade pip
```

يمكننا استدعاء بايثون من Chocolatey عبر الأمر python سنستخدم الراية m- لتنفيذ الوحدة كأنها سكربت، وإنهاء قائمة الخيارات، ومن ثمّ نستخدم pip لتثبيت الإصدار الأحدث.

بعد تثبيت بايثون وتحديث pip، فنحن جاهزون لإعداد بيئة افتراضية لمشاريع التطوير خاصةً.

- **الخطوة الخامسة: إعداد بيئة افتراضية**

الآن بعد تثبيت Chocolatey و nano وبايثون، يمكننا المضي قدمًا لإنشاء بيئة البرمجة خاصةً عبر الوحدة venv.

تُمكّنك البيئات الافتراضية من إنشاء مساحة معزولة في حاسوبك مخصصة لمشاريع بايثون، مما يعني أنّ كل مشروع تعمل عليه ستكون له اعتماديته (dependencies) الخاصة به، والتي لن تؤثّر على غيره من المشاريع.

يوفّر لنا ضبط بيئة برمجية تحكماً أكبر بمشاريع بايثون، وإمكانية التعامل مع إصدارات مختلفة من حزم بايثون، وهذا مهمٌ كثيرًا عندما تتعامل مع الحزم الخارجية.

يمكنك ضبط أيّ عدد تشاء من البيئات الافتراضية، وكل بيئة ستكون ممثلة بمجلد في حاسوبك يحتوي على عدد من السكربتات.

اختر المجلد الذي تريد أن تضع فيه بيئات بايثون، أو يمكنك إنشاء مجلد جديد باستخدام الأمر mkdir كما يلي:

```
mkdir environments
cd environments
```

بعد أن انتقلتَ إلى المجلد الذي تريد احتواء البيئات فيه، تستطيع الآن إنشاء بيئة جديدة بتنفيذ الأمر التالي:

```
python -m venv my_env
```

استخدام الأمر python، سننفّذ الوحدة venv لإنشاء البيئة الافتراضية التي أطلقنا عليها في هذه الحالة my_env.

ستنشئ venv مجلدًا جديدًا يحتوي على بعض العناصر التي يمكن عرضها باستخدام الأمر ls

```
ls my_env
```

سنحصل على المخرجات التالية:

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         8/22/2016   2:20 PM                Include
d-----         8/22/2016   2:20 PM                Lib
d-----         8/22/2016   2:20 PM                Scripts
-a----         8/22/2016   2:20 PM            107 pyvenv.cfg
```

تعمل هذه الملفات مع بعضها لضمان أنَ تكون مشاريعك معزولةٌ عن سياق الآلة المحلية، لكي لا تختلط ملفات النظام مع ملفات المشاريع، وهذا أمرٌ حسنٌ لإدارة الإصدارات ولضمان أنَّ كل مشروع يملك وصولًا إلى الحزم التي يحتاجها.

عليك تفعيل البيئة لاستخدامها، وذلك بكتابة الأمر التالي الذي سيُنفّذ سكربت التفعيل في المجلد Scripts

```
my_env\Scripts\activate
```

يجب أن تظهر الآن سابقةٌ (prefix) في البحث (prompt) والتي هي اسم البيئة المستخدمة، وفي حالتنا هذه يكون اسمها my_env

```
(my_env) PS C:\Users\Sammy\Environments>
```

تتيح لنا هذه البادئة معرفة أنّ البيئة my_env مفعّلة حالياً، وهذا يعني أننا لن سنستخدم إلا إعدادات وحزم هذه البيئة عند إنشاء مشاريع جديدة.

- **الخطوة الثالثة: إنشاء برنامج بسيط**

بعد أن أكملنا ضبط بيئتنا الافتراضية، لننشِئ برنامجاً بسيطاً يعرض العبارة «مرحبا بالعالم!»، وبهذا سنتحقق من أنَّ البيئة تعمل بالشكل الصحيح، ولكي تتعوّد على إنشاء برامج بايثون إن كنت وافداً جديداً على اللغة.

علينا أولاً تشغيل المحرر nano وإنشاء ملف جديد:

```
(my_env) PS C:\Users\Sammy> nano hello.py
```

بعد فتح الملف في نافذة الطرفية، سنكتب البرنامج الخاص بنا:

```
print("مرحبا بالعالم!")
```

أغلق محرر nano بالضغط على Ctrl+x ثم اضغط على y عندما يسألك عن حفظ الملف.

بعد أن يُغلَق المحرر nano وتعود إلى سطر الأوامر، حاول تنفيذ البرنامج:

```
(my_env) PS C:\Users\Sammy> python hello.py
```

سيؤدي برنامج hello.py الذي أنشأتَه إلى طباعة الناتج التالي في الطرفية:

```
مرحبا بالعالم!
```

للخروج من البيئة، اكتب الأمر deactivate وستعود إلى مجلدك الأصلي.

❖ محررات الأكواد

يسمى محرر الأكواد البرمجية IDE وهي اختصار لعبارة Integrated Development Environment وهو عبارة عن برنامج مخصص لتطوير البرمجيات أي يقدم مجموعة من الأدوات التي تساعد في هذه العملية، **والتي غالبا ما تتضمن ما يلي:**

١. محرر لمعالجة الأكواد البرمجية (مع ميزات مهمة مثل الإكمال التلقائي أو تعليم الكلمات المفتاحية وغيرها من الميزات).

٢. بناء التطبيقات وتنفيذها وكذلك أدوات تصحيح الأخطاء.

٣. ميزات أخرى إضافية (مثل نشر التطبيق أو رفعه إلى المتجر وغيرها).

تدعم أغلب المحررات لغات برمجة مختلفة فلا تقتصر على لغة برمجية واحدة فقط وكذلك تحتوي على العديد من الميزات وبالتالي نجد أنفسنا أمام بحر من الخيارات فيما يخص محررات الأكواد البرمجية فلنتأنى قليلاً ولنحاول اختيار المناسب لنا.

وعلى الرغم مما سبق فإن بعض المبرمجين يفضلون محررات الأكواد البرمجية البسيطة لسهولة التعامل معها ولأنها تقدم الجوهر الأساسي وهم معالجة الكود وتنفيذه.

بعض محررات الأكواد الشائعة:

PyCharm: محرر قوي مع ميزات متقدمة، مفيد للمشاريع الكبيرة.

Visual Studio Code: محرر خفيف الوزن وقابل للتخصيص مع دعم كبير للامتدادات.

Sublime Text: محرر نصوص سريع وخفيف الوزن.

Atom: محرر نصوص مفتوح المصدر وقابل للتخصيص.

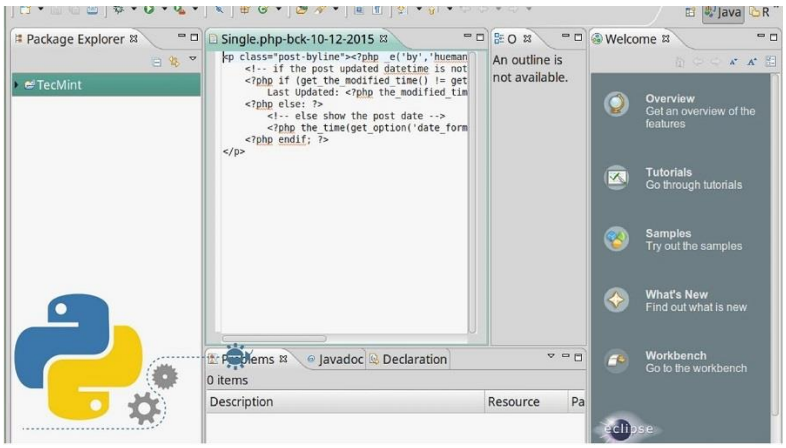
Jupyter Notebook: ممتاز لتحليل البيانات والتعلم الآلي.

- **المحررات البرمجية التي تدعم لغة البرمجة بايثون**
  - **Eclipse+PyDev**

إذا كنت من محبي البرمجيات والبرامج مفتوحة المصدر فلابد أنك سمعت ببرنامج Eclipse حتماً فهو أشهر من نار على علم.

يتوافر من هذا البرنامج إصدارات عديدة لأغلب أنظمة التشغيل مثل Linux,Windows وكذلك Mac Os ، وعلى الرغم من انه محرر رئيسي للغة البرمجة الشهيرة جافا فإنه يتمتع بميزة مهمة جداً وهي قابلية إضافة العديد من الإضافات تسمى (Plugins) التي تجعل منه مناسب للغات برمجية أخرى.

من أهم هذه الإضافات PyDev وهي تسمح لبرنامج Eclipse بالتعامل مع لغة البرمجة بايثون من كتابة أكود وتصحيح أخطاء وعمليات الإكمال التلقائي وغيرها من الميزات الهامة.



كما أن تنصيب هذه الإضافة سهل جداً ووفق ما يلي:

من القائمة Help نختار Eclipse Marketplace ومن ثم نبحث عن PyDev ونضغط install ومن ثم نعيد تشغيل البرنامج ويصبح جاهزاً للاستخدام كمحرر للغة البرمجة بايثون.

وبالحديث عن السلبيات والإيجابيات للبرنامج السابق:

- **الإيجابيات**

إذا كنت تملك Eclipse على جهازك فلن تحتاج عملية إعداده لاستخدامه كمحرر لغة البرمجة بايثون الكثير من الوقت.

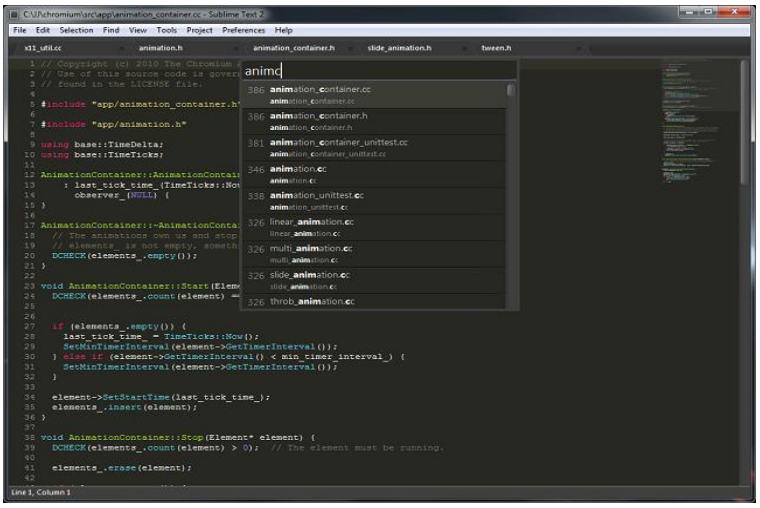
- **السلبيات**

إذا كنت جديد على البرمجة فيعتبر استخدام برنامج Eclipse للتعامل مع البايثون مزعجاً قليلاً وتحتاج إلى بعض الوقت لتتعود عليه.

- **Sublime Text**

تم برمجة هذا التطبيق من قبل مهندس في غوغل على أمل أن يصبح محرر أكواد مميز وهو في الحقيقة يمتلك شعبية وشهرة كبيرة ويدعم جميع أنظمة التشغيل المتاحة اليوم.

يملك محرر Sublime Text دعم مسبقاً للغة البرمجة بايثون ليس هذا فحسب، بل يملك مجموعة من الحزم (Packages) التي تقدم ميزات إضافية داعمة لها.



ولكن يمكن القول بأن عملية إضافة الحزم الجديدة إليه قد تكون مربكة قليلاً للمستخدم ومما يميز هذا المحرر حقيقة أن كل الحزم الخاصة به مبرمجة بلغة البرمجة بايثون ويتطلب إضافة الحزم كتابة مجموعة من الأسطر البرمجية (script) ضمنه بشكل مباشر.

بالانتقال إلى سلبيات هذا المحرر وإيجابياته يمكن القول:

- **الإيجابيات**

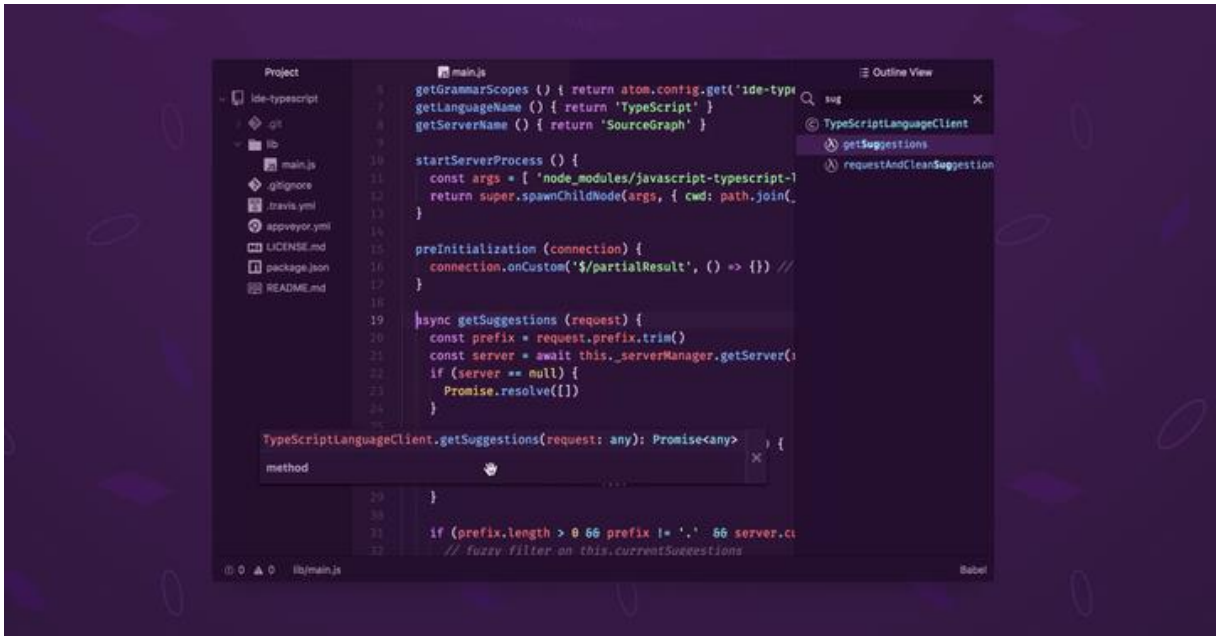
هو برنامج بسيط وسهل الاستخدام وسريع ويملك دعم واسع ضمن مجتمع مبرمجي بايثون.

- **السلبيات**

في الحقيقة هو برنامج غير مجاني، ولكن يتوفر منه نسخة تجريبية وهذا مزعج لبعض المبرمجين ومكلف قليلاً.

- **Atom**

هذا البرنامج متاح لجميع أنظمة التشغيل ويمكن القول بأنه محرر مثالي يملك واجهة سهلة ومريحة ومتصفح ملفات داخلي وكذلك سوق لشراء الإضافات.



ومن الإضافات السابقة هناك إضافة خاصة بلغة البرمجة بايثون يمكن تثبيتها بسهولة.

- **الإيجابيات**

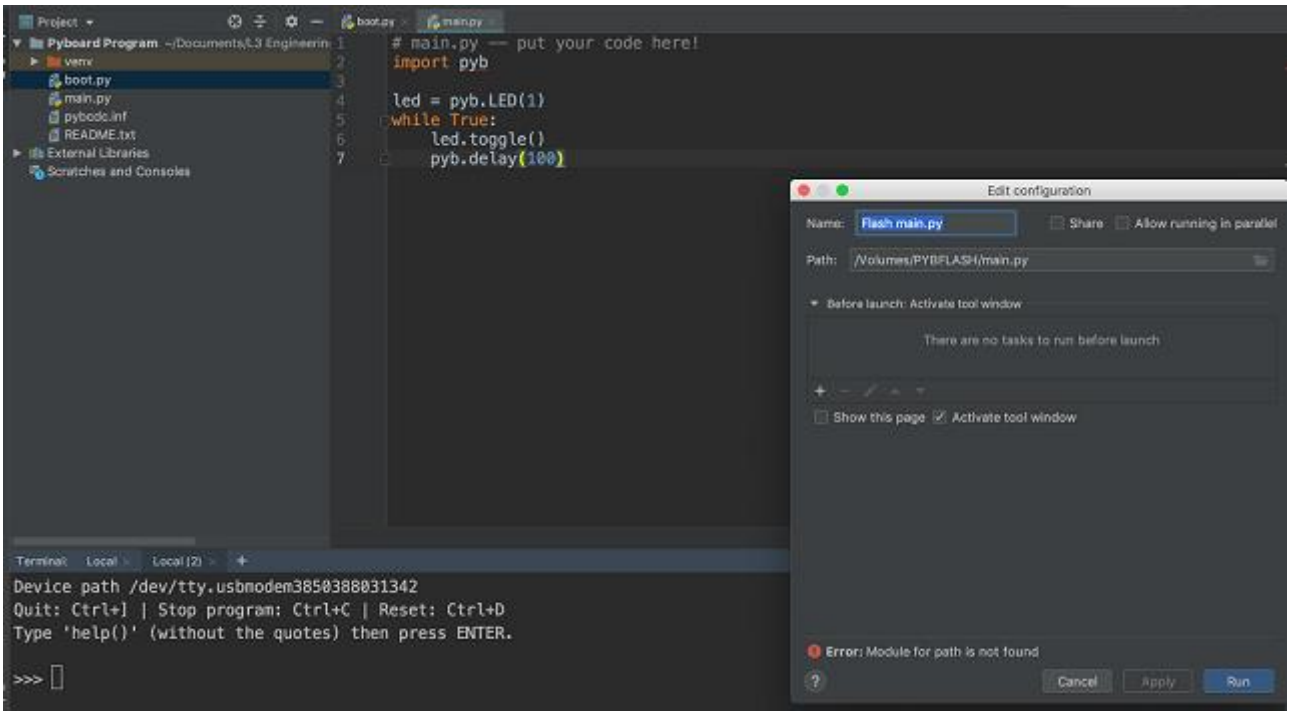
خفيف الحجم سهل التحميل والتثبيت ويملك دعم متكامل لجميع أنظمة التشغيل.

- **السلبيات**

هو تطبيق مصمم باستخدام جافا سكريبت وبالتالي سيعمل دائماً على تحويل الكود وتنفيذه كجافا سكريبت.

PyCharm

يمكن القول بأنه هذا البرنامج هو البرنامج الأشهر والمليء بالميزات والدعم المطلق للغة البرمجة بايثون.



يتوفر من هذا التطبيق نسختين الأولى المدفوعة وهي النسخة الاحترافية والثانية نسخة مجانية تسمى Community Edition

تعتبر عملية تنصيبه عملية سهلة ومريحة جداً ويوجد نسخة خاصة بمعظم أنظمة التشغيل : Windows,Linux و Mac Os .

الإيجابيات

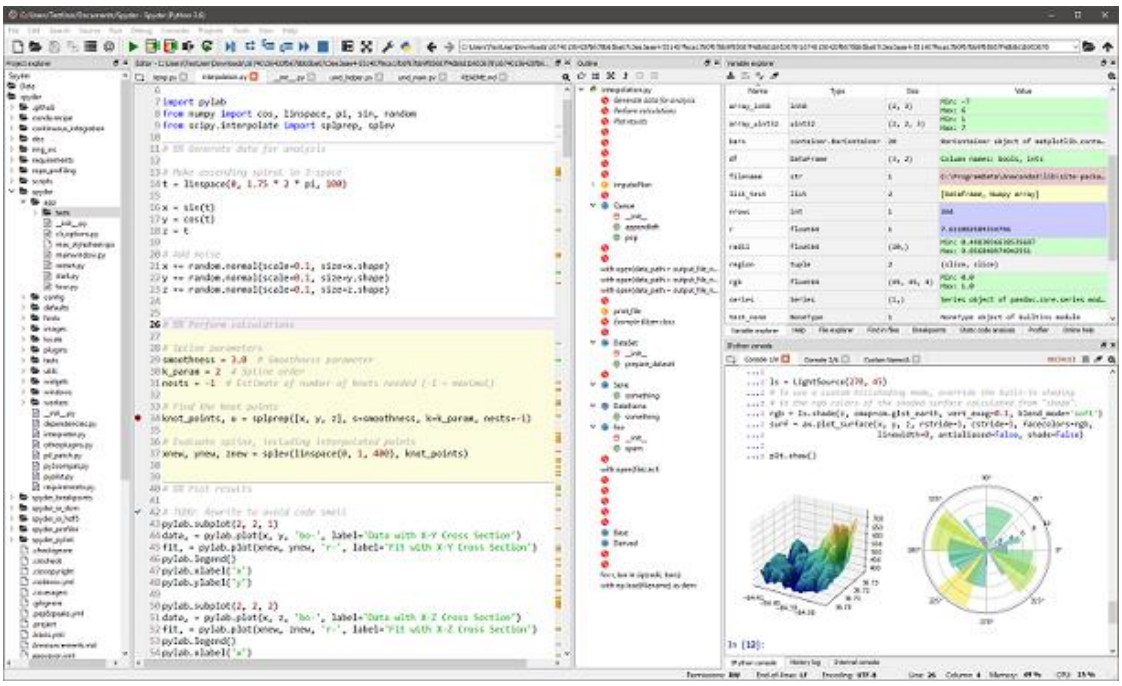
يعتبر هذا البرنامج بيئة متكاملة للغة البرمجة بايثون مع دعم كبير وخارق للغة فهو يمكنك من الكتابة والتنفيذ والتصحيح.

السلبيات

يعاني هذا التطبيق من القليل من البطء في التحميل ويحتاج أحيانا إلى تغيير بعض الإعدادات الافتراضية.

- **Spyder**

هو برنامج مجاني مفتوح المصدر يستخدم كمحرر للغة البرمجة بايثون لأغراض البحث العلمي وما يتعلق بها.



ما يميز هذا البرنامج أنه موجه للباحثين في علوم البيانات باستخدام لغة البرمجة بايثون حيث يأتي مزودا بالعديد من المكتبات التي تدعم ذلك.

- **الإيجابيات**

أنه يدعم بحوث علوم البيانات.

- **السلبيات**

يمكن النظر إليه بأنه ليس متكامل تماماً ليدعم لغة البرمجة بايثون بالشكل الأمثل.

المحرر الأمثل للاستخدام: برنامج PyCharm المدفوع إن أمكن وإن لم يكن ذلك متاحاً فالبرنامج بنسخة Community هو رائع جدا ومناسب.

❖ بيئة Python 's IDLE

تأتي كل عملية تثبيت للغة Python مزودة ببيئة تطوير وتعلم متكاملة، والتي ستشاهدها مختصرة إلى IDLE أو حتى IDE هذه فئة من التطبيقات التي تساعدك على كتابة التعليمات البرمجية بشكل أكثر كفاءة، في حين أن هناك العديد من IDEs لتختار من بينها، فإن Python IDLE هي عظام مكشوفة للغاية، مما يجعلها الأداة المثالية للمبرمجين المبتدئين.

يتم تضمين Python IDLE في عمليات تثبيت Python على نظامي التشغيل Windows و Mac إذا كنت من مستخدمي Linux ، فيجب أن تكون قادرًا على العثور على Python IDLE وتنزيله باستخدام مدير الحزم الخاص بك، بمجرد تثبيته يمكنك بعد ذلك استخدام Python IDLE كمترجم تفاعلي أو كمحرر ملف.

- **مترجم تفاعلي:** أفضل مكان لتجربة كود بايثون هو المترجم التفاعلي، والمعروف باسم القشرة (shell)، القشرة عبارة عن حلقة أساسية للقراءة والتقييم والطباعة (REPL) يقرأ بيان بايثون، ويقيم نتيجة ذلك البيان، ثم يطبع النتيجة على الشاشة ثم يعود مرة أخرى لقراءة البيان التالي.

تعد قشرة Python مكاناً ممتازاً لتجربة مقتطفات التعليمات البرمجية الصغيرة، يمكنك الوصول إليه من خلال Terminal أو تطبيق سطر الأوامر على جهازك، يمكنك تبسيط سير عملك باستخدام Python IDLE ، والذي سيبدأ فوراً في تشغيل Python shell عند فتحه.

- **محرر ملف:** يحتاج كل مبرمج إلى أن يكون قادراً على تحرير الملفات النصية وحفظها، برامج Pythonهي ملفات بامتداد py تحتوي على سطور من كود Python يمنحك Python IDLE القدرة على إنشاء هذه الملفات وتحريرها بسهولة.

يوفر Python IDLE أيضاً العديد من الميزات المفيدة التي ستراها في IDEs الاحترافية، مثل تمييز البنية الأساسية وإكمال التعليمات البرمجية والمسافة البادئة التلقائية IDEs المهنية هي أجزاء أكثر قوة من البرامج ولديها منحنى تعليمي حاد، إذا كنت قد بدأت للتو رحلة برمجة Python ، فإن Python IDLE هو بديل رائع.

❖ فهم نافذة IDLE

١. بدء تشغيل - Windows كافة البرامج - IDLE - <version> - Python

٢. النافذة الرئيسية

بعد تشغيل Python IDLE ،فإن أول ما تراه هو النافذة الرئيسية، هذه النافذة الرئيسية عبارة عن مترجم لغة Python ، أي أنك تدخل عبارة Python في هذا المترجم، ثم اضغط علىEnter ، وسيقوم برنامج Python بتنفيذ العبارة على الفور يسمى هذا المترجم أيضاً شل.

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (I
Type "help", "copyright", "credits" or "license()" for more information.
>>> print ("Hello world!")
Hello world!
>>>
```

```
>>> for i in range(3):
    print(i)


0
1
2
>>>
```

٣. تحرير الملف

في النافذة الرئيسية، حدد القائمة ملف / ملف جديد لفتح نافذة تحرير جديدة، اكتب برنامج Python واختر حفظ.

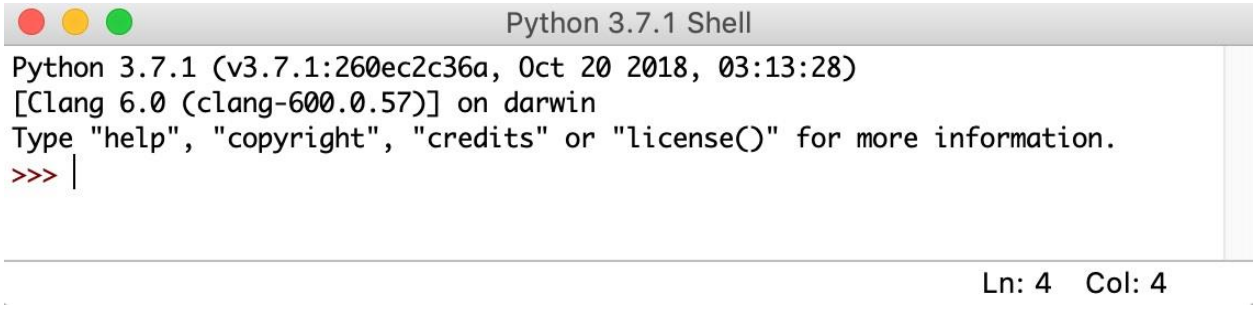
```
for i in range(3):
    print (i)
```

٤. قم بتشغيل الملف

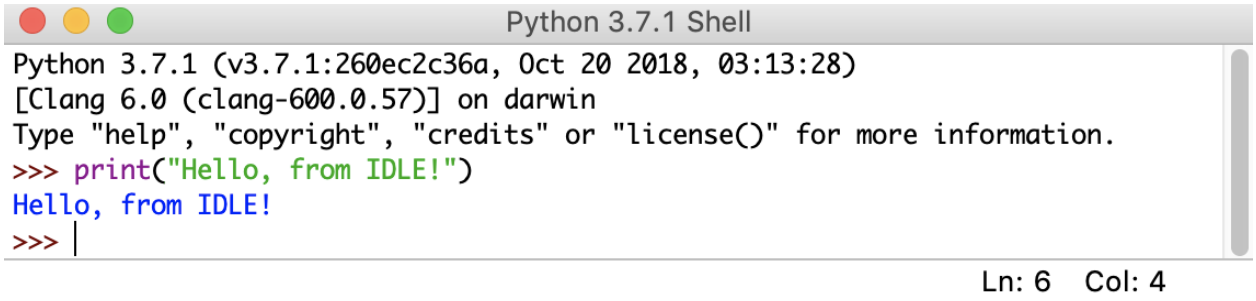
بعد حفظ الملف، يمكنك تشغيله مباشرة في نافذة التحرير في برنامج Python اضغط F5 ، أو حدد قائمة تشغيل / تشغيل الوحدة النمطية، يتم تشغيل البرنامج، يمكن رؤية نتيجة العملية في النافذة الرئيسية.

❖ استخدام python shell

القشرة هي الوضع الافتراضي للتشغيل في Python IDLE عندما تضغط على الأيقونة لفتح البرنامج، فإن الصدفة هي أول ما تراه:



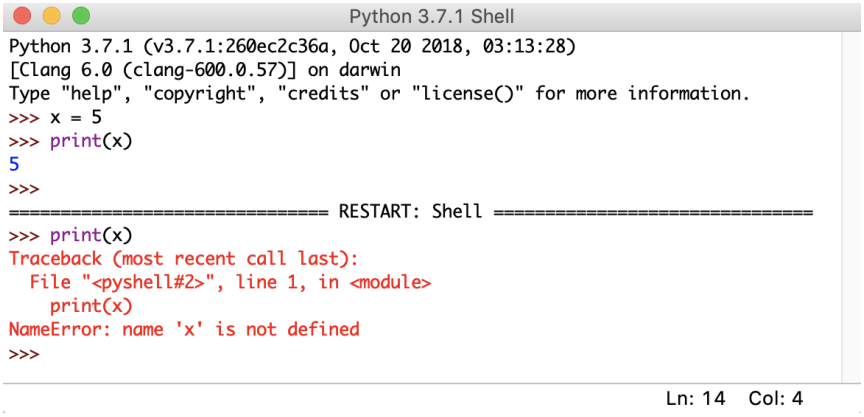
هذه نافذة فارغة لمترجم بايثون يمكنك استخدامه لبدء التفاعل مع Python على الفور، يمكنك اختباره بسطر قصير من التعليمات البرمجية:



هنا، استخدمت (print) لإخراج السلسلة "Hello,from IDLE!"على شاشتك، هذه هي الطريقة الأساسية للتفاعل مع Python IDLE ،تكتب أوامر واحداً تلو الآخر وتستجيب Python بنتيجة كل أمر.

بعد ذلك، ألق نظرة على شريط القوائم سترى بعض الخيارات لاستخدام القشرة:

يمكنك إعادة تشغيل قذيفة من هذه القائمة، إذا حددت هذا الخيار، فسوف تمسح حالة الغلاف، سيتصرف كما لو كنت قد بدأت نسخة جديدة من Python IDLE سوف تنسى القشرة كل شيء من حالتها السابقة:



في الصورة أعلاه، تقوم أولاً بتعريف متغير، x = 5 عند استدعاء print (x) ، يُظهر الغلاف الإخراج الصحيح ، وهو الرقم 5 ومع ذلك ، عند إعادة تشغيل shell ومحاولة استدعاء print (x) مرة أخرى ، يمكنك أن ترى أن الغلاف يقوم بطباعة traceback هذه رسالة خطأ تقول أن المتغير x غير معرف، نسيت القشرة كل ما جاء قبل إعادة تشغيلها.

يمكنك أيضاً مقاطعة تنفيذ الصدفة من هذه القائمة، سيؤدي هذا إلى إيقاف أي برنامج أو بيان يعمل في shell في وقت الانقطاع، ألق نظرة على ما يحدث عندما ترسل مقاطعة لوحة المفاتيح إلى الغلاف:

```
>>> while True:
        print('infinite')


infinite
infinite
infinite
infinite
infinite
infinite
infinite
infinite
infinite
infinite
infinite
infinite
infinite
infinite
infinite
infiniteTraceback (most recent call last):
  File "<pyshell#2>", line 2, in <module>
    print('infinite')
KeyboardInterrupt
```

يتم عرض رسالة خطأ KeyboardInterrupt بنص أحمر أسفل النافذة، تلقى البرنامج المقاطعة وتوقف عن التنفيذ.

لإتقان مكتبة Python القياسية، من المهم قراءة وثائقها الرسمية، هذه المقالة ليست نسخة من هذا المستند، ولكنها ملخص لكل مكتبة ووظائفها الرئيسية، لذلك سوف تعرف المكتبة التي ستستخدمها.

- **معالجة النصوص**

  - سلسلة: توفر مجموعات الأحرف: ascii_lowercase ، ascii_uppercase.
  - re دعم التعبير العادي (النمط ، السلسلة): تطابق ، بحث ، findall، فرعي ، انقسام ، باحث.
  - difflib مقارنة تفاضلية للتتابعات:(s1 , s2) context_diff.

- **هيكل البيانات**

  - datetime: تاريخ العملية ، يمكنك استخدام السهم بدلاً من ذلك.
  - التقويم: التقويم: التقويم.
  - المجموعات: هياكل البيانات الأخرى: deque ، Counter ، defaultdict ، المسماة tuple .
  - Heapq: تنفيذ فرز كومة الذاكرة المؤقتة nlargest ، nsmallest، دمج.
  - Bisect: تنفيذ البحث الثنائي bisect ، insort .
  - الصفيف: تحقيق التسلسل: الصفيف.
  - نسخة: نسخة ضحلة ونسخة عميقة: نسخة، نسخ عميق.
  - Pprint: إخراج النص بشكل جميل pprint.
  - تعداد: تنفيذ فئة التعداد: تعداد.

- **الرياضيات**

  - الرياضيات: مكتبة وظائف الرياضيات، هناك العديد من الوظائف، لذلك لم يتم سردها واحدة تلو الأخرى.
  - الكسور: عملية الكسر: الكسر كـ F .
  - عشوائي: رقم عشوائي: الاختيار، randint، randrange، عينة ، خلط ، غاوس
  - الإحصائيات: الوظائف الإحصائية: المتوسط ، الوسيط، الوضع، التباين.

- **البرمجة الوظيفية**

- أدوات itertools: أدوات المكرر: التباديل، التوليفات، المنتج، السلسلة، التكرار، الدورة، التراكم.
- functools:أدوات الوظائف wraps:تقليل ، جزئي ، singledispatch،lrucache.
- عامل التشغيل: عامل التشغيل الأساسي.

- **الوصول إلى دليل الملف**

- pahlib: كائن المسار: المسار.
- Fileinput: قراءة ملف واحد أو أكثر وخطوط المعالجة: الإدخال.
- Filecmp: مقارنة ما إذا كان الملفان متماثلان cmp.
- glob: file wildcard: glob
- linecache: قراءة خطوط الملفات وتحسين ذاكرة التخزين المؤقت getline و getlines
- إغلاق: عمليات الملف: نسخ، نسخه، رمتري، نقل.

- **استمرارية البيانات**

- مخلل: تسلسل مخلل الملف: تفريغ، مقالب، حمل، أحمال.
- sqlite3: عذر قاعدة بيانات sqlite.

- **تنسيق الملف**

- Csv: معالجة ملفات csv: القارئ ، رأس الكتابة ، الكاتب.
- Configparser: معالجة ملفات التكوين: get ، ConfigParset، المقاطع.

- **التشفير**

- hashlib: خوارزمية تشفير التجزئة: sha256 ، سداسي الأضلاع.

- **نظام التشغيل**

- os: نظام التشغيل ، انظر الوثائق.
- io: قراءة وكتابة str و bytes في الذاكرة: StringIO و BytesIO والكتابة وget_value
- الوقت: المؤقت: الوقت والنوم ووقت الاستراحة.
- argparse ، getopt: معالجة سطر الأوامر ، يوصى باستخدام click أو docopt.
- تسجيل: تسجيل: تصحيح، معلومات، تحذير، خطأ، حرج.
- Getpass: الحصول على كلمة المرور التي أدخلها المستخدم getpass.
- النظام الأساسي: يوفر دعماً عبر الأنظمة الأساسية: uname ، system.

- **التنفيذ المتزامن**

- الترابط: نموذج multithreading: الموضوع ، بدء ، الانضمام.
- المعالجة المتعددة: نموذج متعدد العمليات: تجمع، خريطة، عملية.
- concurrent.futures: نموذج تنفيذ غير متزامن ThreadPoolExecutor ، ProcessPoolExecutor.
- عملية فرعية: إدارة العمليات الفرعية: تشغيل.
- جدولة: أداة الجدولة.
- قائمة الانتظار: قائمة انتظار متزامنة: قائمة الانتظار.

- **التواصل والتشبيك بين العمليات**

- مأخذ التوصيل: مأخذ التوصيل، شائع الاستخدام في تطوير الخادم.
- SSL: وصول HTTPS.
- حدد، محدد I / O معدد.
- غير متزامن: IO غير متزامن ، run_until_complete، coroutine: get_event_loop ، eventloop انتظر ، غير متزامن ، انتظر.

- **معالجة بيانات الشبكة**

- البريد الإلكتروني: التعامل مع البريد الإلكتروني.
- Json: مقبض json: مقالب ، أحمال.
- base64: معالجة ترميز b64encode , b64decode: base64 .

- **أداة لغة الترميز المنظمة**

- html: escape html:escape, unescape
- html.parser : دعم بروتوكول الشبكة
- متصفح الويب: متصفح مفتوح: مفتوح
- Wsgiref: تنفيذ واجهة WSGI
- Uuid: رمز التعريف الفريد العالمي :uuid1 و uuid3 و uuid4 وuuid5
- ftplib، poplib، imaplib، nntplib، smptlib، telnetlib :تنفيذ بروتوكولات شبكة مختلفة.

يتم استبدال المكتبات المتبقية بالطلبات.

- **إطار البرنامج**

- سلحفاة: أداة رسم.
- كمد: تنفيذ قذيفة تفاعلية.
- Shlex استخدم صيغة shell لتقسيم الأوتار: انقسام.
- GUI.
- tkinter.

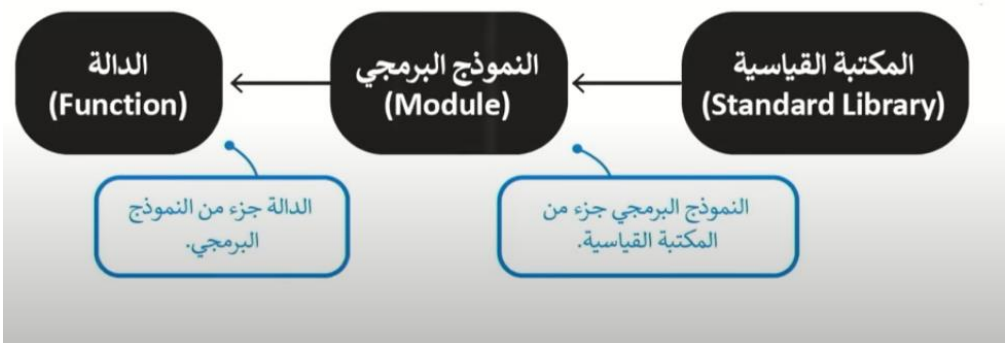
❖ المكتبة القياسية

المكتبات البرمجية عبارة عن مجموعة من التعليمات البرمجية المدمجة سابقا في المكتبات من الموارد القابلة للاستخدام في أي برنامج لأنها مستقلة عن البرامج التي يتم كتابتها.

ويشير مصطلح النموذج البرمجي module إلى الحزمة من الملفات تحتوي على مقاطع برمجية يتم استيرادها إلى البرنامج لتنفيذ وظائف مختلفة ويكون امتدادها عادةً " PY " مثل نموذج برمجي واجهة المستخدم الرسومية tkinter module ونموذج برمجي معرفة خصائص الحاسب ونظام التشغيل Platform module.

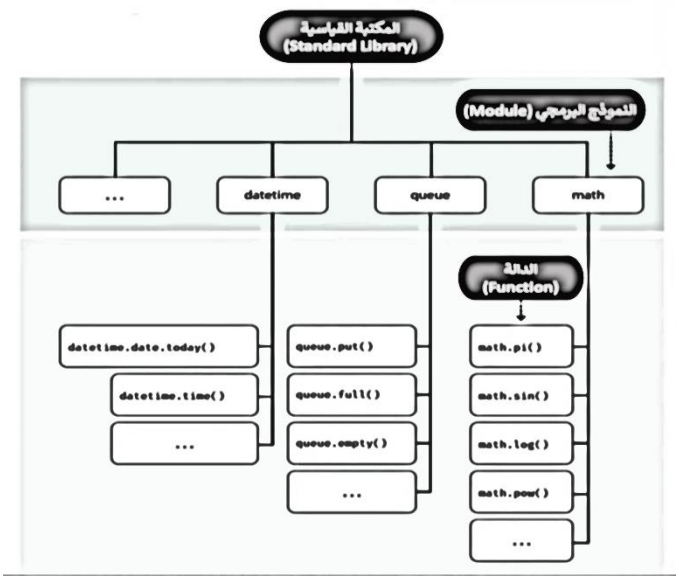
- **مكتبة بايثون القياسية Python Standard Library**

هي مكتبة تقدم مجموعة واسعة من النماذج البرمجية فهي تحتوي على نماذج برمجية مدمجة مكتوبة بلغة البرمجة C توفر الوصول إلى الوظائف للنظام مثل الملفات وكذلك على النماذج البرمجية التي تم كتابتها بلغة بايثون وتلك توفر حلولا للعديد من المشكلات البرمجية وتوجد الدوال داخل النماذج البرمجية داخل المكتبات القياسية ، وهي تثبت تلقائيا عند تثبيت البايثون مما يجعلها متاحة بشكل موثوق وبهذا تكون هذه النماذج جزءا أساسيا من لغة بايثون .



- **استخدام مكتبة بايثون القياسية**

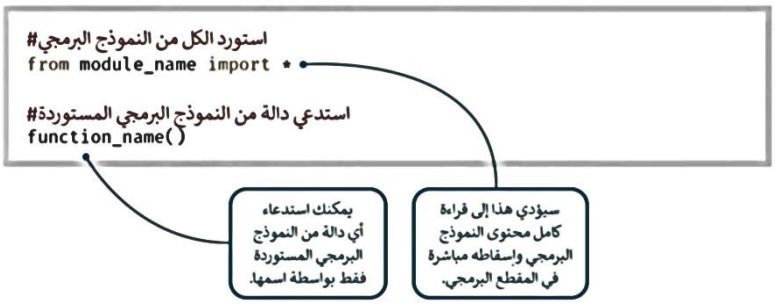
نظرا لأن مكتبة بايثون القياسية مثبتة بالفعل فقط يتم استيراد نماذجها البرمجية إلى البرنامج عن طريق إضافة سطر الأوامر في أعلى السطر البرمجي في شاشة بايثون.



- **طرق استيراد نماذج المكتبة القياسية**

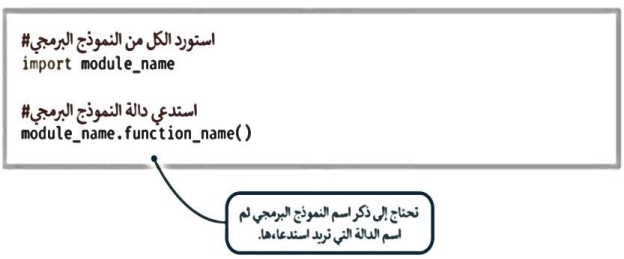
١. الطريقة الأولى: استيراد الكل

يمكن تضمين محتويات المكتبة البرمجية باستخدام هذا السطر.



٢. الطريقة الثانية: استيراد النماذج البرمجية

استيراد كل محتوياتها وجعلها متاحة فقط من خلال كتابة اسم النموذج البرمجي ثم اسم الدالة.



- **النماذج الأكثر استخداما**

١. **نموذج sys البرمجي**

الهدف منه هو مساعدة المطور في معرفة المزيد عن النظام الخاص بجهاز المستخدم ومشغل بايثون الذي ثبت على الجهاز ويتم استيراد نموج sys البرمجي باستخدام الأمر " import "

- لعرض نظام التشغيل

Import sys

```
#اعرض نظام تشغيل جهاز الحاسب
print(sys.platform)
```

Win 64

٢. **نموذج os البرمجي**

يوفر بعض الوظائف للمستخدم التفاعل مع جهازه دون الحاجة لنظام التشغيل مثل معرفة مسار العمل الحالي أو إنشاء مجلد وحذفه.

❖ قواعد بناء جملة بايثون وإضافة التعليقات

في هذا الدرس ستتعلم كيف تكتب كود بايثون بشكل صحيح يفهمه الكمبيوتر ويفهمه أي شخص يحاول قراءة الكود الذي ستقوم أنت بكتابته مستقبلاً عند بناء تطبيقاتك الخاصة.

- **Case Sensitivity**

Case Sensitivity تعني أن لغة البرمجة تميز بين الأحرف الكبيرة و الأحرف الصغيرة.

بايثون تعامل الأسماء التي نستخدمها بتأني سواء كنا نضع هذه الأسماء للمتغيرات، الدوال، الكلاسات، الكائنات...الخ.

- **مثال:** note و Note ليسوا شيئاً واحداً.

- **اسم الكلاس**

دائماً ابدأ اسم الكلاس بحرف كبير وفي حال كان اسم الكلاس يتألف من أكثر من كلمة، اجعل أول حرف من كل كلمة كبيراً.

- **أمثلة:**

في حال كان اسم الكلاس يتألف من كلمة واحدة.

```
class First:
```

في حال كان اسم الكلاس يتألف من أكثر من كلمة.

```
class FirstPythonClass:
```

- **اسم المتغير**

استخدم الأحرف الصغيرة عند وضع أسماء للمتغيرات وفي حال كان اسم المتغير يتألف من أكثر من كلمة قم بوضع _ بين كل كلمتين.

- **أمثلة:**

في حال كان اسم المتغير يتألف من كلمة واحدة.

average = 10

في حال كان اسم المتغير يتألف من أكثر من كلمة.

total_score = 20

- **اسم الدالة**

استخدم الأحرف الصغيرة عند وضع أسماء للدوال وفي حال كان اسم الدالة يتألف من أكثر من كلمة قم بوضع _ بين كل كلمتين.

- **أمثلة:**

في حال كان اسم الدالة يتألف من كلمة واحدة.

```
def display():
```

في حال كان اسم الدالة يتألف من أكثر من كلمة.

```
def display_user_info():
```

- **إضافة التعليقات**

نستخدم التعليقات لنضع ملاحظات حول الكود الذي كتبناه فقط لكي لا ننسى كيف برمجنا الكود في حال أردنا مراجعته أو التعديل عليه بعد وقت طويل.

التعليقات لا تؤثر إطلاقاً على الكود المكتوب الموضوع في البرنامج و يمكن وضع عدد غير محدود من التعليقات.

لتضع تعليق ضع الرمز # ثم أكتب بعده ما شئت.

تذكر: أنت لست مجبراً على وضع تعليقات في برامجك، ولكننا ننصحك بوضع تعليقات دائماً حتى تساعدك في فهم الكود الذي كتبته.

- **مثال:**

```
#هذا تعليق يتألف من سطر واحد و هو لا يؤثر أبداً على الكود الموضوع.

#هذا تعليق آخر.. كما تلاحظ يمكنك وضع العدد الذي تريده من التعليقات.
```

❖ التعامل مع بايثون وكتابة الأكواد

في البداية إذا كنت تستخدم برنامج PyCharm فإنه سيقوم بتنبيهك إذا خالفت أي قاعدة من قواعد كتابة الكود الإجبارية في بايثون.

- **قواعد كتابة الكود في بايثون هي التالية:**

- لا تقم بإضافة أي مسافة فارغة باستخدام الزر TAB لأن المسافة التي يعطيها هذا الزر غير مسموح استخدامها في لغة بايثون.
- استخدم 4 مسافات فارغة Space عند وضع الكود بشكل متداخل.
- ضع سطر فارغ على الأقل بين السطر الذي تم فيه تعريف الكلاس والدوال المعرفة بداخله.
- ضع سطر فارغ على الأقل بين كل دالتين.
- ضع سطر فارغ بين كل إثنين بلوك تضيفهما بداخل الدوال.
- ضع مسافة فارغة حول جمل التحكم وجمل الشرط.
- عند وضع التعليقات يفضل استخدام الرمز # في بداية كل سطر حتى وإن كان التعليق يتألف من عدة أسطر.
- عدد الأحرف القصوى التي يمكن وضعها في كل سطر هو 79 حرف.
- استخدم محرر الأكواد لكتابة الأكواد البرمجية ثم احفظ الملف بامتداد .py. يمكنك تنفيذ الكود من خلال IDLE أو من سطر الأوامر باستخدام:

```
python script.py
```

-

- **أمثلة حول طريقة كتابة الكود في بايثون**

الهدف من المثال التالي إظهار كيف يجب ترتيب كود البايثون وليس معرفة طريقة عمله.

- **المثال الأول**

```
Test.py

# هنا قمنا بتعريف متغير اسمه note قيمته ١٤

note = 14

# هنا وضعنا شرط يعني أنه إذا كانت قيمة المتغير note أكبر أو تساوي ١٠ سيتم تنفيذ أمر الطباعة الموضوع بداخلها

if note >= 10:

    print("Congratulations.. you passed the test!")
```

#أكبر أو تساوي ١٠ سيتم تنفيذ أمر الطباعة الموضوع بداخلها note هنا لنا أنه إذا لم تكن قيمة المتغير

```
    else:

        print("Sorry.. you failed the test!")
```

سنحصل على النتيجة التالية عند التشغيل.

```
Congratulations.. you passed the test!
```

في حال كنت تنوي وضع كل الكود السابق بداخل كلاس، يجب أن تضيف ٤ مسافات فارغة قبل كل سطر.

تذكر: الهدف من المثال التالي إظهار كيف يجب ترتيب كود البايثون وليس معرفة طريقة عمله.

- المثال الثاني

```
Test.py
# Test هنا قمنا بتعريف كلاس إسمه
class Test:
  # ١٤ قيمته note هنا قمنا بتعريف متغير إسمه
  note = 14
```

أكبر أو تساوي ١٠ سيتم تنفيذ أمر الطباعة الموضوع بداخلها **note** هنا وضعنا شرط يعني أنه إذا كانت قيمة المتغير

```
    if note >= 10:
        print("Congratulations.. you passed the test!")
```

أكبر أو تساوي ١٠ سيتم تنفيذ أمر الطباعة الموضوع بداخلها **note** هنا كأننا قلنا أنه إذا لم تكن قيمة المتغير

```
    else:
        print("Sorry.. you failed the test!")
```

سنحصل على النتيجة التالية عند التشغيل.

```
Congratulations.. you passed the test!
```

❖ تعريف المتغيرات وأنواع البيانات في بايثون

المتغيرات (variables) عبارة عن أماكن يتم حجزها في الذاكرة بهدف تخزين بيانات فيها أثناء تشغيل البرنامج.

في بايثون المبرمج غير مسؤول عن تحديد أنواع المتغيرات التي يعرّفها في برنامجه، فعلياً، عندما تقوم بتعريف متغير وتضع فيه أي قيمة، سيقوم مفسّر لغة بايثون بتحديد نوع هذا المتغير بناءا على القيمة التي أسندتها إليه بشكل تلقائي وقت التشغيل.

في بايثون يجب إسناد قيمة إلى المتغير أثناء تعريفه.

- **المثال الأول:**

```
Test.py

var = 5                          # هنا قمنا بتعريف متغير اسمه var و قيمته 5

print(var)                       # هنا قمنا بطباعة قيمة المتغير var
```

سنحصل على النتيجة التالية عند التشغيل.

```
5
```

في بايثون يمكن تعريف عدة متغيرات متساوية في القيمة في وقت واحد.

- **المثال الثاني:**

```
Test.py

x = y = z = 10      # هنا قمنا بتعريف ثلاث متغيرات قيمتها 10

print('x = ', x)      # هنا قمنا بطباعة قيمة المتغير x

print('y = ', y)      # هنا قمنا بطباعة قيمة المتغير y

print('z = ', z) # هنا قمنا بطباعة قيمة المتغير z
```

سنحصل على النتيجة التالية عند التشغيل.

```
x = 10
y = 10
z = 10
```

- **معرفة نوع المتغير**

لمعرفة نوع أي متغير يمكنك استخدام الدالة type ()

تذكر: نوع المتغير في بايثون غير ثابت لأنه يتغير بشكل تلقائي على حسب نوع القيمة التي يتم تخزينها فيه.

- **مثال:**

```
Test.py
var = 10            # var هنا وضعنا رقم في المتغير
print(type(var))          # هنا طبعنا نوع var. لاحظ أن نوعها سيكون int لأنها عبارة عن رقم
قيمة المتغير
var = 'harmash'       # var هنا وضعنا نص في المتغير
print(type(var))        # هنا طبعنا نوع قيمة var. لاحظ أن نوعها سيكون str لأنها عبارة عن نص
المتغير
```

سنحصل على النتيجة التالية عند التشغيل.

```
<class 'int'>
<class 'str'>
```

- **أنواع البيانات في البايثون**

تنقسم أنواع البيانات أو المتغيرات في بايثون إلى 7 أنواع أساسية وهي:

١. أرقام (Numbers).
٢. نصوص (Strings).
٣. منطقية (Booleans).
٤. مصفوفات ليس لها حجم ثابت يقال لها Lists.
٥. مصفوفات حجمها وقيمها ثابتة، وغير قابلة للتغيير يقال لها Tuples.
٦. مصفوفات ليس لها حجم ثابت، ولا يمكن حذف قيمها، ويمكن إضافة قيم جديدة فيها يقال لها Sets.
٧. جداول تخزن البيانات فيها بطريقة مفاتيح (Keys) وقيم (Values) يقال لها Dictionaries.

- **الأرقام**

عند تعريف متغير وتخزين رقم فيه، فإن مفسّر لغة بايثون سيقوم بشكل تلقائي بتحديد نوع هذا المتغير بناءاً على نوع القيمة الرقمية التي تم إسنادها إليه، فإذا وضعت فيه عدد صحيح، يصبح نوعه int وإذا وضعت فيه عدد عشري (أي يحتوي على فاصلة) يصبح نوعه float وهكذا.

أنواع الأرقام في بايثون تنقسم إلى 3 أنواع كما في الجدول التالي:

النوع	استخدامه	مثال
Int	يستخدم لتخزين أعداد صحيحة.	x = 3
Float	يستخدم لتخزين أعداد تحتوي على فاصلة عشرية.	x = 1.5
complex	يستخدم لتخزين أعداد مركبة (Complex Number) والتي غالباً ما يحتاجها المهندسون عند إجراء عمليات حسابية معقدة. **ملاحظة:** هنا يجب وضع الحرف J or j مباشرةً بعد العدد حتى يعرف مفسّر بايثون أنك تقصد عدد مركب وليس عدد عادي.	x = 4J

- مثال

في المثال التالي قمنا بتعريف ثلاث متغيرات وكل متغير وضعنا فيه قيمة رقمية مختلفة في النوع والقيمة، بعدها قمنا بعرض نوع كل متغير منهم.

```
Test.py
x = 3           # هنا قمنا بتعريف متغير إسمه x , قيمته عبارة عن عدد صحيح
y = 1.5         # هنا قمنا بتعريف متغير إسمه y , قيمته عبارة عن عدد عشري
z = 4J          # هنا قمنا بتعريف متغير إسمه z , قيمته عبارة عن عدد مركب
print(type(x))      # هنا طبعنا نوع قيمة المتغير x
print(type(y))      # هنا طبعنا نوع قيمة المتغير y
print(type(z))          # هنا طبعنا نوع قيمة المتغير z
```

سنحصل على النتيجة التالية عند التشغيل

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

- ملاحظة

صحيح أن مفسّر بايثون يقوم بتحديد أنواع القيم بشكل تلقائي عنك لكن هذا لا يعني أنك غير قادر على تحويل أنواع الأرقام إلى النوع الذي يناسبك.

ستتعرف على دوال خاصة للتعامل مع الأرقام وستتعلم طريقة تحويل أنواع الأرقام في درس خاص لاحقاً في هذه الدورة.

- **النصوص**

لتعريف نص في بايثون نستخدم الرمز ' أو الرمز " أو الرمز """

هل يوجد فرق بين هذه الرموز؟

بالنسبة للرمز ' والرمز " فإنه لا يوجد أي فرق بينهما ويمكن استخدام أي واحد منهما لتعريف نص يتألف من سطر واحد.

بالنسبة للرمز ''' والرمز """ فإنه لا يوجد أي فرق بينهما، ويمكن استخدام أي واحد منهما لتعريف نص كبير يتألف من عدة أسطر.

- **القيم المنطقية**

النوع bool يستخدم في العادة عند وضع شروط منطقية أو لمعرفة ما إذا تم تنفيذ أمر معين بنجاح أم لا،

عند إسناد القيمة True أو القيمة False إلى المتغير فإنه يصبح من النوع bool .

- **معلومة تقنية**

في الواقع القيمة True تساوي ١ والقيمة False تساوي ٠.

في بايثون يفضل استخدام الصفر والواحد بدلاً من استخدام القيم المحجوزة True و False عند التشييك على قيمة المتغير أو على ما سترجعه الدالة.

سيتم استعراض المصفوفات Tubles , Lists في الوحدة الثالثة.

❖ التعامل مع الوقت والتاريخ

بايثون تحتوي على أكثر من موديول جاهز للتعامل مع التاريخ، الوقت والتقويم بكل سهولة.

في هذا الدرس ستتعلم طريقة الاستفادة من الموديول datetime والموديول calendar لعرض التاريخ، الوقت والتقويم بالشكل الذي تحتاجه.

- **الموديول datetime**

الموديول datetime يحتوي على مجموعة كلاسات فيها دوال جاهزة للتعامل مع التاريخ والوقت.

- **Date** : يحتوي على مجموعة دوال خاصة للتعامل مع التاريخ.
- **Time**:يحتوي على مجموعة دوال خاصة للتعامل مع الوقت.
- **Datetime**:يحتوي على مجموعة دوال خاصة للتعامل مع التاريخ و الوقت.
- **Timedelta**: يحتوي على مجموعة دوال خاصة لحساب الفرق بدقة بين تاريخ و آخر.
- **Timezone**: يحتوي على مجموعة دوال خاصة لحساب فرق التوقيت بين تاريخ و آخر على حسب المنطقة الزمنية لكل تاريخ.

في **المثال** التالي قمنا بتخزين التاريخ والوقت الحالي بالاعتماد على الدالة now() الموجودة في الكلاس datetime الموجود بداخل الموديول datetime

```
# datetime هنا قمنا بتضمين كل محتوى الموديول
import datetime


# dt في الكائن datetime ككائن من الكلاس now() هنا قمنا بتخزين التاريخ و الوقت الحالي
الذي سترجعه الدالة
dt = datetime.datetime.now()
# dt هنا قمنا بعرض قيمة الكائن
print(dt)
```

سنحصل على نتيجة تشبه النتيجة التالية عند التشغيل

```
2018-12-01 09:13:09.598797
```

نلاحظ من المثال السابق أن الكلاس datetime يتيح لنا الحصول على التاريخ والوقت بدقة عالية لأنه أعطانا المعلومات التالية بالترتيب: السنة، الشهر، اليوم، الساعة، الدقيقة، الثانية، أجزاء الثانية القادمة.

طبعاً الكلاس datetime يحتوي على دوال وخصائص جاهزة تتيح لك التعامل مع التاريخ والوقت كيفما شئت لتخزينه أو عرضه بالشكل الذي تريد.

عند إنشاء كائن ,datetime يمكنك مباشرةً أن تدخل تاريخ ووقت فيه.

فعلياً، أنت مجبر على إدخال قيمة مكان البارميترات year و month و day لأنه لم يتم إعطائهم قيم افتراضية.

بالنسبة للباراميترات الأخرى فيمكنك تحديد قيمهم الافتراضية أو عدم تحديدها لأنه تم إعطائهم قيم افتراضية.

القيم التي يمكنك تمريرها للبارميترات هي التالية:

```
year: عدد صحيح قيمته ضمن النطاق 1 <= year <= 9999.

month: عدد صحيح قيمته ضمن النطاق 1 <= year <= 12

day: عدد صحيح قيمته بين ١ و آخر يوم موجود في الشهر.

hour: عدد صحيح قيمته ضمن النطاق ٠ 0 <= year < 24

minute: عدد صحيح قيمته ضمن النطاق 0 <= year < 60.

second: عدد صحيح قيمته ضمن النطاق 0 <= year < 60.

microsecond: عدد صحيح قيمته ضمن النطاق 0 <= year < 1000000.

fold: عدد صحيح قيمته ٠ أو ١.
```

في حال قمت بتمرير أي قيمة للباراميترات خارجة عن النطاق المسموح سيحدث خطأ ValueError.

في المثال التالي: قمنا بإنشاء كائن من الكلاس datetime مع تحديد التاريخ الذي نريد تخزينه فيه مباشرةً عند إنشاءه.

```
Test.py

# datetime هنا قمنا بتضمين كل محتوى الموديول

import datetime

# dt هنا قمنا بإنشاء كائن من الكلاس datetime يمثل تاريخ محدد و قمنا بتخزينه في الكائن

dt = datetime.datetime(2012, 4, 5)

# dt هنا قمنا بعرض قيمة الكائن

print(dt)
```

سنحصل على نتيجة تشبه النتيجة التالية عند التشغيل.

```
2012-04-05 00:00:00
```

❖ التعامل مع المعاملات

المعاملات (Operators) في لغة البرمجة بايثون هي عبارة عن رموز خاصة لها وظيفة معينة، وفي لغة البرمجة بايثون، **هناك أنواع مختلفة من هذه المعاملات:**

- المعاملات التي تُستخدم في إجراء العمليات الحسابية (Arithmetic Operators).
- المعاملات التي تستخدم في إجراء عمليات المقارنة (Comparison Operators).
- المعاملات التي تستخدم في الشروط منطقية (Logical Operators) .
- المعاملات التي تستخدم لإسناد قيم للمتغيرات (Assignment Operators).
- المعاملات التي تستخدم للبحث في المصفوفات (Membership Operators).

- **المعاملات المستخدمة في العمليات الحسابية Arithmetic Operators**

الاستخدام	الرمز	العامل
يُستخدم لإعطاء قيمة معينة للمتغيرات.	=	المساواة (Assignment)
تستخدم لجمع القيم.	+	الجمع (Addition)
تستخدم لطرح القيم المختلفة.	-	الطرح (Subtraction)
تستخدم لقسمة قيمة معينة على قيمة أخرى.	/	القسمة (Division)
يستخدم لضرب قيمة معينة بقيمة أخرى.	*	الضرب (Multiplication)
يستخدم لمضاعفة قيمة معينة بعدد معين مثال: y ** x وكانت قيمة x تساوي ٢ وقيمة (y) تساوي ٥ فستقوم بمضاعفة ال (x) خمس مرات وتكون النتيجة (32)	**	Exponentiation
تستخدم لحساب باقي القسمة.	%	باقي القسمة (Modulo)

- **المعاملات التي تستخدم في إجراء عمليات المقارنة Comparison Operators**

المعامل	الرمز	الاستخدام
Equal to	==	يستخدم لإجراء عملية مقارنة وتعني هل القيمة الأولى مساوية للثانية (a == b) في حال كانت القيم متساوية يتم إرجاع قيمة (True) وإذا لم تكن متساوية يتم إرجاع قيمة (False)
Not equal to	=!	يستخدم لإجراء عملية مقارنة قيما إذا كانت القيمة الأولى لا تساوي القيمة الثانية وإذا كان الجواب نعم فإنها ترجع قيمة (True)
أكبر من(Greater than)	<	تستخدم لإجراء مقارنة بين قيمتين أيهما أكبر وإذا كان الجواب نعم فإنها ترجع قيمة (True)
أصغر من(Less than)	>	تستخدم لإجراء مقارنة بين قيمتين أيهما أصغر وإذا كان الجواب نعم فإنها ترجع قيمة (True)
أكبر من أو يساوي (Greater than or Equal to)	=<	تستخدم لإجراء مقارنة بين قيمتين هل قيمة الأولى أكبر من أو تساوي القيمة الثانية وإذا كان الجواب نعم فإنها ترجع (True)
أصغر من أو يساوي (Less than or Equal to)	=>	تستخدم لإجراء مقارنة بين قيمتين هل قيمة الأولى أصغر من أو تساوي القيمة الثانية وإذا كان الجواب نعم فإنها ترجع (True)

- **المعاملات التي تستخدم في شروط منطقية Logical Operators**

المعامل	الرمز	الاستخدام
Logical AND	and	تستخدم للمقارنة بين قيمتين هل قيمة y و x تساويان True ؟ وفي هذه الحالة يتم تحقيق الشرطين معاً ليرجع قيمة (True)
Logical OR	or	في هذه الحالة يتم المقارنة هل قيمة (x) أو (y) أو كلاهما تساويان True؟ هنا يكفي أن يتم تحقيق شرط واحد من الشرطين ليرجع قيمة (True)
Logical NOT	not	هنا يتم المقارنة هل قيمة (x) لا تساوي True؟ إذا كان الجواب نعم فإنها ترجع (True)

- **المعاملات التي تستخدم للبحث في المصفوفات Membership Operators**

المعامل	الرمز	الاستخدام
In	In	هل قيمة المتغير (x) موجودة في المصفوفة (array) وإذا كان الجواب نعم فإنها ترجع (True) وتستخدم كما يلي: (x in arr)
Not In	not in	هل قيمة المتغير (x) غير موجودة في المصفوفة (array) وإذا كان الجواب نعم فإنها ترجع (True) وتستخدم كما يلي: (x not in arr)

- **المعاملات التي تستخدم لإسناد قيم للمتغيرات Assignment Operators**

المعامل	الرمز	الاستخدام
Basic Assignment	=	يستخدم لإسناد قيمة معينة في قيمة أخرى على سبيل المثال: x = y وتعني ضع قيمة (y) في (x)
Add AND Assignment	=+	تستخدم لإضافة قيمة إلى قيمة أخرى فعلى سبيل المثال: x+= y هنا يتم إضافة قيمة (x) على قيمة (y) ويتم تخزين الناتج في (x)
Subtract AND Assignment	=-	تستخدم لطرح قيمة معينة من قيمة أخرى وتخزينها فعلى سبيل المثال: x -= y في هذه الحالة يتم إنقاص قيمة (x) من قيمة (y) ومن ثم يتم تخزين الناتج في (x)
Multiply AND Assignment	=*	تستخدم لضرب قيمة بقيمة أخرى مثال: x *= y يتم هنا ضرب قيمة (x) بقيمة (y) وتخزين الناتج في (x)
Exponent AND Assignment	=**	تستخدم لمضاعفة قيمة معينة وتخزينها مرة أخرى فعلى سبيل المثال: x **= y في هذه الحالة يتم مضاعفة قيمة (x) بقيمة (y) ومن ثم تخزين الناتج في (x)
Divide AND Assignment	=/	تستخدم في قسمة قيمة على قيمة أخرى فعلى سبيل المثال: x /= y في هذه الحالة يتم قسمة قيمة (x) على قيمة (y) ومن ثم تخزين الناتج في (x)

❖ اتخاذ القرارات وتنفيذ الأعمال المتكررة

الشروط (conditions) تستخدم لتحديد طريقة عمل البرنامج نسبةً للمتغيرات التي تطرأ على الكود، كمثال بسيط يمكنك بناء برنامج لمشاهدة الأفلام، عند الدخول إليه يطلب من المستخدم في البداية أن يدخل عمره لكي يقوم بعرض أفلام تناسب عمره.

يمكنك وضع العدد الذي تريده من الشروط في البرنامج الواحد وتستطيع وضع الشروط بداخل بعضها البعض أيضاً.

- **جمل الشرط**

طريقة وضع الشروط (Syntax) :

```
if condition:
    # إذا كان الشرط صحيحاً نفذ هذا الكود
elif condition:
    # إذا كان الشرط صحيحاً نفذ هذا الكود
else:
    # إذا لم يتحقق أي شرط نفذ هذا الكود
```

لست بحاجة إلى استخدام الجمل الثلاثة في كل شرط تضعه في البرنامج، ولكنك مجبر على استخدام جملة الشرط if مع أي شرط.

- **أمثلة على جمل الشرط**

اسم الجملة	دواعي الاستخدام
IF Statement	If في اللغة العربية تعني "إذا" و هي تستخدم فقط في حال كنت تريد تنفيذ كود معين حسب شرط معين.
Else Statement	else في اللغة العربية تعني "أي شيء آخر" وهي تستخدم فقط في حال كنا نريد تنفيذ كود معين في حال كانت نتيجة جميع الشروط التي قبلها تساوي false يجب وضعها دائماً في الأخير، لأنها تستخدم في حال لم يتم تنفيذ أي جملة شرطية قبلها.
Else IF Statement	elif تستخدم إذا كنت تريد وضع أكثر من إحتمال (أي أكثر من شرط) جملة أو جمل الـ elif يوضعون في الوسط، أي بين الجملتين if و else.

- **أفكار وأساليب أخرى لوضع الشروط في بايثون**

طرق وضع الشروط عديدة ومتنوعة ويمكننا وضع شروط بداخل بعضها ويسمى ذلك Nested Conditional.

كما يمكننا وضع أكثر من شرط بداخل جمل الشرط if أو else if باستخدام العوامل المنطقية.

هنا وضعنا أمثلة تعلمك طريقة وضع أكثر من شرط في الجملة if أو elif باستخدام الـ Relational Operators.

- المثال الأول

إذا كانت قيمة المتغير a بين 0و 20 اطبع الجملة acceptable number :

```
Test.py

a = 14;

if a >= 0 and a <= 20:

print("acceptable number")
```

سنحصل على النتيجة التالية عند التشغيل

```
acceptable number
```

لاحظ أنه قد تم تنفيذ أمر الطباعة لأن قيمة المتغير a بين ٠ و ٢٠.

هنا سأل نفسه سؤالين..

السؤال الأول: هل قيمة المتغير a أكبر أو تساوي ٠؟

فكان جواب الشرط الأول true

السؤال الثاني: هل قيمة المتغير a أصغر أو تساوي ٢٠؟

فكان جواب الشرط الثاني أيضاً true

بما أن كلا الجوابين كانا true قام بتنفيذ أمر الطباعة.

- المثال الثاني

إذا كانت قيمة المتغير a تساوي 1أو 2 أو 3 اطبع الجملة you choose a valid number

```
Test.py

a = 2

if a == 1 or a == 2 or a == 3:

print('you choose a valid number')
```

سنحصل على النتيجة التالية عند التشغيل.

```
you choose a valid number
```

لاحظ أنه قام بتنفيذ أمر الطباعة لأن قيمة المتغير a تساوي ٢

هنا : كان سيسأل نفسه ثلاثة أسئلة لأنه يوجد ثلاث شروط لكنه سأل نفسه سؤالين فقط..

السؤال الأول: هل قيمة المتغير a تساوي ١؟

فكان جواب الشرط الأول false فانتقل للشرط الذي يليه.

السؤال الثاني: هل قيمة المتغير a تساوي ٢؟

فكان جواب الشرط الثاني true

بما أن واحداً من الأجوبة كان true نفذ مباشرةً أمر الطباعة ولم ينظر حتى للشرط الأخير.

❖ التعامل مع السلاسل النصية

في عالم البرمجة، نقول للنص "String" سواء كان يتألف من حرف واحد، كلمة، جملة أو نص كبير جداً ومن هذا المنطلق نستنتج أن النص عبارة عن سلسلة من الأحرف ليس لها حجم محدد.

في بايثون تم إنشاء الكلاس أو النوع str خصيصاً لتخزين القيم النصية.

النوع str يعتمد على الترميز Unicode وهذا يعني أنك لن تواجه أي مشكلة عند التعامل مع نصوص عربية، إنجليزية، فرنسية ...إلخ

المتغيرات النصية في بايثون (التي نوعها str) تعتبر Immutable وهذا يعني أنك عندما تقوم بتعريف أي متغير تخزن فيه نص, فإن هذه النص سيحجز له مكان في الذاكرة مهما كان حجمه، و إذا قمت بإعطاء قيمة جديدة لهذا المتغير فإنه سيتم حذف القيمة القديمة من الذاكرة و إنشاء مكان جديد في الذاكرة ووضع القيمة الجديدة فيه لأنه لا يمكن تعديل نفس القيمة في نفس المكان في الذاكرة.

- **طريقة تعريف str**

لتعريف نص في بايثون نستخدم الرمز ' أو الرمز " أو الرمز """.

هل يوجد فرق بين هذه الرموز؟

بالنسبة للرمز ' والرمز " فإنه لا يوجد أي فرق بينهما ويمكن استخدام أي واحد منهما لتعريف نص يتألف من سطر واحد.

بالنسبة للرمز ''' والرمز """ فإنه لا يوجد أي فرق بينهما ويمكن استخدام أي واحد منهما لتعريف نص كبير يتألف من عدة أسطر.

- المثال الأول:

```
Test.py
# هنا قمنا بتعريف ثلاث متغيرات تحتوي على قيم نصية
name = 'Mohamed'
job = "Programmer"
message = '''This string that will span across multiple lines. No need to use newline characters for the next lines.
The end of lines within this string is counted as a newline when printed.'''
# هنا قمنا بعرض قيم المتغيرات النصية بأسلوب مرتب
print('Name: ', name)
print('Job: ', job)
print('Message: ', message)
```

سنحصل على النتيجة التالية عند التشغيل.

```
Name: Mohamed
Job: Programmer
Message: This string that will span across multiple lines. No need to use newline characters for the next lines.
The end of lines within this string is counted as a newline when printed.
```

- المثال الثاني:

في المثال التالي قمنا بتعريف نص يحتوي على نفس الرموز التي تستخدم لتعريف النصوص.

```
Test.py
# هنا قمنا بتعريف متغير إسمه text يحتوي على قيمة نصية
text = """In this line we print 'single quotations'
In this line we print "double quotations" """
# هنا قمنا بعرض قيمة المتغير text
print(text)
```

سنحصل على النتيجة التالية عند التشغيل

```
In this line we print 'single quotations'
In this line we print "double quotations"
```

- **مفهوم الـConcatenation الدمج**

Concatenation تعني وضع سلسلة من النصوص بجانب بعضها لعرضها كنص واحد ، وهذا الأمر ستحتاجه في أي تطبيق.

فمثلاً، في البرامج أو المواقع التي تستخدمها تلاحظ أنه عند إنشاء حساب جديد يطلب منك أن تدخل اسمك على مرحلتين **كالتالي:**

- الاسم (First Name)
- اسم العائلة (Last Name)

بعد أن تنشئ حسابك تلاحظ أنه قام بعرض اسمك الكامل (الاسم + اسم العائلة).

عند وضع الاسمين بجانب بعضهما وكأنهما نص واحد يكون المبرمج فعلياً قد قام بدمجهما فقط وليس إعادة كتابتهما من جديد.

في المثال التالي سنقوم بتعريف المتغير first_name لنضع فيه الاسم، والمتغير last_name لنضع فيه اسم العائلة، والمتغير full_name لنضع فيه الاسم واسم العائلة.

- **مثال:**

```
Test.py
# هنا قمنا بإنشاء المتغير first_name و وضعنا فيه نص يمثل الاسم
first_name = 'Mohamed'
# هنا قمنا بإنشاء المتغير last_name و وضعنا فيه نص يمثل إسم العائلة
last_name = 'Harmush'
هنا قمنا بإنشاء المتغير full_name ووضعنا فيه الاسم الموجود في المتغير first_name واسم
العائلة الموجود في المتغير last_name وأضفنا بينهما مسافة فارغه #
full_name = first_name + ' ' + last_name
هنا عرضنا قيمة المتغير full_name وبالتالي سيتم عرض الاسم الكامل الذي قمنا بدمجه ووضعه
فيه #
print(full_name)
```

نحصل على النتيجة التالية عند التشغيل

```
Mohamed Harmush
```

- **طرق دمج النصوص**

يمكنك تطبيق الـ Concatenation في بايثون بطريقتين:

باستخدام العامل + أو باستخدام الدالة () join التي تعطيك طرق متقدمة أكثر لدمج النصوص، ستجد شرح مفصل لهذه الدالة لاحقاً في هذا الدرس.

- **الدمج التلقائي للنصوص**

في حال وضعت قيمتين نصيّتين بجانب بعضهما و وضعت بينهما مسافة فارغة (أو عدة مسافات فارغة) فإن مترجم بايثون سيقوم بدمجهم لك بشكل تلقائي.

- **مثال:**

```
Test.py
s = 'Mohamed' ' Harmush'
                # s هنا قمنا بوضع قيمتين نصيّتين بجانب بعضهما و قمنا بتخزينهما في المتغير
print(s)
                # s3 هنا عرضنا قيمة المتغير لنتأكد من أنه قد تم دمج النصين بشكل صحيح
```

سنحصل على النتيجة التالية عند التشغيل

```
Mhamad Harmush
```

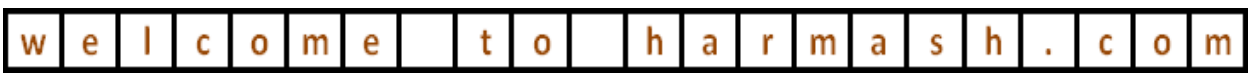
- **الوصول لأحرف النص**

لنفترض أننا قمنا بتعريف متغير اسمه s وقيمته النص 'welcome to harmash.com'

- **مثال:**

```
s = 'welcome to harmash.com'
```

سيتم تخزين نص المتغير s في الذاكرة حرفاً حرفاً وبالترتيب كما في هذه الصورة التالية..



- **طرق الوصول لأحرف النص**

في حال أردت الوصول لأحرف هذا النص، فأمامك خيارين:

الوصول لأحرف النص من جهة اليسار إلى اليمين، وهذا يحدث بشكل تلقائي عندما تستخدم أرقام أكبر أو تساوي 0 للوصول لهذه الخانات التي تحتوي الأحرف.

الوصول لأحرف النص من جهة اليمين إلى اليسار وهذا يحدث بشكل تلقائي عندما تستخدم أرقام أصغر من 0 للوصول لهذه الخانات التي تحتوي الأحرف.

- **الوصول لأحرف النص من اليسار إلى اليمين**

في حال أردت المرور على أحرف هذا النص من اليسار إلى اليمين، سيتم اعتبار أن الخانات قد تم ترقيمها ابتداءً من الرقم 0 كالتالي:

معلومة: طريقة ترقيم الخانات تسمى (Forward Indexing).

w	e	l	c	o	m	e		t	o		h	a	r	m	a	s	h	.	c	o	m
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

في المثال التالي سنقوم بطباعة محتوى أول 7 خانات ابتداءً من الجهة اليسرى وبالتالي سنحصل على الكلمة welcome

- **مثال:**

```
Test.py
# هنا قمنا بتعريف متغير إسمه s يحتوي على نص
s = 'welcome to harmash.com'
# هنا قمنا بطباعة أول ٧ أحرف موجودة في المتغير s وبدأنا من اليسار
print(s[0] + s[1] + s[2] + s[3] + s[4] + s[5] + s[6])
```

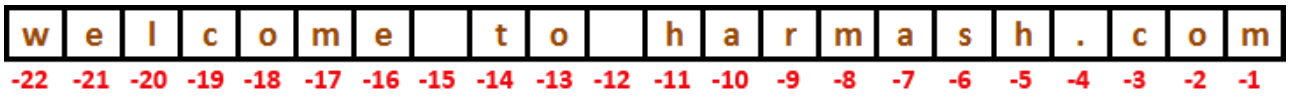
سنحصل على النتيجة التالية عند التشغيل..

```
welcome
```

- **الوصول لأحرف النص من اليمين إلى اليسار**

في حال أردت المرور على أحرف هذا النص من اليمين إلى اليسار، سيتم اعتبار أن الخانات قد تم ترقيمها ابتداءً من الرقم ١- كالتالي:

معلومة: طريقة ترقيم الخانات تسمى (Backward Indexing).



- **مثال**

في المثال التالي سنقوم بطباعة محتوى آخر ٣ خانات ابتداءً من الجهة اليمنى وبالتالي سنحصل على الكلمة com

```
Test.py
# هنا قمنا بتعريف متغير إسمه s يحتوي على نص
s = 'welcome to harmash.com'
# هنا قمنا بطباعة آخر ٣ أحرف موجودة في المتغير s وبدأنا من اليمين
print(s[-3] + s[-2] + s[-1])
```

سنحصل على النتيجة التالية عند التشغيل..

```
com
```

- **معرفة عدد أحرف النص بواسطة الدالة len(s)**

لمعرفة عدد أحرف أي نص نقوم باستدعاء الدالة len() ومن ثم نمرر لها النص مكان الباراميتر s.

ملاحظة: المسافات الفارغة (White Spaces) أيضاً يتم حسابها.

- **مثال:**

```
Test.py
s = 'welcome to harmash.com'      # هنا قمنا بتعريف متغير نصي إسمه s
print('Length of s=',len(s))      # هنا قمنا بطباعة عدد أحرف (خانات) المتغير s الذي سترجعه الدالة len()
```

سنحصل على النتيجة التالية عند التشغيل..

```
Length of s = 22
```

- **مصطلحات تقنية**

- عدد أحرف النص يسمى Length
- رقم الخانة يسمى Index
- أرقام الخانات تسمى Indices
- إذا قمنا بأخذ جزء من النص الجزء المأخوذ يسمى Substring

أنت كمبرمج يمكنك استغلال أرقام الخانات لتصل لمحتوى النص كالتالي:

- **تجزيء النص في بايثون بواسطة العامل**

لأخذ جزء من أي نص نعتمد على أرقام الخانات التي يتم إعطاؤها لكل حرف في النص.

- **مثال**

في المثال التالي سنقوم بعرض جزء محدد من النص الذي يحتويه المتغير s فعلياً، سنحدد أننا نريد عرض جميع الأحرف الموجودة ابتداءً من الخانة رقم 11وصولاً إلى ما قبل الخانة رقم 18.

```
Test.py
# هنا قمنا بتعريف متغير اسمه s يحتوي على نص
s = 'welcome to harmash.com'
 # هنا قمنا بطباعة جميع الأحرف الموجودة ابتداءً من الخانة رقم ١١ وصولاً إلى الخانة الموجودة ما قبل الخانة ١٨ في المتغير
                                                                                                    s

print(s[11:18])
```

سنحصل على النتيجة التالية عند التشغيل..

```
harmash
```

- **الرموز التي من خلالها نعرف نهاية السطر في بايثون**

الرمز الذي يتم وضعه للإشارة لنهاية السطر يختلف باختلاف أنواع التقنيات المستخدمة لحفظ البيانات التي نتعامل معها، **الرموز التالية كلها تعني نهاية السطر:**

الرمز	معناه بالإنجليزية
\n	Line Feed
\r	Carriage Return
\r\n	Carriage Return + Line Feed
\x0b or \v	Line Tabulation
\x0c\ or \f	Form Feed
\x1c	File Separator
\x1d	Group Separator
\x1e	Record Separator
\x85	Next Line (C1 Control Code)
\u2028	Line Separator
\u2029	Paragraph Separator

- **مثال**

في المثال التالي سنقوم بتخزين نص في المتغير s وضعنا فيه ثلاث مؤشرات \n لجعل النص يظهر على ثلاث أسطر عند عرضه.

```
Test.py
# \n هنا قمنا بتعريف متغير نصي اسمه s يحتوي على ٣ موشرات
s = 'This is fist line.\nThis is second line.\nThis is third line.'
# هنا قمنا بطباعة النص الموجود في المتغير s و الذي سيظهر على ٣ أسطر
print(s)
```

سنحصل على النتيجة التالية عند التشغيل.

```
This is fist line.
This is second line.
This is third line.
```

لاحظ أن الرمز \n لم يظهر في نهاية كل سطر، بل أدى فقط للنزول على السطر.

- **ملاحظة**

عندما تكون تستخدم هاتفك أو حاسوبك تقوم في العادة بالنقر على الزر Enter كلما أردت أن تنزل على سطر جديد، الذي عليك معرفته كمبرمج هو أن سبب النزول على سطر جديد هو أنه قد تم إضافة رمز خاص بدون علمك في المكان الذي نقرت فيه على الزر Enter للإشارة إلى نهاية السطر، الرمز الذي يضاف يقال له مؤشر نهاية السطر (End Of Line Flag).

❖ شرح موجز لاستخدامات الدمج للسلاسل النصية

- **استخدام العامل (+) لدمج السلاسل النصية :**

يمكن ربط اكثر من سلسلتين باستخدام العامل + كما في المثال التالي

```
"a1 = "Learn
"a2 = "Python
a3 = a1 + a2
a4 = a1 + " " + a2
print(a3)
print(a4)
```

سيكون الناتج

```
LearnPython
Learn Python
```

- **استخدام العامل (*) لدمج وتكرار السلاسل النصية :**

يمكن الحاق نفس السلسلة بسلسلة أخرى باستخدام العامل * كما في المثال التالي

```
"string = "Hello
print("String 1:", string)
string = string * 3
print("Concatenated same string:", string)
```

سيكون الناتج

```
String 1: Hello
Concatenated same string: HelloHelloHello
```

- **استخدام العامل (٪) لدمج السلاسل النصية**

يدمج العامل (٪) سلسلة في سلسلة أخرى هذه العملية تسمى استيفاء سلسلة بايثون . **يمكنك** إضافة سلسلة إلى أي موضع في سلسلة موجودة باستخدام العامل (٪) . لاحظ أن العامل (+) يضيف قيمة إلى نهاية السلسلة ، ولكن العامل (٪) يمكنه إضافة قيمة إلى الموضع الذي تحدده في السلسة. يتيح لك ايضاً العامل (٪) إدراج سلسلة في سلسلة أخرى .يمكننا استخدام العامل (٪) لدمج سلسلتين في بايثون. **كما في المثال التالي**

```
"a1 = "Learn
"a2 = "Python
a3 = "%s %s" % (a1, a2)
a4 = "%s%s" % (a1, a2)
print(a3)
print(a4)
```

سيكون الناتج

```
Learn Python
LearnPython
```

❖ شرح موجز لاستخدام فهرسة و تقطيع البايثون

- يصل [0] إلى العنصر الأول
- يصل [–4] إلى العنصر الرابع من النهاية
- يصل [2:] إلى قائمة بالعناصر من الثالث إلى الأخير.
- يصل [: 4 إلى قائمة بالعناصر من الأول إلى الرابع.
- يصل [2: 4] إلى قائمة بالعناصر من الثالث إلى الخامس.
- يصل [1 :: 2] إلى عناصر بديلة ، بدءاً من العنصر الثاني.

مثال

```
list= [1,2,3,4,5]
print(list[0])
print(list[1])
print(list[2])
print(list[3])
  # تقطيع عناصر القائمة
print(list[0:6])
```

سيكون الناتج

```
1
2
3
4
[1,2,3,4,5]
[3 ,4]
[1,2,3,4,5]
```

❖ المجموعات sets و التوبيلات Tuples

التوبيلات Tuples

نوع البيانات Tuple يُشبه القائمة ولكنه غير قابل للتعديل (ثابتة-immutable):

```
tup = (1, 2, 3)

tup[0]      # => 1

tup[0] = 3  # Raises a TypeError
```

في حالة وجود عنصر واحد في tuple لابد من وضع فاصلة عادية بعد العنصر، أما في حالة وجود أكثر من عنصر فتصبح الفاصلة الأخيرة إضافية:

```
type((1))   # => <class 'int'>

type((1,))  # => <class 'tuple'>

type())    # => <class 'tuple'>
```

يُمكننا تنفيذ أغلب عمليات القوائم على Tuples:

```
len(tup)         # => 3

tup + (4, 5, 6)  # => (1, 2, 3, 4, 5, 6)

tup[:2]          # => (1, 2)

2 in tup         # => True
```

يُمكن تفريغ unpacking محتويات Tuples وكذلك القوائم في متغيرات كما في الأمثلة التالية:

```
a, b, c = (1, 2, 3)  # a = 1, b = 2, c = 3

a, *b, c = (1, 2, 3, 4)  # a = 1, b = [2, 3], c = 4
```

من خلال tuple يُمكننا تبديل قيم المتغيرات بطريقة سهلة وسريعة:

```
e, d = d, e  # d = 5, e = 4
```

المجموعات sets

لتعريف مجموعة فارغة نستخدم الباني الخاص بالمجموعات كما يلي:

```
empty_set = set()

some_set = {1, 1, 2, 2, 3, 4}  # some_set is now {1, 2, 3, 4}
```

نوع البيانات الخاص بعناصر المجموعات لابد أن يكون ثابتة immutable وهذا يعني أنه لا يُمكن استخدام عناصر من نوع القائمة في المجموعات:

```
invalid_set = {[1], 1}  # => Raises a TypeError: unhashable type: 'list'

valid_set = {(1,), 1}
```

للإضافة على المجموعة نستخدم الدالة add:

```
filled_set.add(5)  # filled_set is now {1, 2, 3, 4, 5}
```

إجراء عملية التقاطع بين مجموعتين نستخدم العملية & :

```
other_set = {3, 4, 5, 6}

filled_set & other_set  # => {3, 4, 5}
```

إجراء عملية الاتحاد بين مجموعتين نستخدم العملية | :

```
filled_set | other_set  # => {1, 2, 3, 4, 5, 6}
```

إجراء عملية الطرح بين مجموعتين:

```
{1, 2, 3, 4} - {2, 3, 5}  # => {1, 4}
```

لإجراء عملية فرق التماثل بين مجموعتين:

```
{1, 2, 3, 4} ^ {2, 3, 5}  # => {1, 4, 5}
```

لفحص إذا كانت المجموعة على الشمال هي مجموعة تحتوي المجموعة على اليمين أم لا:

```
{1, 2} >= {1, 2, 3} # => False
```

عكس المثال السابق:

```
{1, 2} <= {1, 2, 3} # => True
```

فحص وجود قيمة في مجموعة:

```
2 in filled_set   # => True

10 in filled_set  # => False
```

❖ الوحدات النمطية

التعابير النمطية يقال لها Regular Expressions أو RegEx وهي عبارة عن نصوص تحتوي على أحرف ورموز لها معاني محددة، للدقة أكثر، كل حرف أو رمز نضعه في هذا النص يعني شيء معين.

إذاً.. التعابير النمطية تستخدم بهدف البحث في النصوص بطريقة سهلة جداً بدل الحاجة إلى كتابة خوارزميات معقدة من أجل الوصول إلى النتيجة المرجوة، وبالتالي يمكنك استغلال التعابير النمطية في حال أردت البحث في النص عن شيء محدد بهدف إجراء تعديل أو تحديث عليه.

من أشهر الاستخدامات للتعابير النمطية هي عندما تطلب من المستخدم إدخال بريده الإلكتروني وإدخال كلمة سر، عندها تجد أنك أثناء كتابة البريد الإلكتروني وكلمة المرور فإنه يظهر لك تنبيهات في حال لم تقم بإدخال بريد إلكتروني صحيح أو كلمة سر لها شكل معين.

فمثلاً تجد أنه يطلب منك وضع كلمة سر تتألف من ثمانية أحرف على الأقل ويجب أن تحتوي على حرف كبير، حرف صغير، رقم ورمز.

- **التعامل مع التعابير النمطية**

re هو موديول جاهز في بايثون يحتوي على دوال و ثوابت جاهزة مخصصة للتعامل مع التعابير النمطية، لهذا السبب عليك تضمين هذا الموديول عندما تريد التعامل مع التعابير النمطية.

لتضمين الموديول re نكتب السطر التالي:

```
import re
```

- **دوال الموديول re**

بعد تضمين الموديول re نصبح قادرين على استخدام الدوال الموجودة فيه **ومن أهمها:**

١. **search (pattern, string, flags=0)**

تبحث في النص الذي نمرره له مكان البارامتر string لترى ما إذا كان يتطابق أو فيه جزء يتطابق مع التعبير النمطي الذي نمرره لها مكان البارامتر pattern في حال تم إيجاد جزء أو أكثر في النص يتطابق مع التعبير النمطي، تقوم بإرجاع كائن من الكلاس Match يحتوي على معلومات أول مكان في هذا النص تطابق مع التعبير النمطي.

- في حال لم يتم العثور على النص المراد البحث عنه ترجع None.

٢. **findall (pattern, string, flags=0)**

تبحث في النص الذي نمرره له مكان البارامتر string لترى ما إذا كان يتطابق أو فيه جزء يتطابق مع التعبير النمطي الذي نمرره لها مكان البارامتر pattern، في حال تم إيجاد جزء أو أكثر في النص يتطابق مع التعبير النمطي، ترجع list كل عنصر فيه يمثل الجزء الذي يتطابق مع التعبير النمطي.

- في حال لم يتم إيجاد أي تطابق ترجع list فارغ.

٣. **finditer (pattern, string, flags=0)**

تبحث في النص الذي نمرره له مكان البارامتر string لترى ما إذا كان يتطابق أو فيه جزء يتطابق مع التعبير النمطي الذي نمرره لها مكان البارامتر pattern، في حال تم إيجاد جزء أو أكثر في النص يتطابق مع التعبير النمطي ترجع iterator كل عنصر فيه عبارة عن كائن Match يمثل الجزء الذي يتطابق مع التعبير النمطي.

- في حال لم يتم إيجاد أي تطابق ترجع list فارغ.

٤. **split (pattern, string, maxsplit=0, flags=0)**

ترجع نسخة من النص الذي نمرره لها مكان الباراميتر string مقسمة على شكل مصفوفة (list) من النصوص، مكان الباراميتر pattern نمرر تعبير نمطي يحدد الطريقة التي سيتم على أساسها تقسيم النص ووضع كل قسم فيه في عنصر واحد بداخل المصفوفة.

- في حال لم يتم العثور على النص المراد البحث عنه ترجع list فيه عنصر واحد يحتوي على كل النص الذي تم البحث فيه، مكان الباراميتر maxsplit يمكنك تمرير رقم يحدد إلى كم قسم تريد أن يتم تقسيم النص.

٥. **sub (pattern, repl, string, count=0, flags=0)**

ترجع نسخة من النص الذي نمرره لها مكان الباراميتر string ، مع تبديل كل جزئية فيها تتطابق مع التعبير النمطي الذي نمرره لها مكان الباراميتر pattern بالنص الذي نمرره لها مكان الباراميتر repl مكان الباراميتر count يمكنك تمرير رقم أكبر من صفر يمثل أول كم جزئية يتم العثور عليها تريدها أن تتبدل.

- **السلاسل المميزة**

السلاسل المميزة (Special Sequences) عبارة عن مجموعة أحرف يصبح لها معنى خاص حين يتم وضعها بعد الرمز / **ولقد ذكرناهم بالجدول التالي:**

الرمز	استخدامه
\w	يطابق أي حرف بين a-z أو A-Z أو أي رقم بين ٠-٩ أو الرمز هذا الرمز يعتبر اختصار للتعبير النمطي [_a-zA-Z0-9].
\W	يطابق أي أحرف ليس بين a-z أو A-Z أو رقم أو الرمز ، هذا الرمز يعتبر اختصار للتعبير النمطي [^_a-zA-Z0-9].
\d	يطابق أي حرف يمثل رقم بين ٠ و ٩ ، هذا الرمز يعتبر اختصار للتعبير النمطي [٠-٩].
\D	يطابق أي أحرف لا يمثل رقم بين ٠ و ٩ ، هذا الرمز يعتبر اختصار للتعبير النمطي [^٠-٩].
\s	يطابق أي حرف يمثل مسافة فارغة، هذا الرمز يعتبر اختصار للتعبير النمطي [\t\n\r\f\v]
\S	يطابق أي حرف لا يمثل مسافة فارغة، هذا الرمز يعتبر اختصار للتعبير النمطي [^ \t\n\r\f\v]
\Z	يطابق أي سلسلة أحرف نمررها قبله مع الأحرف الموجودة في نهاية النص.
\A	يطابق أي سلسلة أحرف نمررها بعده مع الأحرف الموجودة في بداية النص.
\b	يطابق أول أو آخر أي سلسلة أحرف تحتوي على أحرف بين a-z أو A-Z أو أي رقم بين ٠-٩ أو الرمز. -في حال تم وضع التعبير في آخر سلسلة الأحرف فإنه يبحث عن تطابق موجود في آخر كل سلسلة أحرف غير مقطوعة بمسافة فارغة في النص. -في حال تم وضع التعبير في أول سلسلة الأحرف فإنه يبحث عن تطابق موجود في أول كل سلسلة أحرف غير مقطوعة بمسافة فارغة في النص.
\B	لا يطابق أول أو آخر أي سلسلة أحرف تحتوي على أحرف بين a-z أو A-Z أو أي رقم بين ٠-٩ أو الرمز - في حال تم وضع التعبير في آخر سلسلة الأحرف فإنه يبحث عن تطابق غير موجود قبل آخر كل سلسلة أحرف غير مقطوعة بمسافة فارغة في النص. - في حال تم وضع التعبير في أول سلسلة الأحرف فإنه يبحث عن تطابق غير موجود بعد أول كل سلسلة أحرف غير مقطوعة بمسافة فارغة في النص.

- **انتبه**

الرمز \ يسمى ,Backslash وهو Escape Character أي حرف له معنى خاص في بايثون كما لاحظت في الجدول السابق.

- في حال كنت تريد البحث عن الحرف \ نفسه، فعليك معرفة أن مفسّر لغة بايثون يرى كل \ محرفين، أي يراه \\ ، وبالتالي سيكون عليك وضع \\\\ كلما كنت تقصد \ حتى يفهم مفسر لغة بايثون أنك تريد البحث عن هذا الحرف ولا تقصد شيء آخر.
- إذا لم ترد فعل ذلك فيمكنك ببساطة اعتماد أسلوب يقال له ,Raw String أي أن تضع الحرف r قبل نص التعبير النمطي مباشرةً كالتالي "r"\Bea .
- طبعاً كنت تستطيع كتابة التعبير السابق هكذا أيضاً "\\Bea" .

في النهاية.. ننصحك باعتماد أسلوب الـ Raw String لأنه أسهل وأقل تعقيداً.

- **الكائن Match**

الكائن Match الذي ترجعه بعض دوال الموديول re يوفر لك أيضاً أربع دوال أساسية للتعامل مع المعلومات التي يحتويها.

- **اسم الدالة مع تعريفها**

١. (group)ترجع سلسلة الأحرف التي تم إيجادها في النص و التي تم تخزينها في كائن الـ ()Match الذي قام باستدعائها.

٢. (start) ترجع عدد صحيح يمثل Index أول حرف في سلسلة الأحرف التي تم إيجادها في النص و التي تم تخزينها في كائن الـ ()Match الذي قام باستدعائها.

٣. (end)ترجع عدد صحيح يمثل Index آخر حرف في سلسلة الأحرف التي تم إيجادها في النص و التي تم تخزينها في كائن الـ ()Match الذي قام باستدعائها.

٤. (span) تم تخزينها ترجع tuple يمثل أين تم إيجاد سلسلة الأحرف التي تم تخزينها في كائن الـ ()Match الذي قام باستدعائها.

هذا الـ tuple يحتوي على رقمين فقط، الرقم الأول يمثل Index أول حرف وجد عنده التطابق، الرقم الثاني يمثل Index آخر حرف وجد عنده التطابق

أمثله:

التعامل مع الوقت والتاريخ

```
import datetime
now = datetime.datetime.now()
print(now)
```

اتخاذ القرارات وتنفيذ الأعمال المتكررة

```
 #if-else statements
if x > 5:
    print("x is greater than 5("
else:
    print("x is not greater than 5("

 #for loop
for i in range(5):
    print(i)

 #while loop
count = 0
while count < 5:
    print(count)
    count += 1
```

الوحدات النمطية

```
import os
print(os.getcwd())  # طباعة المسار الحالي للعمل

import random
print(random.randint(1, 10))  # طباعة رقم عشوائي بين ١ و ١٠
```

مثال:

```
# تعريف دالة في بايثون
def greet(name:(
```

```
    return f"Hello, {name"!{

# تعريف المتغيرات وأنواع البيانات في بايثون
x = 10  # int
y = 3.14  # float
name = "Alice"  # str
is_valid = True  # bool

# التعامل مع الوقت والتاريخ
import datetime
now = datetime.datetime.now()

# التعامل مع المعاملات
a = 10
b = 3

# استخدام العمليات الحسابية
addition = a + b
subtraction = a - b
multiplication = a * b
division = a / b

# اتخاذ القرارات وتنفيذ الأعمال المتكررة
if x > 5:
    decision = "x is greater than 5"
else:
    decision = "x is not greater than 5"

#for استخدام حلقة
for_loop_result[] =
for i in range(5:(
    for_loop_result.append(i(
```

```
 استخدام حلقة #while
while_loop_result[] =
count = 0
while count < 5:
    while_loop_result.append(count(
    count += 1

التعامل مع السلاسل النصية #
s = "Hello, World"!
uppercase = s.upper()
lowercase = s.lower()
split_string = s.split(",")

الوحدات النمطية #
import os
import random

current_directory = os.getcwd()
random_number = random.randint(1, 10(
```

```
# طباعة النتائج
print(greet(name((
print("x:", x(
print("y:", y(
print("name:", name(
print("is_valid:", is_valid(
print("Current date and time:", now(
print("Addition:", addition(
print("Subtraction:", subtraction(
print("Multiplication:", multiplication(
print("Division:", division(
print("Decision:", decision(
print("For loop result:", for_loop_result(
print("While loop result:", while_loop_result(
print("Uppercase string:", uppercase(
print("Lowercase string:", lowercase(
print("Split string:", split_string(
print("Current directory:", current_directory(
print("Random number:", random_number(
```

الفصل الثاني

التعامل مع الدوال

في هذا الفصل سنتعرف على المواضيع التالية:

- تعريف الدوال وأنواعها واستخداماتها
- دوال الإدخال والإخراج
- إنشاء الدوال واستدعائها
- تمرير المتغيرات للدالة
- إعادة القيمة من الدالة
- نطاق رؤية المتغيرات

❖ تعريف الدوال وأنواعها واستخداماتها

الدالة (Function) عبارة عن مجموعة أوامر مجمعة في مكان واحد وتتنفذ عندما نقوم باستدعائها، بايثون تحتوي على مجموعة كبيرة جداً من الدوال الجاهزة مثل الدوال print() و min() و max() وغيرهم من الدوال

والدالة في بايثون هي كتلة من التعليمات البرمجية التي يتم تنفيذها عند استدعائها. تتيح الدوال تقسيم البرنامج إلى وحدات أصغر وأكثر قابلية للإدارة. يمكن استخدام الدوال لإعادة استخدام الكود وتقليل التكرار وتحسين تنظيم الكود.

- **استخدامات الدوال**

تقسيم المهام: تجزئة المهام الكبيرة إلى أجزاء أصغر.

إعادة استخدام الكود: يمكن استدعاء الدوال في أي وقت ومن أي مكان في الكود.

تحسين التنظيم: تنظيم الكود وجعله أكثر قابلية للفهم والصيانة.

- الدوال الجاهزة في بايثون يقال لها Built-in Functions
- الدوال التي يقوم المبرمج بتعريفها يقال لها User-defined Functions

❖ دوال الإدخال والإخراج

يعتبر الإدخال في لغات البرمجة وخصوصا في لغة بايثون حيث سوف تستخدمها في كل برنامج أو مشروع بحيث لا يمكن تنفيذ البرنامج إلا بدون تدخل المستخدم وذلك يترتب عليه استعمال البرنامج المستخدم للبرنامج أو المشروع وسوف نستخدم في الإدخال بعض الدوال الجاهزة والتي تساعد في العمل في المشروع و هذا من الأشياء التي تجعل من لغة بايثون من افضل اللغات حيث تعطيك إمكانية باستخدام دالة جاهزة ،بحيث تقوم بتعريف متغير من أي نوع وسوف نقوم باستخدام دالة الإدخال التي بدورها تعلن عن المتغير و تقوم بإظهار الرسالة التي تريد عرضها على الشاشة واستقبال البيانات من الشاشة ،بحيث تقوم بتخزين هذه البيانات ووضعها في متغير بحيث استخدامه فيما بعد بحيث اذا كنت تستخدم الأرقام تستطيع استعمال العمليات المنطقية عليها كالجمع و الطرح والضرب والقسمة وجميع العمليات المعقدة دالة الإدخال في البايثون هي دالة جاهزة الاستخدام وهي input .

وتعتبر هذه الدالة كما ذكرنا هي أساس أي مشروع وتكون هذه الدالة من أكثر من مهمه بحيث أنها تقوم بتعريف عن متغير وطباعة رسالة على الشاشة واستقبال البيانات وتخزينها في المتغير المعلن عنه هذه الدالة قامت بالعديد من المميزات وقامت بتوفير الوقت والمجهود وأعطاها سرعة عالية، حيث قامت هذه الدالة في لغة بايثون بدمج العديد من الاكواد في دالة واحده **وقامت أيضا بعمل التالي:**

- الإعلان عن متغير.
- طباعة جملة على الشاشة.
- استقبال بيانات من المستخدم.
- حفظ وتخزين البيانات في المتغير المعلن في بداية استخدام الدالة.

- سوف نقوم بعمل مشروع صغير عبارة عن استخدام لدالة input المتعلقة بالإدخال في بايثون وسف نقوم بعمل متغير ونسميه a ونقوم بعرض رسالة لمعرفة اسم المستخدم الذي يستخدم البرنامج وسوف يتم تخزين الاسم في متغير a الذي قد أعلناه في بداية الكود وسوف نقوم بتخزين نص فيجب إدخال النص في داخل علامات تنصيص لكي تتجنب جميع الأخطاء التي تتعلق بنوع التخزين في المتغيرات.

```
a = input("enter your name :")
```

```
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:19:30) [MSC v.1500 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Python27\aa.py ==========
enter your name :"abdallah ahmed"
>>>
```

- سوف نقوم بعمل مشروع صغير وهو استقبال بيانات من المستخدم، ولكن سوف نقوم بتخزين أرقام فقط وسوف نقوم باستقبال رقم وتخزينه واستعماله في برامج ومشاريع أخرى وسوف تقوم بعمل متغير وسوف نقوم بتسمية المتغير num ونقوم بطلب من المستخدم إدخال أي رقم لتخذينه في المتغير الذي قد أنشأناه في بداية البرنامج

```
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:19:30) [MSC v.1500 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Python27\aa.py ==========
Enter any Number :10
>>>
```

- **شرح استخدام دالة print**

سوف نقوم باستخدام هذه الدالة في الطباعة فقط وسوف نعرض لكم بعض الأمثلة على الشرح

```
print("welcom in prog egypt");
```

نقوم في هذا الكود عمل طباعة لجملة welcome in prog Egypt وسوف نستخدم دالة print كما ظاهر في الكود السابق.

سوف نقوم بعمل مثال صغير مزيج بين الإدخال والخراج وسوف نستعمل print و input وسوف نعرض لكم المشروع وذلك بمثابة الخطوة الأولى في عالم البرمجة باستخدام لغة بايثون

```
print("welcom in prog egypt")
text = input("Enter your name :")
print("welcom :",text)
```

في هذا البرنامج قمنا باستخدام دالة الطباعة print وقد قمنا بطباعة جملة بسيطة على الشاشة للمستخدم وفي السطر الثاني قمنا بالإعلان عن التغير وقد قمنا بتسمية المتغير باسم text وقد قمنا بالإعلان عن المتغير باستخدام دالة input وقد عرضنا على الشاشة جملة enter your name وقد قمنا بتسجيل القيمة في هذا المتغير لنستعمله فيما بعد في السطر الذي يليه باستعمال دالة الطباعة print

وقد قمنا بعمل مزيج من دمج النصوص بوضع علامة الدمج بين جملة welcome والمتغير text وسوف يتم عرض النتيجة

```
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:19:30) [MSC v.1500 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Python27\aa.py ==========
welcom in prog egypt
Enter your name :"abdallah ahmed"
welcom :abdallah ahmed
>>>
```

❖ إنشاء الدوال واستدعائها

الدوال أو الوظائف (functions)عبارة عن مجموعة من التعليمات البرمجية المنظمة والقابلة لإعادة الاستخدام التي يتم استخدامها لتنفيذ إجراء واحد ذي صلة، تمنحك Python العديد من الدوال الـ built in مثل ()print و ()len وما إلى ذلك لكن ماذا لو تريد إنشاء دالة بنفسك لعمل مهمة معينة؟ يمكنك إنشاء دوال خاصة، تسمى' defined functions وهي الدوال التي يتم إنشاؤها من قبل المستخدم.

- **إنشاء دالة في بايثون**

فيما يلي بعض القواعد البسيطة لإنشاء دالة في Python :

١. تبدأ الدالة بالكلمة الأساسية def.

٢. متبوعة باسم الدالة لتعريف الوظيفة بشكل فريد.

٣. ثم قوسين مستديرين () لنمرر القيم (parameters أو arguments) إلى الدالة من خلالها.

٤. نقطتان (:) لتحديد نهاية رأس الدالة.

٥. ثم البدء بكتابة العبارات البرمجية (statements) التي تشكل جسم الوظيفة، يجب أن تحتوي العبارات على نفس مستوى المسافة البادئة.

- **بناء جملة الدالة Syntax of Function**
  - **مثال للتوضيح**

```
def functionname():
    function_suite
```

def: تعني أنك تعرف دالة جديدة.

functionname: نضع مكانها الاسم الذي نعطيه للدالة, و الذي من خلاله يمكننا استدعاءها.

(): بداخل القوسين يمكنك وضع باراميترات و يجب أن تضع: مباشرةً بعد القوسين و من ثم تنزل على سطر جديد لتبدأ بكتابة الأوامر التي ستتنفذ عند استدعاء الدالة.

function_suite: تعني الأوامر التي سنضعها في الدالة و التي ستتنفذ عند استدعائها.

- مثال

إنشاء دالة واستدعائها:

```
# my_function هنا قمنا بتعريف دالة إسمها

def my_function():

print('My first function is called')

# هنا قمنا باستدعاء الدالة my_function حتى يتنفذ الأمر الموضوع فيها

my_function()
```

سنحصل على النتيجة التالية عند التشغيل

```
My first function is called
```

❖ تمرير المتغيرات للدالة

يمكن تمرير المعلومات إلى الدالة كـ arguments أو parameters ، يتم وضع الـ parameters بعد اسم الدالة، داخل القوس () يمكنك إضافة العديد من الـ parameters كما تريد، فقط افصل بينها بفاصلة (,) كالتالي:

```
def printMyName(firstName, secondName):
print("My first name is: " + firstName + " & second name is: " +
secondName )
printMyName("Hamed", "Esam")
```

مخرجات الكود السابق

```
My first name is: Hamed & second name is: Esam
```

بشكل افتراضي، يجب استدعاء دالة بعدد الـ parameters الصحيحة، وهذا يعين أنه إذا كانت دالتك تتوقع (two parameters) ،فيجب عليك استدعاء الدالة باستخدام (two parameters) ليس أكثر ولا أقل.

إذا قمت باستدعاء الدالة السابقة بدون تمرير بيانات أو بيانات ناقصة أو زيادة، فستحصل على خطأ، كالتالي:

```
#--- First example ---
def printMyName(firstName, secondName):
print("My first name is: " + firstName + " & second name is: " +
secondName)
printMyName()    # outputs: TypeError: printMyName() missing 2 re-
quired positional arguments
#--- Second example ---
def printMyName(firstName, secondName) :
print("My first name is: " + firstName + " & second name is: " +
secondName )
printMyName("Hamed") # outputs: TypeError: printMyName() missing 1
required positional arguments
#--- Third example ---
def printMyName(firstName, secondName) :
print("My first name is: " + firstName + " & second name is: " +
secondName )
printMyName("Hamed", "Esam", "Mohamed")      # outputs: TypeError:
printMyName() takes 2 positional arguments but 3 were given
```

❖ إعادة القيمة من الدالة

لإرجاع قيمة للدالة، استخدم عبارة أو كلمة return كالتالي:

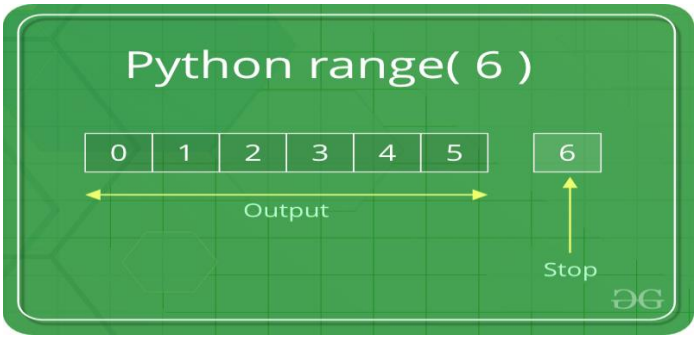
```
def sum(num1, num2):
return num1 + num2
print(sum(10, 2)) # outputs: 12
```

❖ نطاق رؤية المتغيرات

بعبارات بسيطة، يسمح النطاق () للمستخدم بتوليد سلسلة من الأرقام ضمن نطاق معين، اعتماداً على عدد الوسائط التي يقوم المستخدم بتمريرها إلى الوظيفة، يمكن للمستخدم تحديد المكان الذي ستبدأ فيه سلسلة الأرقام هذه ونهايتها، بالإضافة إلى حجم الاختلاف بين رقم وآخر، يمكن تهيئة الدالة () Python range بثلاث نطاقات:

١. النطاق (strop) يأخذ حجة واحدة

عندما يقوم المستخدم باستدعاء النطاق () مع وسيطة واحدة، سيحصل المستخدم على سلسلة من الأرقام التي تبدأ من 0 وتتضمن كل رقم كامل حتى، ولكن لا يشمل الرقم الذي قدمه المستخدم على أنه المحطة.

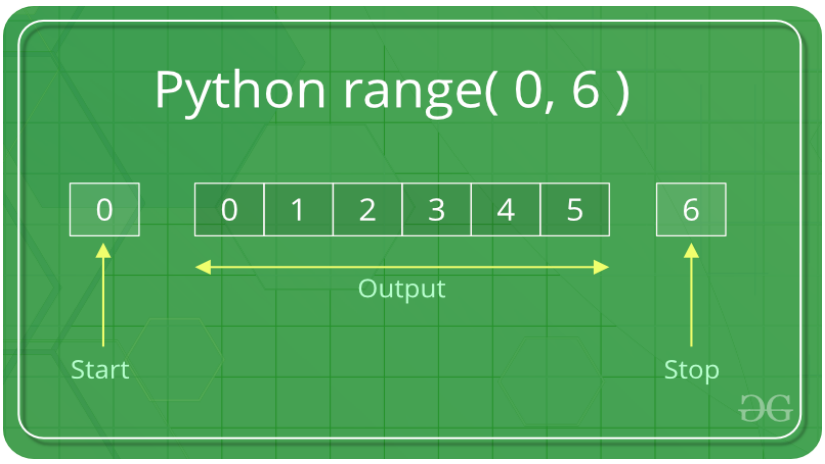


Example:

```
# printing first 6
# whole number
for i in range(6):
    print(i, end=" ")
print()
Output:
0 1 2 3 4 5
```

٢. النطاق (start ، stop) يأخذ وسيطتين

عندما يكون نطاق استدعاء المستخدم () مع وسيطتين، يجب على المستخدم أن يقرر ليس فقط مكان توقف سلسلة الأرقام ولكن أيضاً من أين تبدأ، لذلك لا يتعين على المستخدم البدء من الصفر طوال الوقت، يمكن للمستخدمين استخدام النطاق () لتوليد سلسلة من الأرقام من X إلى Y باستخدام النطاق (X، Y)



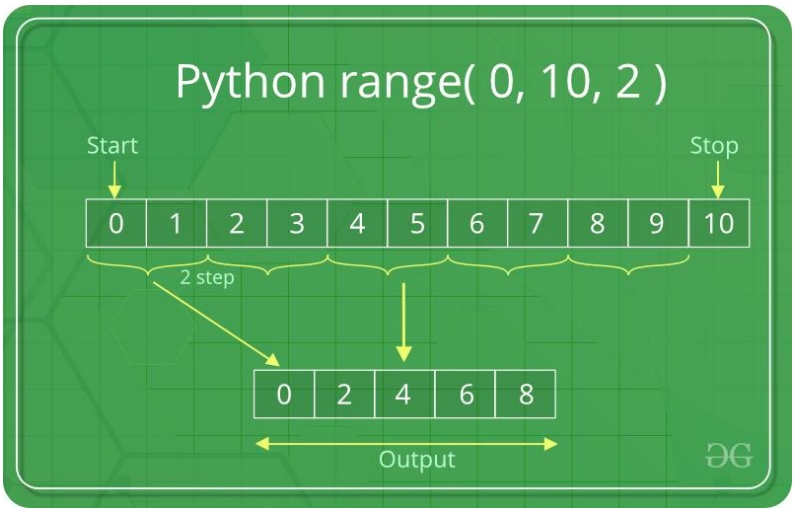
Example:

```
# printing a natural

# number from 5 to 20

for i in range(5, 20):

    print(i, end=" ")
```

```
Output:
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

٣. النطاق (start ، stop ، step) يأخذ ثلاث حجج

عندما يكون نطاق استدعاء المستخدم () مع ثلاث وسيطات، يمكن للمستخدم أن يختار ليس فقط المكان الذي ستبدأ فيه سلسلة الأرقام وتتوقف، ولكن أيضًا مدى الاختلاف الكبير بين رقم وآخر، إذا لم يقدم المستخدم خطوة، فسيعمل النطاق () تلقائيًا كما لو كانت الخطوة ١ ، في هذا المثال نطبع أرقامًا زوجية بين ٠ و ١٠ ، لذلك نختار نقطة البداية من ٠ (البداية = ٠) وأوقف السلسلة عند ١٠ (توقف = ١٠) لطباعة رقم زوجي ، يجب أن يكون الفرق بين رقم واحد والتالي ٢ (الخطوة = ٢) بعد تقديم خطوة نحصل على الإخراج التالي (٠ ، ٢ ، ٤ ، ٨).



Example:

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

```
Output:
0 2 4 6 8
```

امثله:

نطاق رؤية المتغيرات

المتغيرات المحلية (Local Variables)

```
def my_function:()
    x = 10  # Local variable
    print(x)
my_function()
 #print(x)  # This will cause an error because x is not defined
outside the function
```

المتغيرات العامة (Global Variables)

```
y = 20  # Global variable

def my_function:()
    print(y(
my_function()  # Output: 20
print(y)       # Output: 20
```

استخدام الكلمة المفتاحية global

```
z = 30


def my_function:()
    global z
    z = 40
my_function()
print(z)  # Output: 40
```

مثال مجمع يغطي كل هذه المفاهيم

```
# تعريف الدالة وأنواعها واستخداماتها
# دالة للإدخال
def get_user_input(prompt:(
    return input(prompt(


# دالة للإخراج
def print_message(message:(
    print(message(
```

```
# إنشاء دالة واستدعائها
def greet:()
    print("Hello, World("!

greet()

# تمرير المتغيرات للدالة
def greet_user(name:(
    print(f"Hello, {name("!{

user_name = get_user_input("Enter your name(" :
greet_user(user_name(

# إعادة القيمة من الدالة
def add_numbers(a, b:(
    return a + b

sum_result = add_numbers(3, 5(
print_message(f"The sum is: {sum_result("{
```

```
# نطاق رؤية المتغيرات

global_var = "I'm a global variable"


def scope_example:()

    local_var = "I'm a local variable"

    print(local_var(

    print(global_var(


scope_example()

print(global_var(

 #print(local_var)
```

شرح المثال المجمع:

دالة الإدخال get_user_input: تأخذ نصًا كمدخل وتعيد المدخل الذي يدخله المستخدم.

دالة الإخراج print_message: تأخذ رسالة وتطبعها.

إنشاء دالة واستدعائها greet: دالة بسيطة تطبع "Hello, World!".

تمرير المتغيرات للدالة greet_user: تأخذ اسمًا وتطبع رسالة ترحيبية.

إعادة القيمة من الدالة add_numbers: تأخذ رقمين وتعيد ناتج جمعهما.

نطاق رؤية المتغيرات scope_example: توضح الفرق بين المتغيرات المحلية والعامة

الفصل الثالث

التعامل مع بنية البيانات

في هذا الفصل سنتعرف على المواضيع التالية:

- مقدمة
- الصفوف Tuple
- القوائم Lists
- المجموعات Sets
- القواميس Dictionaries
- امثلة شاملة

❖ مقدمة

هياكل البيانات هي التركيبات الأساسية التي تبني حولها برامجك توفر كل بنية بيانات طريقة معينة لتنظيم البيانات بحيث يمكن الوصول إليها بكفاءة، اعتماداً على حالة الاستخدام الخاصة بك، تأتي Python مع مجموعة واسعة من هياكل البيانات في مكتبتها القياسية.

ومع ذلك، فإن اصطلاح التسمية في Python لا يوفر نفس مستوى الوضوح الذي ستجده في اللغات الأخرى في جافا، القائمة ليست مجرد قائمة – إنها إما قائمة LinkedList أو ArrayList ليس الأمر كذلك في بايثون، حتى مطوري Python ذوي الخبرة يتساءلون أحياناً عما إذا كان نوع القائمة المضمنة يتم تنفيذه كقائمة مرتبطة أو مصفوفة ديناميكية.

❖ الصفوف Tuples

- **مفهوم الكلاس tuple**

الـ tuple عبارة عن مصفوفة لها حجم ثابت، يمكنها تخزين قيم من مختلف الأنواع في وقت واحد ولا يمكن تبديل قيمها.

- **طريقة تعريف tuple**

لتعريف tuple نستخدم الرمز ( ) بداخل هذا الرمز يمكنك تمرير القيم بشكل مباشر له بشرط وضع فاصلة بين كل عنصرين ويمكنك تحديد نوع وعدد العناصر التي تريد وضعها فيه فقط.

- **المثال الأول**

في المثال التالي قمنا بتعريف tuple فارغ أي لا يحتوي أي عنصر.

```
Test.py
A = ()          # هنا قمنا بتعريف tuple فارغ اسمه A
print(A)        # هنا قمنا بعرض ما يحتويه الكائن A كما هو (أي كما قمنا بتعريفه) ( )
```

سنحصل على النتيجة التالية عند التشغيل

```
()
```

- **المثال الثاني**

في المثال التالي قمنا بتعريف tuple يتألف من عنصر واحد فقط.

```
Test.py

A = (10,)           # هنا قمنا بتعريف tuple يتألف من عنصر واحد قيمته ١٠
print(A)            # هنا قمنا بعرض ما يحتويه الكائن A أي كما قمنا بتعريفه ) كما هو (
```

سنحصل على النتيجة التالية عند التشغيل.

```
(10)
```

- انتبه!

وضع فاصلة بعد القيمة 10 هنا يعتبر أمر إجباري حتى يفهم مترجم بايثون أنك تنوي تعريف tuple وليس متغير عادي قيمته 10، كما أنه لا حاجة لوضع فاصلة إضافية كما فعلنا هنا في حال كان الـ tuple يحتوي على أكثر من قيمة.

- **استخدامات الصفوف**

استخدام الصفوف كمفاتيح في القواميس نظرًا لعدم قابليتها للتغيير.

تخزين مجموعات ثابتة من البيانات

❖ القوائم Lists

- **مفهوم الكلاس list**

الـ list عبارة عن مصفوفة ليس لها حجم ثابت، يمكنها تخزين قيم من مختلف الأنواع في وقت واحد ويمكنك تبديل قيمها متى شئت

. القوائم هي مجموعة قابلة للتغيير (mutable) من العناصر المرتبة. يتم تعريفها باستخدام الأقواس المربعة [].

- **طريقة تعريف List**

لتعريف list نستخدم الرمز [ ] بداخل هذا الرمز يمكنك تمرير القيم بشكل مباشر له بشرط وضع فاصلة بين كل عنصرين ويمكنك تحديد نوع و عدد العناصر التي تريد وضعها فيه فقط.

- **المثال الأول**

في المثال التالي قمنا بتعريف list فارغ أي لا يحتوي أي عنصر.

```
Test.py
A = []     # A فارغ اسمه هنا قمنا بتعريف
print(A)   # ( أي كما قمنا بتعريفه ) كما هو A هنا قمنا بعرض ما يحتويه الكائن
```

سنحصل على النتيجة التالية عند التشغيل.

```
[]
```

- **المثال الثاني**

في المثال التالي قمنا بتعريف list وضعنا فيه أعداد صحيحة.

```
numbers = [10, 20, 30, 40, 50]
# اسمه هنا قمنا بتعريف numbers يحتوي على أعداد صحيحة فقط
print(numbers)
# هنا قمنا بعرض محتوى الكائن numbers كما هو (أي كما قمنا بتعريفه)
```

سنحصل على النتيجة التالية عند التشغيل.

```
[10, 20, 30, 40, 50]
```

- **إضافة عنصر في القوائم باستخدام الدوال**

يمكن إضافة عنصر إلى قائمة في بايثون باستخدام الدوال (append)(insert)((extend)

(append) يضيف كل العناصر المنقولة إلى القائمة كعنصر واحد(extend)

(extend) يضيف عناصر إلى القائمة واحدة تلو الأخرى.

(insert) يضيف عنصرًا في فهرس معين إلى القائمة.

امثلة :

```
n [7]: my_list = [7, 2, 1]

In [8]: my_list

Out [8]: [7, 2, 1]

In [9]: my_list.append([44, 15,'python'])

In [10]: my_list

Out [10]: [7, 2, 1, [44, 15, 'python']]

In [11]: my_list.extend(['example',2])

In [12]: my_list

Out [12]: [7, 2, 1, [44, 15, 'python'], 'example', 2]

In [13]: my_list.insert(1, 'insert_example1')

In [14]: my_list.insert(6, 'insert_example2')

In [15]: my_list

Out [15]: [7, 'insert_e1', 2, 1, [44, 15, 'python'], 'example', 'insert_e2', 2]
```

- **حذف عنصر في القوائم باستخدام الدوال**

يمكن إزالة عنصر من قائمة في بايثون باستخدام(pop) (clear) (remove)(وdel)

- **باستخدام الدالة (**clear **) يتم حذف جميع عناصر القائمة .**
- **تحذف الدالة (**pop **)عنصرًا بناءً على الفهرس وتعرض قيمته في المخرجات .**
- **باستخدام الدالة (**remove **)يمكن إزالة عنصر بناءً على قيمته .**
- **باستخدام الدالة (**del **)يمكن حذف عناصر المصفوفة بناءً على الفهرس. الفهرس ألول هو** 0 **والفهرس الأخير هو** -1 **.**

امثلة :

```
In [16]: my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
In [17]: my_list
Out [17]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
In [18]: my_list.clear()
Out [18]: []
In [19]: my_list = [8, 2, 3, 7, 9, 1]
In [20]: my_list.pop(0)
Out [20]: 8
In [21]: my_list.pop(4)
Out [21]: 1
In [22]: my_list = [12, 1, 5, 2, 4]
Out [23]: 5
In [24]: my_list = ['Ali', 'Mohammad', 'Milad', 1, 5.69]
In [25]: my_list.remove('Milad')
In [25]: my_list
Out [25]: ['Ali', 'Mohammad', 1, 5.69]
In [26]: my_list.remove(1)
In [27]: my_list
Out [27]: ['Ali', 'Mohammad', 5.69]
In [28]: my_list = [4, 7, 5, 1, 4]
In [29]: del my_list[0]
In [30]: my_list
Out [30]: [7, 5, 1, 4]
In [31]: del my_list[-1]
In [32]: my_list
Out [32]: [7, 5, 1]
```

- **دوال اخرى**

هناك العديد من الدوال ألألخرى التي يمكن استخدامها عند العمل مع القوائم:

- **ترجع الدالة (**len **) طول القائمة.**
- **ترجع الدالة (**index **) فهرس أحد العناصر , ملاحظة: إذا ظهر عنصر في القائمة عدة مرات ، فسيتم إرجاع الفهرس الأول المطابق.**
- **باستخدام الدالة (**sort **)يتم فرز القائمة بترتيب صعودي.**

```
In [33]: my_list1= [4, 7, 5, 1, 4, 12]
In [34]: len(my_list)
Out [34]: 6
In [35]: my_list.index(5)
Out [35]: 2
In [36]: my_list.sort()
In [37]: my_list
Out [37]: [1, 4, 4, 5, 7, 12]
```

المجموعات sets

الـ set عبارة عن مصفوفة ليس لها حجم ثابت، يمكنها تخزين قيم من مختلف الأنواع في وقت واحد ولا يمكن تبديل أو حذف قيمها بشكل مباشر، كما أنه لا يمكنها أن تحتوي على قيم مكررة أي إذا وضعت فيها نفس القيمة مرتين فإنه سيتم تخزين قيمة واحدة فيها وليس قيمتين.

النوع set لا يحافظ على الترتيب الذي تم فيه إدخال العناصر لأنه لا يضيف رقم Index لكل عنصر كما يفعل النوع list والنوع tuple لذلك لا تستغرب إذا قمت بتخزين مجموعة قيم بداخل set ثم حاولت عرضها لأنك في كل مرة ستقوم فيها بتشغيل البرنامج من جديد ستتبدل أماكن القيم.

يتم تعريف المجموعة (Set)على أنها مجموعة من العناصر الفريدة التي ال تتبع ترتيبًا معينًا.

تُستخدم المجموعات عندما يكون وجود كائن في مجموعة من الكائنات أكثر أهمية من عدد

المرات التي تظهر فيها الكائنات أو ترتبها.في المجموعات، إذا تكررت البيانات أكثر من مرة، يتم

إدخالها في المجموعة مرة واحدة فقط. على عكس الصفوف، فإن المجموعات قابلة للتغيير؛ أي

أنه يمكن تعديلها أو إضافتها أو استبدالها أو إزالتها. يمكن عرض مجموعة مثال على النحو التالي:

```
set_a = {"item 1", "item 2", "item 3",....., "item n"}
```

```
In [1]: my_set = {2, 2, 3, 1, 4, 5, 5, 5}

 In [2]: my_set

Out [2]: {1, 2, 3, 4, 5}
```

يمكنك استخدام الدالة (add) لاضافة عنصر:

```
In [3]: my_set = {8, 1, 5}

In [4]: my_set.add(6)

In [5]: my_set

Out [5]: {1, 5, 6, 8}
```

هنالك عمليات تطبق على مجموعات الرياضيات مثل الاتحاد (Union) والتقاطع (Interaction) وما إلى ذلك. يوضح المثال التالي المجموعة المكونة من اتحاد مجموعتين:

```
In [3]: a = {1, 2, 3, 4, 5}
In [4]: b = {6, 4, 5, 1, 3, 8, 7}
In [5]: a.union(b)
Out [5]: {1, 2, 3, 4, 5, 6, 7, 8}
```

- **طريقة تعريف set**

لتعريف set نستخدم الرمز { } بداخل هذا الرمز يمكنك تمرير القيم بشكل مباشر له بشرط وضع فاصلة بين كل عنصرين.

- **المثال الأول**

في المثال التالي قمنا بتعريف set وضعنا فيه أعداد صحيحة.

```
Test.py

numbers = {10, 20, 30, 40, 50}

# هنا قمنا بتعريف set اسمه numbers يحتوي على أعداد صحيحة فقط

print(numbers)

# هنا قمنا بعرض محتوى الكائن numbers كما هو (أي كما قمنا بتعريفه)
```

سنحصل على نتيجة تشبه النتيجة التالية عند التشغيل لأننا لا نعرف كيف سيتم ترتيب العناصر.

```
{40,10,50,20,30}
```

- **المثال الثاني**

في المثال التالي قمنا بتعريف set وضعنا فيه نصوص.

```
Test.py

names = {'Rami', 'Sara', 'Nada'}

# هنا قمنا بتعريف set اسمه names يحتوي على نصوص فقط

print(names)

# هنا قمنا بعرض ما يحتويه الكائن names كما هو (أي كما قمنا بتعريفه)
```

سنحصل على نتيجة تشبه النتيجة التالية عند التشغيل لأننا لا نعرف كيف سيتم ترتيب العناصر.

```
{'Sara', 'Rami', 'Nada'}
```

- **حذف الـ set بواسطة الجملة del**

الجملة del تستخدم لحذف الـ set كما هي من الذاكرة.

- **مثال**

في المثال التالي قمنا بتعريف set وضعنا فيه أرقام، بعدها قمنا بحذفه من الذاكرة بعدها حاولنا أن نعرض ما يحتويه.

```
Test.py
# هنا قمنا بتعريف set إسمه arr وضعنا فيه أعداد صحيحة
arr = {10, 20, 30, 40, 50}
del arr
# هنا قمنا بحذف الكائن arr كما هو من الذاكرة
print(arr)
# هنا حاولنا عرض ما يحتويه الكائن arr و الذي في الأصل قمنا بحذفه من الذاكرة لذلك سيظهر خطأ عند التشغيل
```

سنحصل على النتيجة التالية عند التشغيل

```
NameError: name 'arr' is not defined
```

❖ القواميس Dictionaries

- **مفهوم الكلاس dict**

في البداية كلمة dict اختصار لكلمة dictionary والتي تعني قاموس أو معجم، الـ dict عبارة عن جدول يتألف من عامودين، الأول يحتوي المفاتيح (Keys) والثاني يحتوي القيم (Values) الخاصة بكل عنصر، كل عنصر يتم إضافته في dict يجب إعطاؤه قيمتين، الأولى تمثل المفتاح (Key) والثانية تمثل قيمته (Value)، المفاتيح تستخدم للوصول إلى القيم لهذا لا يمكن وجود عنصرين في الـ dict عندهم نفس المفتاح إذاً كل Key موضوع يسمح لك بالوصول لقيمة واحدة من القيم الموجودة في الـ dict.

- **طريقة تعريف dict**

لتعريف dict نستخدم الرمز { } بداخل هذا الرمز يمكنك تمرير العناصر بشكل مباشر له بشرط وضع فاصلة بين كل عنصرين.

لا تنسى أن كل عنصر يجب أن يملك قيمتين، الأولى تمثل المفتاح والثانية تمثل القيمة بين كل مفتاح و قيمة نضع الرمز :

- **المثال الأول**

في المثال التالي قمنا بتعريف dict فارغ أي لا يحتوي أي عنصر.

```
Test.py

data = {}

  # data هنا قمنا بتعريف dict فارغ اسمه

print(data)

 #   هنا قمنا بعرض ما يحتويه الكائن data كما هو (أي كما قمنا بتعريفه)
```

سنحصل على النتيجة التالية عند التشغيل.

```
{}
```

- **ملاحظة:**

سواء كنت تقوم بتعريف dict أو set فإنك تستخدم الرمز { } كما لاحظت في المثال السابق، لهذا لا نلاحظ وجود أي فرق بين عرض ما يحتويه dict فارغ أو set فارغ.

بمجرد أن تضيف عنصر واحد بين الرمز { } وتعطيه مفتاح وقيمة عندها سيفهم مفسّر لغة بايثون أنك تقصد تعريف dict و ليس set في المثال التالي قمنا بتعريف dict وضعنا فيه ثلاث عناصر.

المفاتيح وضعناها كأرقام، والقيم وضعناها كنصوص مع الإشارة إلى أنه يمكنك وضع أي نوع تريد، كما أننا قمنا تعريف كل عنصر (أي كل مفتاح وقيمة) على سطر منفرد لتكون قراءته أسهل فقط.

- المثال الثاني

```
# data هنا قمنا بتعريف dict يتألف من ثلاث عناصر,إسمه

data = {
    1: 'Admin',
    2: 'Editor',
    3: 'Reader'
}
#  هنا قمنا بعرض ما يحتويه الكائن data كما هو ( أي كما قمنا بتعريفه)
print(data)
```

سنحصل على النتيجة التالية عند التشغيل.

```
{1: 'Admin', 2: 'Editor', 3: 'Reader'}
```

- المثال الثالث

في المثال التالي قمنا بتعريف dict وضعنا فيه ثلاث عناصر، المفاتيح وضعناها كنصوص والقيم وضعناها كأرقام ونصوص

```
# data هنا قمنا بتعريف dict يتألف من ثلاث عناصر,اسمه
data = {
'id': 1,
'name': 'Mhamad',
'mobile': 70123456
}
# هنا قمنا بعرض ما يحتويه الكائن data كما هو (أي كما قمنا بتعريفه)
print(data)
```

سنحصل على النتيجة التالية عند التشغيل.

```
{'id': 1, 'name': 'Mhamad', 'mobile': 70123456}
```

- **الوصول لقيم عناصر الـdict**

للوصول لأي عنصر في الـ dict سواء للحصول على قيمته أو تغييرها أو حذفها فإننا نستخدم المفتاح الخاص بالعنصر.

- **مثال:**

في المثال التالي قمنا بتعريف dict وضعنا فيه ثلاث عناصر بعدها قمنا باستخدام الرمز [] لعرض قيمة العنصر الذي يملك المفتاح رقم 1.

```
Test.py
# data هنا قمنا بتعريف dict يتألف من ثلاث عناصر,إسمه
data = {
    1: 'Admin',
    2: 'Editor',
    3: 'Reader'
}
# 1 هنا قمنا بطباعة قيمة العنصر الذي يملك المفتاح رقم
print(data[1])
```

سنحصل على النتيجة التالية عند التشغيل.

```
Admin
```

امثله:

المثال الاول :

```
# تعريف صف
person = ("Alice", 30, "Engineer")

# الوصول إلى عناصر الصف
print(person[0])  # Output: Alice
print(person[1])  # Output: 30

# صف داخل صف
coordinates = ((10, 20), (30, 40))
print(coordinates[1][0])  # Output: 30

# محاولة تغيير عنصر في الصف (سيؤدي إلى خطأ)
# person[1] = 31  # TypeError: 'tuple' object does not support item assignment
```

المثال الثاني:

```
# تعريف قائمة
fruits = ["apple", "banana", "cherry"]

# الوصول إلى عناصر القائمة
print(fruits[0])  # Output: apple

# تعديل عنصر في القائمة
fruits[1] = "blueberry"
print(fruits)  # Output: ['apple', 'blueberry', 'cherry']

# إضافة عنصر جديد إلى القائمة
fruits.append("orange")
print(fruits)  # Output: ['apple', 'blueberry', 'cherry', 'orange']

# إزالة عنصر من القائمة
fruits.remove("cherry")
print(fruits)  # Output: ['apple', 'blueberry', 'orange']

# التكرار خلال القائمة
for fruit in fruits:
    print(fruit)
```

المثال الثالث:

```
# تعريف مجموعة
unique_numbers = {1, 2, 3, 4}

# إضافة عنصر إلى المجموعة
unique_numbers.add(5)
print(unique_numbers)  # Output: {1, 2, 3, 4, 5}
```

```
# إزالة عنصر من المجموعة
unique_numbers.remove(3)
print(unique_numbers)  # Output: {1, 2, 4, 5}

# التحقق من وجود عنصر في المجموعة
print(2 in unique_numbers)  # Output: True

# التكرار خلال المجموعة
for number in unique_numbers:
    print(number)

# العمليات على المجموعات
set_a = {1, 2, 3}
set_b = {3, 4, 5}

# الاتحاد
print(set_a | set_b)  # Output: {1, 2, 3, 4, 5}

# التقاطع
print(set_a & set_b)  # Output: {3}

# الفرق
print(set_a - set_b)  # Output: {1, 2}
```

المثال الرابع :

```
# تعريف قاموس
student = {"name": "John", "age": 22, "major": "Computer Science"}

# الوصول إلى قيمة باستخدام المفتاح
print(student["name"])  # Output: John

# تعديل قيمة في القاموس
student["age"] = 23
print(student)  # Output: {'name': 'John', 'age': 23, 'major': 'Com-
puter Science'}

# إضافة زوج مفتاح-قيمة جديد إلى القاموس
student["grade"] = "A"
print(student)  # Output: {'name': 'John', 'age': 23, 'major': 'Com-
puter Science', 'grade': 'A'}

# إزالة زوج مفتاح-قيمة من القاموس
del student["major"]
print(student)  # Output: {'name': 'John', 'age': 23, 'grade': 'A'}

# التكرار خلال القاموس
for key, value in student.items():
    print(f"{key}: {value}")

# التحقق من وجود مفتاح في القاموس
print("name" in student)  # Output: True
print("major" in student)  # Output: False

# استخدام القاموس مع البيانات المعقدة
students = {
```

```
    "student1": {"name": "Alice", "age": 21},
    "student2": {"name": "Bob", "age": 22},
}

print(students["student1"]["name"])  # Output: Alice
```

الفصل الرابع

التعامل مع الملفات

في هذا الفصل سنتعرف على المواضيع التالية:

- فائدة الملفات
- القراءة و الكتابة من ملف read file , write file
- تعديل محتويات ملف
- حذف ملف
- التعامل مع الملفات المضغوطة و التعامل مع CSV , JSON
- امثلة شاملة

❖ فائدة الملفات

التعامل مع الملفات أو معالجة الملفات (Files Handling) يقصد منها إجراء عملية ما على الملفات على مختلف أنواعها مثل mp4 - jpg - txt ، سنتعلم كيف نقرأ محتوى ملف، كيف تنشئ نسخة منه، كيف تعدل محتواه، كيف تحذفه .. إلخ.

❖ القراءة و الكتابة من ملف read file , write file

○ الدالة open()

هذه الدالة هي من الدوال الجاهزة في بايثون وهي تستخدم لإنشاء ملف جديد أو لفتح الملف الذي سيتم التعامل معه، في حال تم إنشاء الملف بشكل صحيح أو تم فتح الملف بشكل صحيح ترجع كائن file يتيح لك التعامل معه، في حال لم تستطع إنشاء الملف أو الوصول إليه ترمي استثناء.

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, new-
line=None)
```

- مكان البارميتر file نمرر نص يمثل اسم الملف الذي سيتم إنشاؤه أو التعامل معه.
- **mode** هو باراميتر اختياري نمرر مكانه حرف (أو أكثر) يمثل كيف سنتعامل مع الملف، مثل: هل تنوي القراءة منه أو الكتابة فيها إلخ.
- **Buffering** هو باراميتر اختياري, يمكنك أن تمرر مكانه رقم يحدد كيف سيتم تخزين الأحرف بشكل مؤقت في الذاكرة أثناء الكتابة أو القراءة من الملف.
- **Encoding** هو باراميتر اختياري, يمكنك أن تمرر مكانه اسم الترميز الذي يجب استخدامه عند التعامل مع الملف.
- **Errors** هو باراميتر اختياري, يمكنك أن تمرر مكانه كلمة لتحدد كيف سيتم التعامل مع الأخطاء التي قد تحدث عند التعامل مع الملف.
- **newline** هو باراميتر اختياري, يمكنك أن تمرر مكانه الرمز الذي يمثل نهاية كل سطر في الملف و الذي يجعل النص الذي يوضع بعده ينزل على سطر جديد.

اهم باراميتر اختياري في هذه الدالة هو الباراميتر mode لأنه كما سبق وقلنا إن الحرف الذي نمرره مكانه يحدد الهدف من فتح الملف.

- مثال

في المثال التالي قمنا بإنشاء ملف نصي جديد اسمه demo.txt في نفس المشروع الذي نعمل فيه،

بعدها قمنا كتابة السطر التالي بداخله Python is an easy language to learn:

```
Test.py

هنا قمنا بإنشاء كائن يشير لملف اسمه 'demo.txt' ووضعنا الرمز 'w' لكي يتم إنشاء الملف ولنستطيع الكتابة فيه #

opened_file = open('demo.txt', 'w')



هنا قمنا باستدعاء الدالة write() من الكائن opened_file للكتابة في الملف الذي يشير إليه #



opened_file.write('Python is an easy language to learn.')



هنا قمنا باستدعاء الدالة close() من الكائن opened_file لإغلاق الإتصال مع الملف المفتوح في الذاكرة #

opened_file.close()
```

بعد تشغيل الملف Test.py سيتم إنشاء ملف اسمه demo.txt في نفس المشروع الذي نعمل فيه وبداخله

النص التالي:

```
Python is an easy language to learn.
```

قراءة المحتوى بالكامل

فتح الملف في وضع القراءة

```
with open('example.txt', 'r') as file:
    content = file.read()
    print(content)
```

❖ كتابة الملفات write file

لكتابة ملف عليك فتحه باستخدام الوضع w كمعامل ثاني للتابع open كما في المثال التالي :

```
>>> ('fout = open ('output.txt', 'w
>>> (print(fout
<'io.TextIOWrapper name='output.txt' mode='w' encoding='cp1252>
```

اذا كان الملف موجودًا بالفعل ، فإن فتحه في وضع الكتابة يؤدي إلى مسح البيانات القديمة و يبدأ من جديد ، لذا كن حذرًا ! أما اذا كان الملف غير موجود فسينشأ ملف جديد.
يعمل تابع الكتابة write الخاص بكائن معرف الملف على وضع البيانات في الملف و إرجاع عدد الآحرف المكتوبة كما نلحظ في المثال الأدنى : إذا ارجعت القيمة ٢٤ التي تمثل عدد الحروف الموجودة في السلسلة النصية المجودة بين علمتي التنصيص "" .
إن الوضع الافتراضي هو ملف نصي في حالتي كتابة السلاسل النصية و قراءاتها .

```
"line1 = "This here's the wattle,\n
(fout.write(line1
24
```

مرة اخرى ، يحفظ كائن الملف مكانه ، لذلك إذا استدعيت تابع الكتابة write مرة اخرى ، فإن البيانات الجديدة ستضاف الى النهاية

يجب ان نتأكد من ادارة نهايات الأسطر في اثناء الكتابة على الملف عن طريق ادراج حرف انشاء السطر الجديد \n ادراجاً صريحاً عندما نريد انهاء السطر . من هنا ينبغي ان نعرف أن تعليمة print تضيف تلقائيا سطرا جديدا ، لكن استعمال التابع write لا يضيف السطر الجديد تلقائيا .

```
>>> 'line2 = 'the emblem of our land.\n
>>> (fout.write(line2
24
```

عند الانتهاء من عملية الكتابة ، يجب عليك اغلاق الملف كما في الشيفرة ادناه للتأكد من كتابة آخر جزء من البيانات فعلياً على القرص حتى لا يضيع هذا الجزء إذا انقطع التيار الكهربائي .

```
>>>fout.close ()
```

يمكننا اغلاق الملفات التي نفتحها للقراءة ايضاً ، و لكن يمكن ان نكون مهملين بعض الشيء اذا كنا نفتح بعض الملفات فقط لأن مفسر بايثون يغلق جميع الملفات المفتوحة عند انتهاء البرنامج > أما عندما نكتب على الملفات فيجب علينا اغلاق الملفات اغلاقًا قاطعًا و ذلك تفاديًا من أي شيء قد يحدث .

❖ تعديل محتوى ملف

بعد فتح الملف الذي تريد التعامل معه بنجاح بواسطة الدالة open() يصبح بإمكانك استخدام الدوال التالية من الكائن الذي سترجعه هذه الدالة.

- **اسم الدالة مع تعريفها**

  - **write(string)**

تستخدم للكتابة في الكائن الذي يمثل الملف المفتوح الذي قام باستدعائها مكان الباراميتر string نمرر النص الذي نريد أن يتم كتابته في الملف.

  - **writelines(aList)**

تستخدم لكتابة مجموعة نصوص مخزنة في list في الكائن الذي يمثل الملف المفتوح الذي قام باستدعائها.

مكان الباراميتر lines نمرر كائن aList فيه مجموعة النصوص التي نريد أن يتم كتابتها بنفس الترتيب في الملف.

  - **read(n = -1)**

تستخدم للقراءة من الكائن الذي يمثل الملف المفتوح الذي قام باستدعائها.

- إذا قمت باستدعائها ولم تمرر لها أي رقم، سترجع كل النص الموجود في الملف دفعة واحد.
- **n** هو باراميتر اختياري يمكنك أن تمرر مكانه رقم يمثل عدد الأحرف التي تريد قراءتها من الملف في حال لم ترد أن تقرأ كل محتوى الملف دفعة واحدة، مع الإشارة إلى أنك في كل مرة تقوم فيها باستدعائها ستعطيك الأحرف التالية الموجودة في الملف.

- **readline(limits = -1)**

تستخدم للقراءة سطراً سطراً من الكائن الذي يمثل الملف المفتوح الذي قام باستدعائها.

- إذا قمت باستدعائها ولم تمرر لها أي رقم، سترجع السطر التالي الموجود في الملف.
- **n** هو باراميتر اختياري يمكنك أن تمرر مكانه رقم يمثل عدد الأحرف التي تريد قراءتها من السطر التالي في الملف في حال لم ترد أن تقرأ كل محتوى السطر دفعة واحدة، مع الإشارة إلى أنك في كل مرة تقوم فيها باستدعائها ستعطيك الأحرف الموجودة حتى نهاية السطر الحالي في الملف.

- **readlines(limits = -1)**

تستخدم لإرجاع نسخة من النص الموجود في الكائن الذي يمثل الملف المفتوح الذي قام باستدعائها ككائن list .

- كل عنصر في كائن الـ list الذي ترجعه يمثل سطر موجود في الملف.
- إذا قمت باستدعائها و لم تمرر لها أي رقم، سيتم وضع كل الأحرف الموجودة على كل سطر في الملف في عنصر من عناصر الكائن الـ list.
- **n** هو باراميتر اختياري يمكنك أن تمرر مكانه رقم يمثل عدد الأحرف التي تريد قراءتها من كل سطر في الملف في حال لم ترد أن تقرأ كل محتوى السطر.

- **tell()**

ترجع رقم آخر حرف في الملف تم الوصول إليه أثناء القراءة من الملف عن طريق الكائن الذي يمثل هذا الملف.

- **seek(offset, from_what=0)**

أثناء القراءة من الملف عن طريق الكائن الذي يمثل هذا الملف، يمكنك استخدام هذه الدالة في حال أردت الرجوع إلى الوراء في الملف لقراءة الملف من جديد على سبيل المثال.

from_what هو باراميتر اختياري ، **يمكنك أن تمرر إحدى الأرقام التالية مكانه:**

- الرقم 0 إذا أردت الرجوع إلى أول حرف في الملف.
- الرقم 1 إذا أردت البقاء عند حرف الحالي الذي وصلت إليه في الملف.
- الرقم 2 إذا أردت الذهاب إلى آخر حرف في الملف.

مكان الباراميتر offset تمرر رقم يمثل بعد كم حرف نسبة للبراميتر from_what تريد أن تبدأ.

مثال: إذا قمت باستدعاء الدالة هكذا seek(0,0) أو هكذا seek(0) فهذا يعني أنك تريد العودة إلى أول حرف في الملف.

- **close()**

تستخدم لإغلاق الاتصال مع الملف وتنظيف الذاكرة من كل ما له علاقة بهذا الملف.

ملاحظة: في حال قمت بفتح الملف بالأساس بواسطة الجملة with فلا داعي لإغلاق الملف لأنها تقوم بإغلاقه بشكل تلقائي عنك.

- **مثال**

```
Test.py
هنا قمنا بإنشاء ملف اسمه 'demo.txt' ووضعنا الرمز 'w' للإشارة الى أننا سنستخدم هذا الكائن لكتابة نص جديد في الملف #

  opened_file = open('demo.txt', 'w')

هنا قمنا باستدعاء الدالة write() من الكائن opened_file لكتابة نص جديد في الملف الذي يشير إليه #

opened_file.write('This new text will replace the old text.')

هنا قمنا باستدعاء الدالة close() من الكائن opened_file لإغلاق الإتصال مع الملف المفتوح في الذاكرة #

opened_file.close()
```

بعد تشغيل الملف Test.py سيتم إنشاء ملف اسمه demo.txt في نفس المشروع الذي نعمل فيه وبداخله النص التالي.

```
This new text will replace the old text.
```

❖ حذف ملف

- **الموديول os**

os هو موديول جاهز في بايثون يتيح لك إعادة تسمية الملفات، مسح الملفات، إنشاء مجلدات، مسح مجلدات، التنقل بين المجلدات..إلخ.

لاستخدام هذا الموديول يجب تضمينه كالتالي.

```
import os
```

- **دوال الموديول os الأكثر استخداماً**

١. os.rename (current_file_name, new_file_name): تستخدم لتغيير اسم الملف.

- مكان البارامتر current_file_name نمرر اسم الملف الذي نريد تغيير اسمه.
- مكان البارامتر new_file_name نمرر الاسم الجديد الذي نريد وضعه للملف.

٢. os.remove(file_name): تستخدم لمسح الملف.

- مكان البارامتر file_name نمرر اسم الملف الذي نريد حذفه.

٣. os.path.exists(file_name): تستخدم لمعرفة ما إن كان الملف موجوداً أم لا.

- مكان البارامتر file_name نمرر اسم الملف الذي نريد التأكد ما إن كان موجوداً أم لا.
- ترجع True إذا كان الملف موجوداً وترجع False إن لم يكن كذلك.

٤. os.mkdir(directory_name): تستخدم لإنشاء مجلد جديد.

- مكان البارامتر directory_name نمرر اسم المجلد الذي نريد إنشاؤه.

- **معلومة**: اسم الدالة هو اختصار لجملة Make Directory.

٥. os.rmdir(directory_name): تستخدم لمسح المجلد.

- مكان الباراميتر directory_name نمرر اسم المجلد الذي نريد حذفه.

ملاحظة: يمكنك حذف المجلد في حال كان فارغاً فقط، أي في حال لم يكن يحتوي على أي ملف بداخله.

- **معلومة:** اسم الدالة هو اختصار لجملة Remove Directory.

٦. os.getcwd(): تستخدم لمعرفة اسم المجلد الذي تقف بداخله حالياً.

- **معلومة:** اسم الدالة هو اختصار لجملة Get Current Working Directory.
- **مثال**

```
# هنا قمنا بتضمين الموديول os حتى نستطيع استخدام الدوال الموجودة فيه  import os

# هنا قمنا باستدعاء الدالة remove() لمسح الملف 'demo.txt'

os.remove('demo.txt')
```

مثال:

```
import os

# التأكد من وجود الملف قبل محاولة حذفه

if os.path.exists('example.txt'):
    os.remove('example.txt')
    print("File deleted successfully.")
else:
    print("The file does not exist.")
```

❖ التعامل مع الملفات المضغوطة و التعامل مع JSON , CSV

- **التعامل مع الملفات المضغوطة zip**

دالة zip

تعيد الدالة zip() مكرّرًا يُركب عناصر كل من الكائنات القابلة للتكرار المعطاة.

القيمة المعادة

مُكرر يحتوي على صفوف تكون عناصرها ذات نفس الفهرس (index) في كلّ عنصر من عناصر الكائنات القابلة للتّكرار. إذ يحتوي مثلا أول صف على جميع العناصر التي تكون أول العناصر في الكائنات القابلة للتّكرار المُعطاة (انظر الأمثلة أدناه).

يتوقف المُكرر عندما تنتهي عناصر أصغر كائن قابل للتكرار.

عند استدعائها بكائن واحد قابل للتّكرار فقط، فستُعيد الدالة مُكررًا يحتوي على صفوف من عنصر واحد فقط.

إن لم تمرر للدّالة أية معاملات، فسيعاد مكرّر فارغ.

المثال التالي يوضح كيفية عمل هذه الدالة، نستعمل الدالة list() لعرض ما بداخل المكرّر الناتج:

```
>>> list(zip([1, 2, 3], 'abc'))
[(1, 'a'), (2, 'b'), (3, 'c')]
>>> list(zip([1, 2, 3], 'abcdfe')) # تُتجاهل القيم الزّائدة
[(1, 'a'), (2, 'b'), (3, 'c')]
>>> list(zip([1, 2, 3], (1, 2, 3))) # تركيب قائمة وصفّ
[(1, 1), (2, 2), (3, 3)]
>>> list(zip([1, 2, 3], range(1, 4))) # تركيب قائمة وكائن مجال
[(1, 1), (2, 2), (3, 3)]
```

يُمكن مثلا استخدام الدّالة zip() مع الدّالة dict() لإنشاء قاموس من كائنين قابلين للتّكرار تكون مفاتيحه من الكائن الأول وقيمها من الكائن الثّاني:

```
>>> dict(zip("abc", range(3)))
{'a': 0, 'b': 1, 'c': 2}
```

ملاحظة : لا تستخدم الدّالة zip() إلّا إن كانت المُدخلات ذات طول مُتساوٍ أو إن لم تكن القيم المتبقيّة في الكائن القابل للتّكرار الأطول مُهمّة.

- **فك الضغط**

يمكن استعمال الدالة zip() مع العامل * لفك التركيب:

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> zipped = zip(x, y)
>>> list(zipped)
[(1, 4), (2, 5), (3, 6)]
>>> x2, y2 = zip(*zip(x, y))
>>> x == list(x2) and y == list(y2)
True
```

يعد فك ضغط الملفات في Python عملية مباشرة باستخدام zip.

- **قراءة و كتابة ملفات csv**

ملفات CSV قيم مفصولة بفواصل (Values Separated Comma)

هي أكثر تنسيقات الملفات استخدامًا لإستيراد وتصدير مجموعات البيانات الكبيرة. السبب في تفضيل ملف CSVعلى ملف Excel هو أن ملف CSV يستهلك ذاكرة أقل مقارنة بملف Excel لذلك أثناء تعلم علم البيانات، يجب أن تعرف كيفية قراءة ملفات CSV وكتابتها. إذا كنت تريد معرفة كيفية قراءة ملفات CSV وكتابتها، فهذه المقالة مناسبة لك. في هذه المقالة، سوف آخذك من خالل برنامج تعليمي حول قراءة وكتابة ملفات CSVباستخدام بايثون.

كتابة ملف .CSV سأقوم هنا أولًا إنشاء نموذج بيانات باستخدام قاموس بايثون حول اسم وعمر الطالب، وبعد ذلك تخزين قاموس بايثون هذا في ملفCSV :

```
#writing a csv file

 import csv

 import pandas as pd

 data = {"Name": ["Aman", "Diksha", "Akanksha", "Sajid", "Akshit ,]" " Age}]22 ,23 ,25 ,21 ,23[ :"

data = pd.DataFrame(data)

data.to_csv("age_data.csv", index=False) print(data.head())
```

```
 #reading a csv file
import pandas as pd
data = pd.read_csv("age_data.csv")
print(data.head())
```

هذه هي الطريقة التي يمكنك بها كتابة ملف CSV باستخدام بايثون. آلن فيما يلي كيفية قراءة ملف CSV هذا باستخدام بايثون:

```
#reading a csv file

 import pandas as pd

 data = pd.read_csv("age_data.csv")

print(data.head())
```

```
       Name  Age
0      Aman   23
1    Diksha   21
2  Akanksha   25
3     Sajid   23
4    Akshit   22
```

- **التعامل مع JSON**

استوحت هذه الصيغة من الصيغةالغرضية والمصفوفية في لغة جافا سكربت، إلا أن قواعد كتابة بايثون فيما يتعلق بالقواميس والقوائم أثرت على قواعد JSON

باعتبارأنهاوجدت قبل جافا سكربت، لذلك تعتبر هذه الصيغة خليط من قوائم وقواميس بايثون،

```
{ "name" : "Chuck",
 "phone" : {
"type" : "intl",
 "number" : "+1 734 303 4456"
 },
"email" : {
 "hide" : "yes"
 } }
```

- **تحليل نصوص JSON**

ننشئ ملفات JSON بترتيب القواميس والقوائم داخل بعضهم البعض كما نحتاج، وفي هذا المثال نمثل قائمة مستخدمين بحيث يكون كل مستخدم عبارة عن مجموعة من أزواج مفتاح-قيمة)أي قاموس(أي لدينا قائمة من القواميس، كما سنستخدم مكتبة جاهزة لتحليل نصJSON وقراءة البيانات

```
import json

 data =' ' '

[ { "id" : "001",

 "x" : "2",

"name" : "Chuck"

} ,

{ "id" : "009",

 "x" : "7",

 "name" : "Brent"

 } ] ' ' ' info = json.loads(data)

 print('User count:', len(info))

 for item in info:

print('Name', item['name'])

print('Id', item['id'])

 print('Attribute', item['x'])
```

```
User count: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

امثله:

قراءة من ملف

```
# قراءة محتويات ملف نصي
with open('example.txt', 'r') as file:
  content = file.read()
  print(content(

# قراءة الملف سطر بسطر
with open('example.txt', 'r') as file:
  for line in file:
    print(line.strip(()

# قراءة جميع الأسطر في قائمة
with open('example.txt', 'r') as file:
  lines = file.readlines()
  print(lines(
```

تعديل محتويات ملف

```
# كتابة نص إلى ملف (سيقوم بمسح محتويات الملف إذا كان موجوداً)
with open('example.txt', 'w') as file:
    file.write("This is a new line.\n("

# إضافة نص إلى ملف (سيضيف النص إلى نهاية الملف)
with open('example.txt', 'a') as file:
    file.write("Appending another line.\n("
```

حذف ملف

```
import os

# حذف ملف
if os.path.exists('example.txt:('
    os.remove('example.txt('
    print("File deleted successfully(".
else:
    print("The file does not exist".
```

الفصل الخامس

البرمجة الكائنية OOP

في هذا الفصل سنتعرف على المواضيع التالية:

- مقدمة في البرمجة الكائنية
- التعرف على الكائنات و الفئات understanding objects and classes
- الخصائص و الأساليب في الكائنات attributes and Methods in objects
- التوريث و التعددية في البرمجة الكائنية inheritance and polymorphism in oop
- امثلة شاملة

❖ مقدمة في البرمجة الكائنية

قبل أن أبتدأ في مقدمة البرمجة الكائنية نحتاج أن نضع قاعدة مهمة لكي نستطيع أن نفهم ماهو مفهوم البرمجة الكائنية

(Object-Oriented Programming)

كل شيء عبارة عن كائن Object an is things Every

لقد وضعنا هذا الفتراض لن كل شي تراه بالعي الجردة والغير مجردة عبارة عن كائن

والبرمجة الكائنية هي ببساطة طريقة أخرى للبرمجة بحيث تكون أجزاء الشّيفرة مجموعة داخل دوال تُسمّى التوابع methods والدوال تكون داخل صنف معيّن Class عند إنشاء كائن object من هذا الصنف فإنّنا نستطيع أن نُنفّذ عليه مُختلف العمليات الموجودة داخل التوابع والتي بدورها توجد داخل الصنف.

البرمجة الكائنيّة تُستعمل أساسا إذا كان البرنامج الذي تبنيه مُعقّدا مع العديد من الوظائف المُتعلّقة ببعضها (كمكتبة برمجيّة)، مثلا لنقل بأنّك تُطوّر برنامجا مسؤولا عن جلب بيانات من موقع مُعيّن، وبعدها التّعديل عليها، ثمّ إخراج مستند PDF يحتوي على هذه البيانات بشكل يُسهّل قراءتها، هنا ستحتاج إلى صنف لجلب البيانات والتّعديل عليها، وصنف أخرى لتحويل البيانات إلى نصّ مقروء واضح ثمّ إلى ملفّ PDF.

إذا نشرت برنامجك مع صديقك وأراد أن يعمل على الجزء الثاني لإضافة وظيفة تُمكن المُستخدم من طباعة المُستند فلا يُعقل أن يضطر للمرور على كل ما يتعلّق بجلب البيانات فقط لأنّه يريد أن يضيف خاصيّة لا علاقة لها بجلب البيانات، استعمال البرمجة الكائنيّة في هذا المشروع سيسمح لك بالتّركيز على الجزء الأول، وسيسمح لصديقك بالتّركيز على تطوير الجزء الثاني.

خلاصة الأمر هي أنّك لست مضطرا لاستعمال البرمجة الكائنيّة إلا إذا كان برنامجك طويلا يحتوي على وظائف تتعلّق ببعضها البعض (وظائف من نفس الصنف)، ونسبة استخدام الآخرين لشيفرتك عالية.

❖ التعرف على الكائنات و الفئات understanding objects and classes

- **مفهوم CLASS**

الصنف ببساطة يحتوي على أجزاء مُختلفة من الشيفرة تماما مثل الدالة، الفرق هنا هو أنّ الصنف يحتوي على دوال كذلك، وهذه الدوال تُسمى التّوابع، ويحتوي كذلك على مُتغيّرات وتنقسم هذه الأخيرة إلى نوعين، مُتغيّر الصنف، والمُتغيّر العادي، الفرق بينهما هو أنّك تستطيع الوصول إلى مُتغيّر الصنف في أي مكان داخل الصنف (سواء داخل التوابع أو خارجها).

- **إنشاء Class**

لإنشاء صنف في لغة بايثون كلّ ما عليك فعله هو كتابة كلمة class وبعدها اسم الصنف ثمّ إلحاق نقطتين، بعدها اكتب الشيفرة بإزاحة أربع مسافات:

```
>>> class My_class:
...     pass
```

أنشأنا أعلاه صنفًا بسيطا باسم My_class وهو لا يفعل أي شيء يُذكر كلمة pass تُخبر بايثون بالمرور دون تنفيذ أي شيء.

إذا كتبت اسم الصنف على مفسّر بايثون فستجد مُخرجا كالتّالي:

```
>>> My_class
<class __main__.My_class at 0x7fd0efc41460>
```

لاحظ بأنّ الكلمة الأولى من المُخرج هي class أي أنّنا أصبحنا نمتلك صنفًا جديداً، ما يتبع at هو المكان في الذاكرة الذي وُضع فيه الصنف ويتغيّر بين الحين والآخر.

- **استعمال Class**

بعد أن أنشأنا الصنف سنتمكّن الآن من إنشاء كائن من هذا الصنف، والكائن مُجرّد اسم تماما كالمُتغيّر:

```
my_object = My_class()
```

الآن الكائن my_object هو من صنف My_class .

- **تعريف المتغيرات داخل صنف**

يُمكننا أن نُعرّف مُتغيّرات في الصنف تماما كما نُعرّف المُتغيّرات بشكل عادي.

```
class My_class:
            my_variable = 'This is my variable'
```

للوصول إلى المُتغيّر ننشئ أولا كائنا من الصنف وبعدها نكتب اسم الكائن، ثمّ نقطة ثمّ اسم المُتغيّر:

```
my_object = My_class()
print my_object.my_variable
```

المُخرج:

```
This is my variable
```

يُمكن كذلك الحصول على النّتيجة ذاتها في سطر واحد:

```
print My_class().my_variable
```

مثال:

```
class Car:

    def __init__(self, make, model, year:(

        self.make = make

        self.model = model

        self.year = year



    def display_info(self:(

        print(f"Car: {self.year} {self.make} {self.model}")
```

- **إنشاء التوابع**

التوابع هي دوال خاصّة بالصنف، ويُمكننا إنشاء التابع بنفس الطّريقة التي نُنشئ بها الدالة، الاختلاف هنا هو أنّ جميع التوابع يجب أن تُعرّف مع مُعامل باسم self وذلك للإشارة إلى أنّ الدالة/التابع تابع للصنف، لننشئ تابعا داخل صنف الآن.

```
class My_class:
    my_variable = 'This is my variable'
    def my_method(self):
        print 'This is my method'
```

الآن إذا أنشأنا كائن فإنّنا سنتمكّن من الوصول إلى التابع، وتذكّر بأنّ التابع تلحقه الأقواس:

```
my_object = My_class()
my_object.my_method()
```

المُخرج:

```
This is my method
```

يُمكن كذلك الحصول على النّتيجة ذاتها في سطر واحد:

```
My_class().my_method()
```

كما تُلاحظ فقد نُفّذت الشفرة الموجودة داخل التّابع **my_method** ويُمكننا كذلك أن نجعل التّابع يقبل المُعاملات، لكن تذكّر الحفاظ على الكلمة self كمُتغيّر أول.

```
class My_class:
    my_variable = 'This is my variable'
    def my_method(self, my_parameter):
        print 'This is my method ; {} is my parame-
ter'.format(my_parameter)
```

يُمكنك استدعاء التّابع كالتّالي:

```
my_object = My_class()
my_object.my_method('Parameter1')
my_object.my_method('Parameter2')
```

المُخرج:

```
This is my method ; Parameter1 is my parameter
This is my method ; Parameter2 is my parameter
```

في البرنامج السّابق، أنشأنا أولا صنفًا باسم My_class وقُمنا بتعريف مُتغيّر، ثمّ بتعريف تابع باسم my_method يقبل مُعاملين self وmy_parameter ، بالنّسبة لاستدعاء التّابع، فنحتاج فقط إلى تمرير المُعاملات الموجودة بعد المُعامل self ولا نحتاج إلى تعيين قيمة لهذا المُعامل.

- **مُلاحظة**: يُمكنك إعادة تسميّة المُعامل الأول كما تشاء، أي أنّ البرنامج التّالي سيعمل دون مشاكل.

```
class My_class:
    def my_method(this, my_parameter):
        print '{} is my parameter'.format(my_parameter)
```

ولكن رغم ذلك فالمُتعارف عليه بين مُبرمجي لغة بايثون هو استعمال self ، وفي كثير من اللغات الأخرى تُستعمل this عوضا عن self ، أما في برامِجك فمن المُفضّل الإبقاء على هذه التّسميّة المُتعارف عنها، وذلك لتكون شفراته سهلة القراءة.

- **الوصول إلى متغيرات الصنف داخل التوابع**

تأمّل الصنف التّالي:

```
class Person:
    lastname = 'Dyouri'
    job = 'Writer, Developer'
    def say_hello(self):
        name = 'Abdelhadi'
        print 'Hello, My name is {}'.format(name)
```

البرنامج أعلاه بسيط جدا، أولا نعرّف صنف باسم Person وبعدها نقوم بتعيين قيمتين للمُتغيّرين name و lastname ، وبعدها عرّفنا تابعا باسم say_hello يطبع جملة Hello, My name is Abdelhadi.

كلّ شيء جيد، لكن ماذا لو أردنا أن نصل إلى المُتغيّرات الأخرى الموجودة خارج التّابع، فلا يُمكننا مثلا أن نقوم بالأمر كالتّالي:

```
class Person:
    lastname = 'Dyouri'
    job = 'Writer, Developer'
    def say_hello(self):
        name = 'Abdelhadi'
        print 'Hello, My name is {}'.format(name)
        print lastname
        print job
```

ستحصل على الخطأ التّالي:

```
global name 'lastname' is not defined
```

لتفادي هذا الخطأ سنستعمل كلمة self قبل المُتغيّر.

```
class Person:
    lastname = 'Dyouri'
    job = 'Writer, Developer'
    def say_hello(self):
        name = 'Abdelhadi'
        print 'Hello, My name is {}'.format(name)
        print 'My Last name is {} '.format(self.lastname)
        print 'I am a {}'.format(self.job)
```

استدعاء التّابع:

```
me = Person()

me.say_hello()
```

المُخرج:

```
Hello, My name is Abdelhadi

My Last name is Dyouri

I am a Writer, Developer
```

لاحظ بأنّنا قُمنا بالوصول إلى مُتغيّر lastname عن طريق استدعائه بـ self.lastname وكذا الحال مع المُتغيّر job ، وهذه الطّريقة مُشابهة لاستخدام كلمة global الفرق هنا أنّ هذه الأخيرة تُمكن من الوصول إلى المُتغيّر في كامل البرنامج، أمّا كلمة self فتُشير إلى المُتغيّر المُعرّف في الصنف الحاليّة فقط.

لتفهم أكثر كيفيّة عمل الكلمة self فقط تخيّل بأنّها تحمل نفس اسم الصنف، مثلا:

```
class Person:

    lastname = 'Dyouri'

    job = 'Writer, Developer'

    def say_hello(self):

        name = 'Abdelhadi'

        print 'Hello, My name is {}'.format(name)

        print 'My Last name is {} '.format(Abd.lastname)

        print 'I am a {}'.format(Abd.job)
```

لاحظ بأنّنا غيّرنا كلمة self إلى اسم الصنف واستمرّ عمل البرنامج دون مشاكل.

وبنفس الطّريقة يُمكنك أن تستدعي تابعا داخل تابع آخر في نفس الصنف:

```
class Person:
    def say_name(self):
        print 'Abdelhadi'
    def say_hello(self):
        print 'Hello My name is:'
        self.say_name()
```

المُخرج:

```
Hello My name is:
Abdelhadi
```

ما حدث هو أنّ التّابع say_hello قام بطباعة جملة: Hello My name is ثمّ قام باستدعاء التّابع say_name الذي قام بدوره بطباعة الاسم Abdelhadi.

- **الخصائص و الأساليب في الكائنات attributes and Methods in objects**

لمذا إفترضنا في المقدمة أن كل شي عبارة عن كائن ؟

إفترضنا هذا الافتراض لأن كل كائن يتكون من

1- خصائص (Properties, Attributes)

2-أفعال (Action, Methods, Behaviour)

ولأن بما أننا إعتبرنا أن كل شيء عبارة عن كائن فذلك يدل على أن كل شي له خصائص و أفعال

خصائص الكائن : (Attributes ,Properties) هي كل شي بالكائن ولا تفارقه أبدا (هي مواصفات الكائن) وأقرب مثال على ذلك {الانسان} لنرى ماهي خصائص الانسان:

خصائص الإنسان	
إسم الخاصية	قيمة الخاصية
الإسم	محمد
العمر	٢٥ سنة
الطول	١٦٠ سنتيمتر
الوزن	٧٠ كيلو

لقد ذكرنا بالسابق بعض الخصائص الخاصة بالانسان (هل هذه الخصائص تفارق النسان ؟) .

فهذا يدل على أن كل شي تضع عليه عينيك يعتبر كائن

تعريف توضيحي أخير : كل شي يأتي على صيغة (Value = Name) يعتبر خاصية من خصائص الكائن

أفعال الكائن:

(Action, Methods, Behaviour)

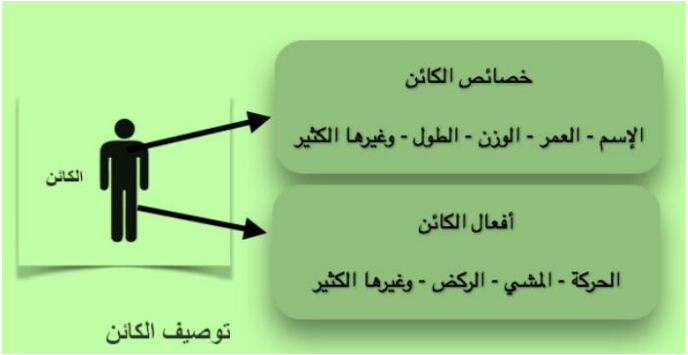
هي كل ما يستطيع القيام به الكائن

بما إننا إفترضنا أن الانسان كائن وله خصائص فلابد من أن له أفعال وأفعاله كالتالي:

- الحركة
- الشي
- الركض

ويوجد غيرها الكثير من الافعال

في هذا الرسم نختصر كل الكلام المكتوب أعلاه



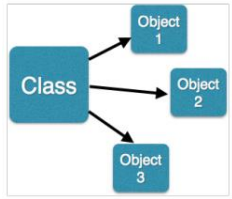
لو نلاحظ أننا إلى الان لم نتحدث عن أى شى له علاقة بالبرمجة فسوف يكون سؤالك ماهى العلاقة بهذا الكلام فى البرمجة

تمثيل مفهوم برمجة الكائنات في البرمجة : لكي نقوم بربط مفهوم الكائنات بالبرمجة نحتاج أن نفهم ماهو الـ(Class) و(Object)

الـ (Class) هو عبارة عن كائن .

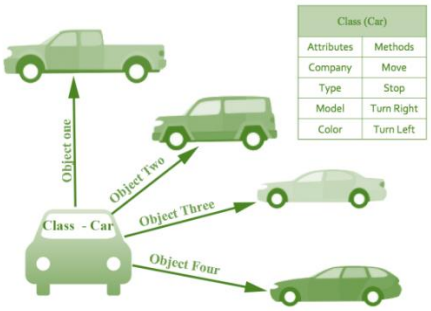
الـ (Object)هو عبارة عن كائن .

كلها نفس المعنى ماهو الفرق بينهم !



من الصورة أعلاه نفهم أن الـ(Object)عبارة عن نسخه من الـ(Class)

مثال توضيحي:



كيف تتعامل لغات البرمجة مع الـ(Classes)و الـ(Objects)؟

لكي تتعامل لغات البرمجة مع الـ(Classes)و الـ(Objects)نحتاج لتمثيل الخصائص والافعال كالتالي:

- الخصائص (Properties, Attributes) المتغيرات (Variables)
- الأفعال (Action, Methods, Behaviour) الدوال (Function)

والان أصبح الـ(Class)و الـ (Object)عبارة عن مجموعة من المتغيرات والدوال

تنبيه: لا بد من إنشاء الـ(Class)لكى نتمكن من إنشاء الـ(Object)

- **التوريث و التعددية في البرمجة الكائنية inheritance and polymorphism in oop**

تُسهّل البرمجة الكائنية كتابة شيفرات قابلة لإعادة الاستخدام وتجنب التكرار في مشاريع التطوير، إحدى الآليات التي تحقق بها البرمجة الكائنية هذا الهدف هي مفهوم الوراثة (inheritance)، التي بفضلها يمكن لصنف فرعي (subclass) استخدام الشيفرة الخاصة بصنف أساسي base class ، ويطلق عليه «صنف أب» أيضاً موجود مسبقاً.

سيستعرض هذا الدرس بعض الجوانب الرئيسية لمفهوم الوراثة في بايثون، بما في ذلك كيفية إنشاء الأصناف الأساسية (parent classes) والأصناف الفرعية (child classes) ، وكيفية إعادة تعريف (override) التوابع والخاصيات، وكيفية استخدام التابع ()super، وكيفية الاستفادة من الوراثة المتعددة (multiple inheritance)

- **ما هي الوراثة؟**

التوريث هو مفهوم يسمح بإنشاء كلاس جديد (فرعي) يعتمد على كلاس موجود (أساسي). الكلاس الفرعي يرث جميع الخصائص والدوال من الكلاس الأساسي، ويمكنه إضافة خصائص ودوال جديدة أو تعديل القائمة.

تقوم الوراثة على استخدام شيفرة صنف معين في صنف آخر أي يرث صنف يراد إنشاؤه شيفرة صنف آخر، يمكن تمثيل مفهوم الوراثة في البرمجة بالوراثة في علم الأحياء تماماً، فالأبناء يرثون خاصيات معينة من آبائهم، ويمكن لطفل أن يرث طول والده أو لون عينيه بالإضافة إلى خاصيات أخرى جديدة خاصة فيه، كما يتشارك الأطفال نفس اسم العائلة الخاصة بآبائهم.

ترث الأصناف الفرعية (subclasses)، تُسمى أيضاً الأصناف الأبناء [child classes] التوابع والمتغيرات من الأصناف الأساسية (base classes) ، تسمى أيضًا الأصناف الآباء [parent classes]

مثلًا، قد يكون لدينا صنف أساسي يسمى Parent يحتوي متغيرات الأصناف last_name و height و eye_color، والتي سيرثها الصنف الابن Child.

لمَّا كان الصنف الفرعي Child يرث الصنف الأساسي Parent، فبإمكانه إعادة استخدام شيفرة Parent، مما يسمح للمبرمج بكتابة شيفرة أوجز، وتقليل التكرار.

- **الأصناف الأساسية**

تشكل الأصناف الأساسية أساسًا يمكن أن تستند إليه الأصناف الفرعية المُتفرّعة منها، إذ تسمح الأصناف الأساسية بإنشاء أصناف فرعية عبر الوراثة دون الحاجة إلى كتابة نفس الشيفرة في كل مرة، يمكن تحويل أي صنف إلى صنف أساسي، إذ يمكن استخدامه لوحده، أو جعله قالباً (نموذجاً).

لنفترض أنّ لدينا صنفاً أساسياً باسم Bank_account ، وصنفين فرعيين مُشتقين منه باسم Personal_account و Business_account ستكون العديد من التوابع مشتركة بين الحسابات الشخصية (Personalaccount) والحسابات التجارية (Businessaccount) ، مثل توابع سحب وإيداع الأموال، لذا يمكن أن تنتمي تلك التوابع إلى الصنف الأساسي Bank_account سيكون للصنف Business_account توابع خاصة به، مثل تابع مخصص لعملية جمع سجلات ونماذج الأعمال، بالإضافة إلى متغير employee_identification_number موروث من الصنف الأب.

وبالمثل، قد يحتوي الصنف Animal على التابعين ()eating و ()sleeping، وقد يتضمن الصنف الفرعي Snake تابعين إضافيين باسم ()hissing و ()slithering خاصين به.

دعنا ننشئ صنفًا أساسياً باسم Fish لاستخدامه لاحقاً أساساً لأصناف فرعية تمثل أنواع الأسماك، سيكون لكل واحدة من تلك الأسماك أسماء أولى وأخيرة، بالإضافة إلى خصائص مميزة خاصة بها.

سننشئ ملفاً جديداً يسمى fish.py ونبدأ بالباني، والذي سنعرّف داخله متغيري الصنف first_name و last_name لكل كائنات الصنف Fish، أو أصنافه الفرعية.

```
class Fish:
    def __init__(self, first_name, last_name="Fish"):
        self.first_name = first_name
        self.last_name = last_name
```

القيمة الافتراضية للمتغير last_name هي السلسلة النصية "Fish"، لأننا نعلم أنّ معظم الأسماك سيكون هذا هو اسمها الأخير.

لنُضف بعض التوابع الأخرى:

```
class Fish:
    def __init__(self, first_name, last_name="Fish"):
        self.first_name = first_name
        self.last_name = last_name

    def swim(self):
        print("The fish is swimming.")

    def swim_backwards(self):
        print("The fish can swim backwards.")
```

لقد أضفنا التابعين swim() و swim_backwards() إلى الصنف Fish حتى يتسنى لكل الأصناف الفرعية استخدام هذه التوابع.

ما دام أنّ معظم الأسماك التي ننوي إنشاءها ستكون عظمية (أي أنّ لها هيكلا عظمياً) وليس غضروفية (أي أن لها هيكلًا غضروفياً)، فيمكننا إضافة بعض الخاصيات الإضافية إلى التابع __init__() :

```
class Fish:
    def __init__(self, first_name, last_name="Fish",
                 skeleton="bone", eyelids=False):
        self.first_name = first_name
        self.last_name = last_name
        self.skeleton = skeleton
        self.eyelids = eyelids

    def swim(self):
        print("The fish is swimming").

    def swim_backwards(self):
        print("The fish can swim backwards").
```

لا يختلف بناء الأصناف الأساسية عن بناء أي صنف آخر، إلا أننا نصممها لتستفيد منها الأصناف الفرعية المُعرّفة لاحقًا.

- **الأصناف الفرعية**

الأصناف الفرعية هي أصناف ترث كل شيء من الصنف الأساسي، هذا يعني أنّ الأصناف الفرعية قادرة على الاستفادة من توابع ومتغيرات الصنف الأساسي.

على سبيل المثال، سيتمكن الصنف الفرعي Goldfish المشتق من الصنف Fish من استخدام التابع ()swim المُعرّف في Fish دون الحاجة إلى التصريح عنه.

يمكننا النظر إلى الأصناف الفرعية على أنها أقسام من الصنف الأساسي، فإذا كان لدينا صنف فرعي يسمى Rhombus (معيّن)، وصنف أساسي يسمى Parallelogram (متوازي الأضلاع) ،يمكننا القول أنّ المعين (Rhombus) هو متوازي أضلاع (Parallelogram)

يبدو السطر الأول من الصنف الفرعي مختلفاً قليلاً عن الأصناف غير الفرعية، **إذ يجب عليك تمرير الصنف الأساسي إلى الصنف الفرعي كمعامل:**

```
class Trout(Fish):
```

الصنف Troutهو صنف فرعي من Fish يدلنا على هذا الكلمةُ Fish المُدرجة بين قوسين.

يمكننا إضافة توابع جديدة إلى الأصناف الفرعية، أو إعادة تعريف التوابع الخاصة بالصنف الأساسي، أو يمكننا ببساطة قبول التوابع الأساسية الافتراضية باستخدام الكلمة المفتاحية pass، **وهو ما سنفعله في المثال التالي.**

- المثال

```
class Trout(Fish):
    pass
```

يمكننا الآن إنشاء كائن من الصنف Trout دون الحاجة إلى تعريف أي توابع إضافية.

```
class Trout(Fish):
    pass
terry = Trout("Terry")
print(terry.first_name + " " + terry.last_name)
print(terry.skeleton)
print(terry.eyelids)
terry.swim()
terry.swim_backwards()
```

لقد أنشأنا كائنًا باسم terry من الصنف Trout، والذي سيستخدم جميع توابع الصنف Fish وإن لم نعرّفها في الصنف الفرعي Trout يكفي أن نمرر القيمة "Terry" إلى المتغير first_name، أما المتغيرات الأخرى فقد جرى تهيئتها سلفاً.

عند تنفيذ البرنامج، سنحصل على المخرجات التالية:

```
Terry Fish
bone
False
The fish is swimming.
The fish can swim backwards.
```

لننشئ الآن صنفاً فرعياً آخر يعرّف تابعاً خاصا به، سنسمي هذا الصنف Clownfish سيسمح التابع الخاص به بالتعايش مع شقائق النعمان البحري:

```
class Clownfish(Fish):
    def live_with_anemone(self):
        print("The clownfish is coexisting with sea anemone").
```

دعنا ننشئ الآن كائناً آخر من الصنف Clownfish

```
casey = Clownfish("Casey")

print(casey.first_name + " " + casey.last_name)

casey.swim()

casey.live_with_anemone()
```

عند تنفيذ البرنامج، سنحصل على المخرجات التالية:

```
Casey Fish

The fish is swimming.

The clownfish is coexisting with sea anemone.
```

تُظهر المخرجات أنّ الكائن casey المستنسخ من الصنف Clownfish قادر على استخدام التابعين __init__() و swim() الخاصين بالصنف Fish، إضافة إلى التابع live_with_anemone() الخاص بالصنف الفرعي.

إذا حاولنا استخدام التابع live_with_anemone() في الكائن Trout، فسوف يُطلق خطأ:

```
terry.live_with_anemone()

AttributeError: 'Trout' object has no attribute 'live_with_anemone'
```

ذلك أنّ التابع live_with_anemone() ينتمي إلى الصنف الفرعي Clownfish فقط، وليس إلى الصنف الأساسي Fish .

ترث الأصناف الفرعية توابع الصنف الأساسي الذي اشتُقّت منه، لذا يمكن لكل الأصناف الفرعية استخدام تلك التوابع.

- **إعادة تعريف توابع الصنف الأساسي**

في المثال السابق عرّفنا الصنف الفرعي Trout الذي استخدم الكلمة المفتاحية pass ليرث جميع سلوكيات الصنف الأساسي Fish، وعرّفنا كذلك صنفاً آخر Clownfish يرث جميع سلوكيات الصنف الأساسي، ويُنشئ أيضاً تابعاً خاصاً به، قد نرغب في بعض الأحيان في استخدام بعض سلوكيات الصنف الأساسي، ولكن ليس كلها يطلَق على عملية تغيير توابع الصنف الأساسي «إعادة التعريف » (Overriding)

عند إنشاء الأصناف الأساسية أو الفرعية، فلا بد أن تكون لك رؤية عامة لتصميم البرنامج حتى لا تعيد تعريف التوابع إلا عند الضرورة.

سننشئ صنفًا فرعيًا Shark مشتقاً من الصنف الأساسي Fish، الذي سيمثل الأسماك العظمية بشكل أساسي، لذا يتعين علينا إجراء تعديلات على الصنف Shark المخصص في الأصل للأسماك الغضروفية، من منظور تصميم البرامج، إذا كانت لدينا أكثر من سمكة غير عظمية واحدة، فيُستحب أن ننشئ صنفاً خاصاً بكل نوع من هذين النوعين من الأسماك.

تمتلك أسماك القرش، على عكس الأسماك العظمية، هياكل مصنوعة من الغضاريف بدلاً من العظام، كما أنّ لديها جفوناً، ولا تستطيع السباحة إلى الوراء، كما أنها قادرة على المناورة للخلف عن طريق الغوص.

على ضوء هذه المعلومات، سنعيد تعريف الباني ()__init__ والتابع ()swim_backwards لا نحتاج إلى تعديل التابع ()swim لأن أسماك القرش يمكنها السباحة

دعنا نلقي نظرة على هذا الصنف الفرعي:

```
class Shark(Fish):
    def __init__(self, first_name, last_name="Shark",
                 skeleton="cartilage", eyelids=True):
        self.first_name = first_name
        self.last_name = last_name
        self.skeleton = skeleton
        self.eyelids = eyelids

    def swim_backwards(self):
        print("The shark cannot swim backwards, but can sink back-
wards").
```

لقد أعدنا تعريف المعاملات التي تمت تهيئتها في التابع __init__()، فأخذ المتغير last_name القيمة "Shark"، كما أسنِد إلى المتغير skeleton القيمة "cartilage"، فيما أسنِدَت القيمة المنطقية True الى المتغير eyelids يمكن لجميع نُسخ الصنف إعادة تعريف هذه المعاملات.

يطبع التابع swim_backwards() سلسلة نصية مختلفة عن تلك التي يطبعها في الصنف الأساسي Fish، لأنّ أسماك القرش غير قادرة على السباحة للخلف كما تفعل الأسماك العظمية.

يمكننا الآن إنشاء نسخة من الصنف الفرعي Shark، والذي سيستخدم التابع swim() الخاص بالصنف الأساسي Fish:

```
sammy = Shark("Sammy")
print(sammy.first_name + " " + sammy.last_name)
sammy.swim()
sammy.swim_backwards()
print(sammy.eyelids)
print(sammy.skeleton)
```

عند تنفيذ هذه الشفرة، سنحصل على المخرجات التالية:

```
Sammy Shark
The fish is swimming.
The shark cannot swim backwards, but can sink backwards.
True
cartilage
```

لقد أعاد الصنف الفرعي Shark تعريف التابعين __init__() و swim_backwards() الخاصين بالصنف الأساسي Fish، وورث في نفس الوقت التابع swim() الخاص بالصنف الأساسي.

- **الدالة super()**

يمكنك باستخدام الدالة super() الوصول إلى التوابع الموروثة التي أعيدت كتابتها.

عندما نستخدم الدالة super()، فإننا نستدعي التابع الخاص بالصنف الأساسي لاستخدامه في الصنف الفرعي، على سبيل المثال، قد نرغب في إعادة تعريف جانب من التابع الأساسي وإضافة وظائف معينة إليه، ثم بعد ذلك نستدعي التابع الأساسي لإنهاء بقية العمل.

في برنامج خاص بتقييم الطلاب مثلاً، قد نرغب في تعريف صنف فرعي Weighted_grade يرث الصنف الأساسي Grade، ونعيد فيه تعريف التابع calculate_grade() الخاص بالصنف الأساسي من أجل تضمين شيفرة خاصة بحساب التقدير المرجّح (weighted grade) ، مع الحفاظ على بقية وظائف الصنف الأساسي، عبر استدعاء التابع super()، سنكون قادرين على تحقيق ذلك.

عادة ما يُستخدم التابع super() ضمن التابع __init__()، لأنّه المكان الذي ستحتاج فيه على الأرجح إلى إضافة بعض الوظائف الخاصة إلى الصنف الفرعي قبل إكمال التهيئة من الصنف الأساسي.

لنضرب مثلاً لتوضيح ذلك، دعنا نعدّل الصنف الفرعي Trout نظرًا لأنّ سمك السلمون المرقّط من أسماك المياه العذبة، فلنضف متغيراً اسمه water إلى التابع __init__() ، ولنعطيه القيمة "freshwater"، **ولكن مع الحفاظ على باقي متغيرات ومعاملات الصنف الأساسي:**

```
class Trout(Fish):
    def __init__(self, water = "freshwater"):
        self.water = water
        super().__init__(self)
```

لقد أعدنا تعريف التابع __init__() في الصنف الفرعي Trout، وغيرنا سلوكه موازنةً بالتابع __init__() المُعرَّف سلفاً في الصنف الأساسي Fish لاحظ أننا استدعينا التابع __init__() الخاص بالصنف Fish بشكل صريح ضمن التابع __init__() الخاص بالصنف Trout.

بعد إعادة تعريف التابع، لم نعد بحاجة إلى تمرير first_name كمعامل إلى Trout، وفي حال فعلنا ذلك، فسيؤدي ذلك إلى إعادة تعيين freshwater بدلاً من ذلك سنُهيّئ بعد ذلك الخاصية first_name عن طريق استدعاء المتغير في الكائن خاصتنا.

الآن يمكننا استدعاء متغيرات الصنف الأساسي التي تمت تهيئتها، وكذلك استخدام المتغير الخاص بالصنف الفرعي:

```
terry = Trout()

# تهيئة الاسم الأول
terry.first_name = "Terry"

# استخدام __init__() الخاص بالصنف الأساسي عبر super()
print(terry.first_name + " " + terry.last_name)
print(terry.eyelids)

# استخدام __init__() المعاد تعريفها في الصنف الفرعي
print(terry.water)

# استخدام التابع swim() الخاص بالصنف الأساسي
terry.swim()
```

سنحصل على المخرجات التالية:

```
Terry Fish
False
freshwater
The fish is swimming.
```

تُظهر المخرجات أنّ الكائن terry المنسوخ من الصنف الفرعي Trout قادر على استخدام المتغير water الخاص بتابع الصنف الفرعي __init__()، إضافة إلى استدعاء المتغيرات first_name و last_name و eyelids الخاصة بالتابع __init__() المُعرَّف في الصنف الأساسي Fish.

يسمح لنا التابع super() المُضمن في بايثون باستخدام توابع الصنف الأساسي حتى بعد إعادة تعريف تلك التوابع في الأصناف الفرعية.

- **الوراثة المُتعدِّدة (Multiple Inheritance)**

المقصود بالوراثة المتعددة هي قدرة الصنف على أن يرث الخاصيات والتوابع من أكثر من صنف أساسي واحد، هذا من شأنه تقليل التكرار في البرامج، ولكنه يمكن أيضاً أن يُعقّد العمل، لذلك يجب استخدام هذا المفهوم بحذر.

لإظهار كيفية عمل الوراثة المتعددة، دعنا ننشئ صنفًا فرعيًا Coral_reef يرث من الصنفين Coral و Sea_anemone يمكننا إنشاء تابع في كل صنف أساسي، ثم استخدام الكلمة المفتاحية pass في الصنف الفرعي Coral_reef.

```
class Coral:


    def community(self):
        print("Coral lives in a community").
class Anemone:
    def protect_clownfish(self):
        print("The anemone is protecting the clownfish").
class CoralReef(Coral, Anemone):
    pass
```

يحتوي الصنف Coral على تابع يسمى ()community، والذي يطبع سطراً واحداً، بينما يحتوي الصنف Anemone على تابع يسمى ()protect_clownfish، والذي يطبع سطراً آخر، سنُمرِّر الصنفين كلاهما بين قوسين في تعريف الصنف CoralReef، ما يعني أنه سيرث الصنفين معاً.

دعنا الآن ننشئ كائنًا من الصنف CoralReef:

```
great_barrier = CoralReef()

great_barrier.community()

great_barrier.protect_clownfish()
```

الكائن great_barrier مُشتقٌ الصنف CoralReef، ويمكنه استخدام التوابع من كلا الصنفين الأساسيين، عند تنفيذ البرنامج، **سنحصل على المخرجات التالية:**

```
Coral lives in a community.

The anemone is protecting the clownfish.
```

تُظهِر المخرجات أنَّ التوابع من كلا الصنفين الأساسيين استُخدماً بفعالية في الصنف الفرعي.

تسمح لنا الوراثة المُتعدِّدة بإعادة استخدام الشفرات البرمجية المكتوبة في أكثر من صنف أساسي واحد، وإذا تم تعريف التابع نفسه في أكثر من صنف أساسي واحد، فإنّ الصنف الفرعي سيستخدم التابع الخاص بالصنف الأساسي الذي ظهر أولاً في قائمة الأصناف المُمرَّرة إليه عند تعريفه.

رغم فوائدها الكثيرة وفعاليتها، إلا أنَّ عليك توخي الحذر في استخدام الوراثة المُتعدِّدة، حتى لا ينتهي بك الأمر بكتابة برامج مُعقَّدة وغير مفهومة للمبرمجين الآخرين.

تعدد الأشكال polymorphisme

مفهوم متعدد الاشكال ان يكون لدينا متغير من فئة مشتقة يمكن اسناده الى متغير من فئة رئيسية كما يعرض في المثال التالي :

```
Dim Person As CLS_PERSON()

Person = New CLS_EMPLOYEE()
```

لاحظ ان المتغير person من نوع الفئة cls_person و يشير الى كائن من الفئة CLS_EMPLOYEE أي انه يمكننا اسناد أي نوع فرعي لنوع رئيسي وهذا هو مضمون مفهوم تعدد الاشكال و فيما يلي مثال اخر لتتضح الفكرة في اذهاننا :

```
Sub ChangeInfos(Person As CLS_PERSON)

 End Sub

 Sub Main()

 Dim P As New CLS_PERSON

 Dim Emp As New CLS_EMPLOYEE

 ChangeInfos(P)

 ChangeInfos(Emp)

 End Sub
```

الاجراء ChangeInfos() ينتظر براميتر من نوع الفئة الرئيسية CLS_PERSON ومع ذلك عند استدعاء هذا الاجراء نستطيع ان نمرر له متغير من نوع الفئة CLS_EMPLOYEE و كذلك متغيرات من نوع فئات مشتقة منها و هذا هو تعدد الاشكال Polymorphisme

امثله:

تعريف #Class بسيط مع خصائص ودوال

```
class Car:
    def __init__(self, make, model, year:(
        self.make = make
        self.model = model
        self.year = year

    def display_info(self:(
        print(f"{self.year} {self.make} {self.model("{

    def start_engine(self:(
        print("The engine is now running(".
```

مثال ثاني:

تعريف #Class أساسي

```
class Animal:
    def __init__(self, name:(
        self.name = name

    def speak(self:(
        print("This animal makes a sound(".
```

```
 موروثClass# تعريف

class Cat(Animal:(

    def __init__(self, name, breed:(

        super().__init__(name) استدعاء مُهيئ #  (initializer لـ (An-
imal

        self.breed = breed


    def speak(self:(

        print(f"{self.name} says Meow(".


 موروث آخرClass# تعريف

class Dog(Animal:(

    def __init__(self, name, breed:(

        super().__init__(name(

        self.breed = breed


    def speak(self:(

        print(f"{self.name} says Woof(".
```

باستخدام الخصائص والإجراءاتclassالتعديلات على

مع خصائص متقدمةClass# تعريف

```
class Rectangle:
    def __init__(self, width, height:(
        self.width = width
        self.height = height

    def area(self:(
        return self.width * self.height

    def perimeter(self:(
        return 2 * (self.width + self.height(

    def display_info(self:(
        print(f"Width: {self.width}, Height: {self.height("{
        print(f"Area: {self.area("{()
        print(f"Perimeter: {self.perimeter("{()
```

مثال مجمع:

```
أساسي#Class تعريف
class Vehicle:
    def __init__(self, make, model:(
        self.make = make
        self.model = model

    def display_info(self:(
        print(f"Vehicle: {self.make} {self.model("{

موروث#Class تعريف
class Car(Vehicle:(
    def __init__(self, make, model, num_doors:(
        super().__init__(make, model) مُهيئ استدعاء #  (initializer (
__Vehicle
        self.num_doors = num_doors

    def display_info(self:(
        print(f"Car:      {self.make}      {self.model},      Doors:
{self.num_doors("{
```

```
# تعريف Class# موروث آخر
class Truck(Vehicle:(
    def __init__(self, make, model, payload_capacity:(
        super().__init__(make, model) # استدعاء مُهيئ (initializer (
__Vehicle
        self.payload_capacity = payload_capacity

    def display_info(self:(
        print(f"Truck: {self.make} {self.model}, Payload Capacity:
{self.payload_capacity} kg("

# إنشاء كائنات من الـ #Classes
my_car = Car("Toyota", "Corolla", 4(
my_truck = Truck("Ford", "F-150", 1000(

# استدعاء دالة #display_info
my_car.display_info()  # Output: Car: Toyota Corolla, Doors: 4
my_truck.display_info()  # Output: Truck: Ford F-150, Payload Ca-
pacity: 1000 kg
```

الفصل السادس

التعامل مع الأخطاء والاستثناءات

في هذا الفصل سنتعرف على المواضيع التالية:

- مفهوم الأخطاء
- أنواع الأخطاء
- الاستثناءات
- معالجة الأخطاء
- امثلة شاملة

في البرمجة، من المهم التعامل مع الأخطاء والاستثناءات بطريقة تضمن استمرار البرنامج في العمل بدون انهيار غير متوقع. بايثون توفر آليات قوية لمعالجة الأخطاء والاستثناءات (Exceptions) لضمان استقرار البرنامج وسهولة اكتشاف المشكلات وإصلاحها.

❖ مفهوم الأخطاء

هناك العديد من الأخطاء التي من الممكن أن تواجه المبرمجين والمبتدئين عند استخدام لغات البرمجة، فالخطأ (error) هو عبارة عن مصطلح يُستخدم لوصف حدوث مشكلة في العادة تنشأ بشكل غير متوقع وتؤدي إلى تعطيل عمل البرنامج أثناء تنفيذه وهنالك العديد من أنواع هذه الأخطاء، ويُعد مفهوم معالجة الأخطاء (Exception Handling) واكتشافها في لغة البرمجة بايثون من أهم المواضيع التي يجب على كل مبرمج الإلمام بها ومعرفتها معرفة تامة.

❖ أنواع الأخطاء

- أخطاء تظهر لك أثناء كتابة الكود هذه الأخطاء يقال لها أخطاء لغوية Syntax Errors.
- أخطاء تحدث أثناء تشغيل البرنامج مما يؤدي إلى تعليقه وإيقافه بشكل غير طبيعي، هذه الأخطاء يقال لها استثناءات (Exceptions) .
- أخطاء منطقية (Logical Errors) ويقصد منها أن الكود يعمل بدون أي مشاكل لكن نتيجة تشغيل هذا الكود غير صحيحة.

إذاً أي خطأ برمجي يحدث معك أثناء تشغيل البرنامج يقال له استثناء (Exception) حتى إن كان اسم الخطأ يحتوي على كلمة Error.

بمعنى آخر أي Error يظهر لك أثناء تشغيل البرنامج يعتبر Exception.

- **الكتل الأساسية للتعامل مع الاستثناءات**

try: يحتوي على الكود الذي قد يسبب استثناء.

except: يحتوي على الكود الذي يتم تنفيذه إذا حدث استثناء.

else: يحتوي على الكود الذي يتم تنفيذه إذا لم يحدث استثناء.

finally: يحتوي على الكود الذي يتم تنفيذه دائمًا، سواء حدث استثناء أم لا.

- **مثال يتضمن خطأ لغوي (Syntax Error)**

في المثال التالي وضعنا قوس إضافي لدالة الطباعة حيث أننا كتبنا (()print بدلاً من ()print

```
x = 10
print(x))
```

سنحصل على النتيجة التالية عند تشغيل الملف Test

```
File "C:/Users/Mhamad/PycharmProjects/myapp/Test.py", line 2
    print(x))
                ^
SyntaxError: invalid syntax
```

- **ملاحظة:**

برنامج PyCharm يضع لك خطأ أحمر يوضح لك تماماً أين يوجد خطأ لغوي قبل تشغيل البرنامج، في حال تشغيل البرنامج بدون إصلاح الخطأ ستجد أن مفسر لغة بايثون أيضاً يضع لك سهم أحمر ^ يخبرك أين وجد عندك خطأ لغوي في الكود عندما حاول تنفيذه.

في المثال التالي قمنا بطباعة قيمة متغير لم نقوم أصلاً بإعطائه قيمة!

- **ملاحظة**: هنا سيحدث الخطأ وقت التشغيل عندما يكتشف مفسّر لغة بايثون أن المتغير لا يحتوي على قيمة.

- المثال

```
print(x)
```

سنحصل على النتيجة التالية عند تشغيل الملف Test

```
File "C:/Users/Mhamad/PycharmProjects/myapp/Test.py", line 1, in
<module>
NameError: name 'x' is not defined
```

في المثال التالي قمنا بتعريف list إسمه aList ويتألف من 4 عناصر، بعدها حاولنا طباعة قيمة كل عنصر فيه وحاولنا طباعة قيمة عنصر غير موجود!

- **ملاحظة**: هنا سيحدث الخطأ وقت التشغيل عندما يكتشف مفسّر لغة بايثون أنه لا يوجد عنصر يملك Index يساوي 4 في الكائن aList.

- مثال

```
#     هنا قمنا بتعريف list إسمه aList يتألف من ٤ عناصر
aList = [10, 20, 30, 40]
#       هنا قمنا بطباعة قيم عناصر الكائن aList
print(aList[0])
print(aList[1])
print(aList[2])
print(aList[3])
وهذا ما سيؤدي لحدوث خطأ برمجي وقت التشغيل هنا قمنا بطباعة قيمة عنصر غير موجود أصلاً في الكائن
#
print(aList[4])
```

سنحصل على النتيجة التالية عند تشغيل الملف **Test**

```
10
20
30
40
File "C:/Users/Mhamad/PycharmProjects/myapp/Test.py", line 8, in
<module>
    print(aList[4])

IndexError: index out of range
```

❖ الاستثناءات

الاستثناءات (Exceptions) :وهي الأخطاء التي تحدث أثناء عملية تشغيل البرنامج **وتنقسم إلى نوعين:**

١. الاستثناءات المبنية (Built-in Exceptions)

٢. الاستثناءات المُعرفة من قبل المستخدم (User defined exception)

- **بعض الأسباب التي تسبب حدوث استثناء:**

- في حال إدخال رقم index غير موجود في مصفوفة أو في متغير نصي.
- في حال كان البرنامج يتصل بالشبكة وفجأةً انقطع الاتصال.
- في حال كان البرنامج يحاول قراءة معلومات من ملف نصي، وكان هذا الملف غير موجود.

تم تقسيم الاستثناءات أو الأخطاء الأساسية في بايثون إلى عدة أنواع وكل نوع تم تمثيله في كلاس خاص، جميع هذه الكلاسات ترث من كلاس أساسي اسمه BaseException

وهذا يعني أنك إذا أردت تعريف استثناء خاص فيك في المستقبل سيكون عليك إنشاء كلاس يرث من هذا الكلاس أو من إحدى الكلاسات التي ترث منه.

إذاً أي كلاس يرث من الكلاس BaseException هو كلاس يمثل استثناء معين.

مثال:

```
try:
    result = 10 / 0 # هذا سيؤدي إلى استثناء قسمة على صفر
except ZeroDivisionError:
    print("لا يمكن القسمة على صفر.")
else:
    print("تمت العملية بنجاح.")
finally:
    print("هذه الرسالة ستظهر دائمًا.")
```

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |    |    +-- FloatingPointError
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
      |    +-- EnvironmentError
      |    |    +-- IOError
      |    |    +-- OSError
      |    |         +-- WindowsError (Windows)
      |    |         +-- VMSError (VMS)
      |    +-- EOFError
      |    +-- ImportError
      |    +-- LookupError
      |    |    +-- IndexError
      |    |    +-- KeyError
      |    +-- MemoryError
      |    +-- NameError
      |    |    +-- UnboundLocalError
      |    +-- ReferenceError
      |    +-- RuntimeError
      |    |    +-- NotImplementedError
      |    +-- SyntaxError
      |    |    +-- IndentationError
      |    |         +-- TabError
      |    +-- SystemError
      |    +-- TypeError
      |    +-- ValueError
      |         +-- UnicodeError
      |              +-- UnicodeDecodeError
      |              +-- UnicodeEncodeError
      |              +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
     +-- BytesWarning
```

❖ معالجة الأخطاء

معالجة الأخطاء (Exception Handling) في لغة البرمجة بايثون عبارة عن عملية ضمان استمرار عمل البرنامج بشكل فعّال أثناء ظهور الأخطاء، حيث تضمن هذه العملية أنه في حال حدوث خطأ (error) لن يتم إيقاف البرنامج بشكل فجائي، وهذه العملية مُهمة جداً لأنه في حال ظهور خطأ ما أثناء تنفيذ البرنامج يؤدي إلى عدم رغبة المُستخدم للعودة لاستخدام هذا البرنامج مرة أخرى، وتستخدم لغة البرمجة بايثون تعبير (try/except) في عملية تعريف مُعالج الأخطاء الذي يعمل كمُراقب في حال وقوع خطأ في جزء مُعين من الشفرة البرمجية (code) ؛ حيث يقوم باستقبال الخطأ عند حدوثه ويقوم بمعالجته بشكل سليم بحيث يجعل البرنامج يستمر بشكل فعّال ، **الجمل try و except و elseوfinally نستخدم هذه الجمل للأسباب التالية:**

- أي كود تشك بأنه قد يسبب خطأ يجب وضعه بداخل بلوك الجملة try لضمان ألا يعلق البرنامج أو يظهر خطأ مفاجئ أثناء التشغيل.
- أي كود تريد تنفيذه لمعالجة الخطأ الذي حدث في الجملة try تضعه بداخل بلوك الجملة except.
- أي كود تريد تنفيذه في حال لم يحدث خطأ في الجملة try تضعه بداخل بلوك الجملة else.
- أي كود تريد تنفيذه سواء حدث أو لم يحدث خطأ في الجملة try تضعه بداخل بلوك الجملة finally.
- بمجرد أن تضع الكود بداخل try ستكون مجبراً على وضع الجملة except أو الجملة finally بعدها أو وضع كلا الجملتين.
- كما أن برنامج PyCharm سيظهر لك تنبيه بمجرد أن تضع الكود بداخل try يخبرك فيه أنك يجب أن تضع إحدى هاتين الجملتين بعدها.
- بعد الجملة except يمكنك وضع الجملة finally أو الجملة else إن أردت لكن لا يمكنك وضع كلاهما في وقت واحد.

- في حال كنت تكتب كود يمكن أن يسبب عدة أنواع من المشاكل يمكنك وضع أكثر من جملة except حتى تعالج كل نوع من المشاكل التي قد تحدث على حدا.
- إذاً في حال أردت استخدام الجمل try و except و finally **سيكون شكل الكود كالتالي:**

```
try:
   # Something
except:
   # Handle Errors
finally:
   # Optional Clean Up Code
```

- إذاً في حال أردت استخدام الجمل try و except و else **سيكون شكل الكود كالتالي:**

```
try:
  # Something
except:
  # Handle Errors
else:
  # If No Errors, Do Extra Things
```

- **مثال**

في المثال التالي استخدمنا الجملتين try و except ولم نضع أي خطأ متعمد في الكود.

تذكر: بما أنه لن يحدث أي خطأ بداخل الجملة try فهذا يعني أنه لن يتم تنفيذ أي أمر موضوع في الجملة except بعدها سيتم إكمال تنفيذ أي أوامر موضوعة في البرنامج.

```
x = 10
في الجملة try حاولنا طباعة قيمة x, وبما أنه لا يوجد أي مشكلة هنا فهذا يعني أنه لن يتم تنفيذ أمر الطباعة
الموضوع except #
try:
print('x =', x)
except:
        print('An exception occurred')
 سيتم تنفيذ أمر الطباعة التالي في حال كان البرنامج لا يوجد فيه أي مشكلة أو حدثت مشكلة سابقاً وتم معالجتها #
print('Program still work')
```

سنحصل على النتيجة التالية عند تشغيل الملف Test

```
x = 10
Program still work
```

- **مثال**

في المثال التالي استخدمنا الجملتين try و except ووضعنا خطأ متعمد في الكود.

تذكر: بما أنه سيحدث خطأ بداخل الجملة try فهذا يعني أنه سيتم الانتقال إلى الجملة except عند حدوث الخطأ، بعدها، سيتم تنفيذ الأوامر الموضوعة فيها، ومن ثم إكمال تنفيذ أي أوامر موضوعة في البرنامج.

```
# في الجملة try حاولنا طباعة قيمة x, والذي لم نخزن أي قيمة فيه بعد
# بما أن هذا الأمر سيؤدي لحدوث خطأ فهذا يعني أنه سيتم الخروج من try والانتقال إلى الجملة except
وتنفيذ أي أمر موضوع فيها
try:
    print('x =', x)
except:
    print('An exception occurred')

    # سيتم تنفيذ أمر الطباعة التالي في حال كان البرنامج لا يوجد فيه أي مشكلة أو حدثت مشكلة سابقاً
وتم معالجتها
print('Program still work')
```

سنحصل على النتيجة التالية عند تشغيل الملف Test

```
An exception occurred
Program still work
```

امثله:

مثال على معالجة الأخطاء:

```
try:
محاولة فتح وقراءة ملف    #
    with open('example.txt', 'r') as file:
        content = file.read()
        print(content)
except FileNotFoundError:
التعامل مع حالة عدم العثور على الملف    #
    print("File not found.")
except Exception as e:
التعامل مع أخطاء أخرى    #
    print(f"An error occurred: {e}")
```

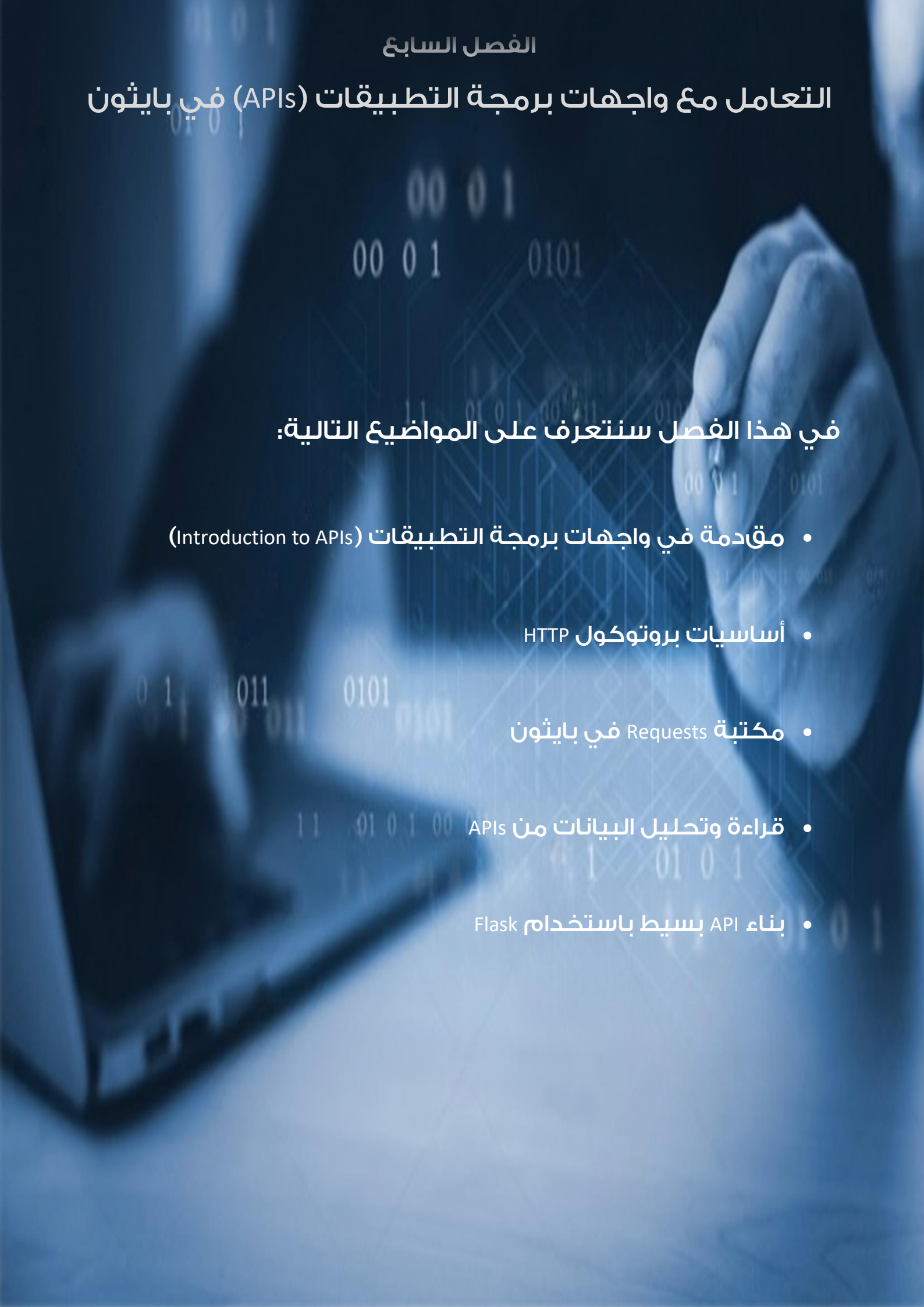
التعامل مع استثناءات متعددة:

```
try:
محاولة تنفيذ الكود    #
    number = int(input("Enter a number: "))
    result = 10 / number
except ValueError:
```

```
التعامل مع خطأ تحويل البيانات    #
    print("Invalid input. Please enter a valid number.")
except ZeroDivisionError:
التعامل مع قسمة على صفر    #
    print("Cannot divide by zero.")
```

التعامل مع الاستثناءات النهائية باستخدام finally:

```
try:
محاولة تنفيذ الكود    #
    file = open('example.txt', 'r')
    content = file.read()
except FileNotFoundError:
التعامل مع حالة عدم العثور على الملف    #
    print("File not found.")
finally:
تنفيذ الكود في جميع الأحوال، سواء حدث استثناء أم لا    #
    if 'file' in locals():
        file.close()
        print("File closed.")
```



الفصل السابع

التعامل مع واجهات برمجة التطبيقات (APIs) في بايثون

في هذا الفصل سنتعرف على المواضيع التالية:

- مقدمة في واجهات برمجة التطبيقات (Introduction to APIs)
- أساسيات بروتوكول HTTP
- مكتبة Requests في بايثون
- قراءة وتحليل البيانات من APIs
- بناء API بسيط باستخدام Flask

❖ مقدمة في واجهات برمجة التطبيقات (Introduction to APIs)

ما المقصود بواجهة برمجة التطبيقات؟

واجهات برمجة التطبيقات هي آليات تُمكّن اثنين من مكونات البرنامج الاتصال ببعضهما باستخدام مجموعة من التعريفات والبروتوكولات. فعلى سبيل المثال، يحتوي نظام البرامج في مكتب المناخ على بيانات الطقس اليومية. و"يتواصل" تطبيق المناخ على هاتفك مع هذا النظام عبر واجهات برمجة التطبيقات، ويعرض تحديثات حول المناخ يوميًا على هاتفك.

لماذا نستخدم APIs؟

إعادة الاستخدام: يمكنك استخدام خدمات جاهزة بدلاً من بناء كل شيء من الصفر.

التكامل: تمكين التطبيقات المختلفة من العمل معًا.

الأمان: يمكن تقييد الوصول إلى البيانات والميزات بشكل محدد.

استخدام APIs في بايثون

لتوضيح كيفية استخدام APIs في بايثون، سنستخدم مكتبة requests للتفاعل مع واجهات برمجة التطبيقات. requests هي مكتبة تتيح إرسال طلبات HTTP بسهولة والتعامل مع الردود.

تثبيت مكتبة requests

إذا لم تكن مثبتة بالفعل، يمكنك تثبيت المكتبة باستخدام pip:

```
pip install requests
```

مثال عملي على استخدام API

لنأخذ مثالاً على استخدام API لجلب بيانات الطقس من خدمة OpenWeatherMap:

التسجيل للحصول على مفتاح API: سجل في OpenWeatherMap واحصل على مفتاح API.

كتابة كود بايثون لاستخدام API

```
import requests


api_key = 'your_api_key’
city = 'Cairo’
url=f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid
={api_key}&units=metric’


response = requests.get(url)
if response.status_code == 200:
data = response.json()
temperature = data['main']['temp ‘]
weather = data['weather'][0]['description’]
print(f'Temperature in {city}: {temperature}°C  ‘]
{print(f'Weather: {weather ‘]
else:
print('Failed to retrieve data’)
```

نصائح عند التعامل مع APIs:

التحقق من المستندات: اقرأ مستندات API للحصول على معلومات حول النقاط النهاية (endpoints) والمعلمات (parameters).

التعامل مع الأخطاء: تحقق دائمًا من حالة الاستجابة وتعامل مع الأخطاء بشكل مناسب.

الأمان: لا تشارك مفاتيح API بشكل عام. استخدم ملفات بيئية أو أدوات لإدارة الأسرار لحمايتها.

استخدام APIs في بايثون يمكن أن يفتح لك العديد من الإمكانيات لتكامل تطبيقاتك مع خدمات أخرى، وتوسيع قدراتها بشكل كبير.

أساسيات بروتوكول HTTP

HTTP (HyperText Transfer Protocol) هو البروتوكول الأساسي المستخدم لنقل البيانات عبر الويب. يعتمد على نموذج الطلب/الاستجابة، حيث يقوم العميل (مثل متصفح الويب) بإرسال طلب إلى الخادم (مثل خادم الويب)، ويستجيب الخادم بالبيانات المطلوبة. لفهم أساسيات HTTP، دعنا نتناول العناصر الرئيسية له:

مكونات HTTP الأساسية

ا. طلبات HTTP (HTTP Requests)

طلبات HTTP هي الرسائل التي يرسلها العميل (مثل متصفح الويب) إلى الخادم لطلب موارد معينة (مثل صفحة ويب أو صورة). الطلبات تتكون من:

السطر الأول (Request Line): يحتوي على ثلاثة أجزاء:

طريقة الطلب (HTTP Method): مثل GET، POST، PUT، DELETE، إلخ.

المسار (Request URL): المسار المطلوب على الخادم.

إصدار البروتوكول (HTTP Version): مثل HTTP/1.1.

مثال:

```
GET /index.html HTTP/1.1
```

رؤوس الطلب (Request Headers): معلومات إضافية تُرسل مع الطلب مثل نوع المحتوى، الترميز، معلومات المصادقة، وغيرها.

مثال:

```
Host: www.example.com
User-Agent: Mozilla/5.0
```

جسم الطلب (Request Body): يحتوي على البيانات التي تُرسل مع الطلب، ويُستخدم عادة في الطلبات مثل POST و PUT.

٢. استجابات HTTP (HTTP Responses)

استجابة HTTP هي الرسالة التي يرسلها الخادم إلى العميل ردًا على طلب HTTP. الاستجابات تتكون من:

السطر الأول (Status Line): يحتوي على ثلاثة أجزاء:

إصدار البروتوكول (HTTP Version): مثل HTTP/1.1.

رمز الحالة (Status Code): رقم مكون من ٣ أرقام يُعبر عن حالة الاستجابة (مثل ٢٠٠، ٤٠٤، ٥٠٠).

الرسالة النصية للحالة (Status Message): وصف قصير لحالة الاستجابة.

مثال:

```
HTTP/1.1 200 OK
```

رؤوس الاستجابة (`Response Headers`): معلومات إضافية تُرسل مع الاستجابة مثل نوع المحتوى، الترميز، طول المحتوى، معلومات التخزين المؤقت، وغيرها.

مثال:

```
Content-Type: text/html
Content-Length: 1234
```

جسم الاستجابة (Response Body): يحتوي على البيانات الفعلية التي تُرسل مع الاستجابة، مثل محتوى صفحة الويب، بيانات JSON، إلخ.

رموز الحالة في HTTP (HTTP Status Codes)

رموز الحالة في HTTP تُستخدم للإشارة إلى نتيجة الطلب. هنا بعض الرموز الشائعة:

٢xx Success:

٢٠٠ OK: الطلب تم بنجاح.

٢٠١ Created: تم إنشاء مورد جديد.

٢٠٤ No Content: الطلب تم بنجاح ولكن لا توجد بيانات لإرجاعها.

٣xx Redirection:

٣٠١ Moved Permanently: المورد تم نقله بشكل دائم.

٣٠٢ Found: المورد موجود في مكان آخر مؤقتًا.

٤xx Client Error:

٤٠٠ Bad Request: الطلب غير صحيح.

٤٠١ Unauthorized: المصادقة مطلوبة.

٤٠٣ Forbidden: الوصول مرفوض.

٤٠٤ Not Found: المورد المطلوب غير موجود.

٥xx Server Error:

٥٠٠ Internal Server Error: خطأ داخلي في الخادم.

٥٠٢ Bad Gateway: استجابة غير صالحة من خادم آخر.

٥٠٣ Service Unavailable: الخدمة غير متوفرة.

طرق HTTP الشائعة (HTTP Methods)

GET: طلب للحصول على مورد.

POST: إرسال بيانات لإنشاء مورد جديد.

PUT: تحديث مورد موجود.

DELETE: حذف مورد.

HEAD: مشابه لـ GET ولكن بدون جسم الاستجابة.

OPTIONS: يعرض الطرق المدعومة من قبل الخادم للمسار المحدد.

مثال عملي باستخدام بايثون ومكتبة requests

لنقم بمثال باستخدام مكتبة requests في بايثون لإرسال طلب GET واستلام استجابة:

```
import requests

 إلى موقع معينGET# إرسال طلب
response = re-
quests.get('https://jsonplaceholder.typicode.com/posts/1('

# التحقق من رمز الحالة
if response.status_code == 200:
 طباعة بيانات الاستجابة بصيغة #    JSON
    print(response.json(()
else:
                                print('Failed to retrieve data('
```

شرح الكود:

استيراد مكتبة requests: لاستخدام طلبات HTTP.

إرسال طلب GET: إلى URL المحدد.

التحقق من رمز الحالة: إذا كانت الاستجابة ناجحة (status_code == 200)، نقوم بطباعة البيانات بصيغة JSON.

فهم أساسيات بروتوكول HTTP يمكن أن يساعدك كثيرًا في التعامل مع واجهات برمجة التطبيقات وتطوير تطبيقات الويب.

مكتبة Requests في بايثون

مكتبة requests هي مكتبة شائعة ومستخدمة بكثرة في بايثون لإرسال طلبات HTTP بسهولة. تتيح لك المكتبة إرسال طلبات GET، POST، PUT، DELETE وغيرها من الطرق، والتعامل مع الردود بشكل بسيط وسلس.

تثبيت مكتبة Requests

لتثبيت المكتبة، يمكنك استخدام pip كالتالي:

```
pip install requests
```

إرسال طلبات HTTP باستخدام Requests

١. إرسال طلب GET

يُستخدم طلب GET لجلب البيانات من الخادم. إليك مثال على كيفية إرسال طلب GET:

```
import requests

response = re-
quests.get('https://jsonplaceholder.typicode.com/posts/1('

if response.status_code == 200:
 print(response.json(()
else:
 print('Failed to retrieve data('
```

التعامل مع الاستجابات (Responses)

مكتبة `requests` توفر عدة طرق للتعامل مع الاستجابات:

`response.status_code`: للحصول على رمز الحالة.

`response.headers`: للوصول إلى رؤوس الاستجابة.

`response.text`: للحصول على نص الاستجابة.

`response.json()`: لتحليل الاستجابة كـ JSON (إذا كانت البيانات بتنسيق JSON).

مثال شامل

```
import requests

# إعدادات الطلب
url = 'https://jsonplaceholder.typicode.com/posts'
headers} =
' Content-Type': 'application/json'
{
params} =
' userId': 1
{

# إرسال طلب GET
response = requests.get(url, headers=headers, params=params(

# التحقق من الاستجابة
if response.status_code == 200:
 posts = response.json()
 for post in posts:
  print(f"Title: {post['title']}, Body: {post['body("{['
else:
 print('Failed to retrieve data('
```

قراءة وتحليل البيانات من APIs

قراءة وتحليل البيانات من APIs باستخدام مكتبة requests في بايثون تعتبر عملية شائعة جدًا في تطوير التطبيقات التي تعتمد على البيانات. سنستعرض هنا كيفية قراءة البيانات من API، وتحليلها بطرق مختلفة.

الخطوات الأساسية

إرسال طلب إلى API: باستخدام مكتبة requests.

قراءة البيانات: تحليل الاستجابة من API.

معالجة البيانات: حسب احتياجات التطبيق.

مثال عملي: قراءة بيانات JSON من API وتحليلها

لنأخذ مثالاً على API عام للحصول على بيانات وهمية: JSONPlaceholder. سنقوم بجلب البيانات من نقطة النهاية /posts وتحليلها.

١. إرسال طلب إلى API

```
import requests


url = 'https://jsonplaceholder.typicode.com/posts'
response = requests.get(url(


# التحقق من نجاح الطلب
if response.status_code == 200:
 data = response.json() # تحويل البيانات إلى تنسيق JSON
else:
 print('Failed to retrieve data('
```

٢. قراءة البيانات

البيانات التي نحصل عليها تكون في تنسيق JSON، لذا نستخدم response.json() لتحويلها إلى كائن بايثون (عادةً ما تكون قائمة تحتوي على قواميس).

```
# استعراض بعض البيانات
for post in data[:5]: # عرض أول ٥ منشورات
    print(f"Title: {post['title']}")
    print(f"Body: {post['body']}\n")
```

٣. معالجة البيانات

يمكنك استخدام مكتبة مثل pandas لمعالجة البيانات بشكل أكثر تعقيدًا، ولكن سنستعرض بعض العمليات الأساسية هنا.

حساب عدد المنشورات لكل مستخدم

```
from collections import Counter

# استخراج معرفات المستخدمين من البيانات
user_ids = [post['userId'] for post in data]

# حساب عدد المنشورات لكل مستخدم
user_post_count = Counter(user_ids)

# عرض النتائج
for user_id, count in user_post_count.items():
    print(f"User {user_id} has {count} posts")
```

إيجاد المنشور الأطول

)#body إيجاد المنشور الأطول بناءً على طول الجسم (

```
longest_post = max(data, key=lambda post: len(post['body(([']


print("Longest post(":
print(f"Title: {longest_post['title("{['
print(f"Body: {longest_post['body("{['
```

شرح الكود

إرسال طلب إلى API باستخدام requests.get().

تحويل البيانات إلى JSON باستخدام response.json().

تحويل البيانات إلى DataFrame باستخدام pandas.DataFrame().

عرض أول ٥ صفوف باستخدام df.head().

حساب عدد المنشورات لكل مستخدم باستخدام value_counts().

إيجاد المنشور الأطول: باستخدام str.len().idxmax() للعثور على طول الجسم الأطول.

بناء API بسيط باستخدام Flask

Flask هو إطار عمل ويب خفيف الوزن وسهل الاستخدام في بايثون يتيح لك بناء تطبيقات ويب وواجهات برمجة التطبيقات (APIs) بسرعة. في هذا الدليل، سنستعرض كيفية بناء API بسيط باستخدام Flask.

المتطلبات

Python مثبت على جهازك.

تثبيت مكتبة Flask. يمكنك تثبيتها باستخدام pip كالتالي:

```
pip install flask
```

خطوات بناء API باستخدام Flask

إنشاء ملف المشروع

إعداد Flask

إنشاء نقط نهاية (Endpoints)

تشغيل الخادم

١. إنشاء ملف المشروع

قم بإنشاء مجلد لمشروعك وليكن اسمه simple_api. داخل هذا المجلد، قم بإنشاء ملف بايثون جديد وليكن اسمه app.py.

٢. إعداد Flask

افتح app.py وأضف الكود التالي لإعداد Flask:

```
from flask import Flask, request, jsonify

app = Flask(__name__)

# نقط نهاية تجريبية للتحقق من عمل التطبيق
@app.route('/')
def home:()
 return "Welcome to the simple API"!

if __name__ == '__main:'__
 app.run(debug=True(
```

٣. إنشاء نقط نهاية (Endpoints)

لنقم بإنشاء نقط نهاية لتطبيقنا. سننشئ API بسيط لإدارة قائمة من المهام (to-do list).

أضف الكود التالي إلى app.py لإنشاء نقط النهاية:

```
from flask import Flask, request, jsonify


app = Flask(__name__)


# قائمة المهام التجريبية

tasks]=

'} id': 1, 'title': 'Learn Flask', 'description': 'Read Flask docu-

mentation', 'done': False,{

'} id': 2, 'title': 'Build an API', 'description': 'Build a simple

API with Flask', 'done': False{

[

# نقط نهاية للحصول على جميع المهام

@app.route('/tasks', methods=['GET('

def get_tasks:()

 return jsonify({'tasks': tasks({
```

نقط نهاية للحصول على مهمة معينة بناءً على ID#

```
@app.route('/tasks/<int:task_id>', methods=['GET(['
def get_task(task_id:(
 task = next((task for task in tasks if task['id'] == task_id),
None(
 if task is None:
  return jsonify({'error': 'Task not found'}), 404
 return jsonify(task(
```

نقط نهاية لإنشاء مهمة جديدة #

```
@app.route('/tasks', methods=['POST(['
def create_task:()
 if not request.json or not 'title' in request.json:
  return jsonify({'error': 'Bad request'}), 400
 new_task}=
'  id': tasks[-1]['id'] + 1,
'  title': request.json['title,['
'  description': request.json.get('description,(",'
'  done': False
{
 tasks.append(new_task(
 return jsonify(new_task), 201
```

نقط نهاية لتحديث مهمة موجودة

```
@app.route('/tasks/<int:task_id>', methods=['PUT(['
def update_task(task_id:(
 task = next((task for task in tasks if task['id'] == task_id),
None(
 if task is None:
  return jsonify({'error': 'Task not found'}), 404
 if not request.json:
  return jsonify({'error': 'Bad request'}), 400
 task['title'] = request.json.get('title', task['title(['
 task['description'] = request.json.get('description',
task['description(['
 task['done'] = request.json.get('done', task['done(['
 return jsonify(task(
```

نقط نهاية لحذف مهمة

```
@app.route('/tasks/<int:task_id>', methods=['DELETE(['
def delete_task(task_id:(
 global tasks
 tasks = [task for task in tasks if task['id'] != task_id[
 return jsonify({'result': True({
if __name__ == '__main:'__
 app.run(debug=True(
```

شرح الكود:

استيراد المكتبات: نستورد `Flask`، `request`، و `jsonify` لإنشاء التطبيق والتعامل مع الطلبات والاستجابات.

إعداد `Flask`: نُنشئ تطبيق `Flask`.

إنشاء قائمة المهام: قائمة تجريبية تحتوي على مهام بسيطة.

إنشاء نقط النهاية:

`/tasks (GET)`: للحصول على جميع المهام.

`/tasks/<int:task_id> (GET)`: للحصول على مهمة معينة بناءً على `ID`.

`/tasks (POST)`: لإنشاء مهمة جديدة.

`/tasks/<int:task_id> (PUT)`: لتحديث مهمة موجودة.

`/tasks/<int:task_id> (DELETE)`: لحذف مهمة.

٤. تشغيل الخادم

لتشغيل الخادم، قم بتنفيذ الملف `app.py` باستخدام بايثون:

```
python app.py
```

سيتم تشغيل الخادم على `http://127.0.0.1:5000/`.

الفصل الثامن

التعامل مع البيانات في بايثون

في هذا الفصل سوف نتعرف على المواضيع التالي:

التعامل مع البيانات النصية (Handling Text Data)

التعامل مع البيانات الرقمية (Handling Numeric Data)

فرز وتصفية البيانات (Sorting and Filtering Data)

تحويل البيانات (Data Transformation)

التخزين المؤقت واسترجاع البيانات (Storing and Retrieving Data)

التعامل مع البيانات النصية (Text Data) هو جزء مهم من البرمجة في بايثون. بايثون توفر مجموعة متنوعة من الأدوات والمكتبات للتعامل مع النصوص بطرق مختلفة. سنستعرض هنا بعض العمليات الشائعة على النصوص وكيفية تنفيذها.

العمليات الأساسية على النصوص

١. إنشاء النصوص والتعامل معها

يمكنك إنشاء نصوص في بايثون باستخدام علامات الاقتباس المفردة أو المزدوجة.

```
text = "Hello, World"!

print(text)
```

٢. الوصول إلى الأحرف والفهارس

يمكنك الوصول إلى حرف معين في النص باستخدام الفهارس.

```
first_char = text[0] # الوصول إلى أول حرف [

last_char = text[-1] # الوصول إلى آخر حرف [

print(f"First character: {first_char}, Last character: {last_char("{
```

٣. تقسيم النصوص

يمكنك تقسيم النص إلى قائمة من الكلمات باستخدام طريقة split().

```
words = text.split()

print(words(
```

٤. الربط بين النصوص

يمكنك ربط النصوص معًا باستخدام عامل الربط +.

```
greeting = "Hello"

name = "Alice"

message = greeting + ", " + name"!" +

print(message(
```

٥. تغيير حالة الأحرف

يمكنك تغيير حالة الأحرف في النص باستخدام طرق مثل upper() و lower().

```
print(text.upperة كبيرة إلى أحرف # ())

print(text.lowerة صغيرة إلى أحرف # ())
```

٦. استبدال النصوص

يمكنك استبدال جزء من النص باستخدام طريقة replace().

```
new_text = text.replace("World", "Python("

print(new_text(
```

التعامل مع النصوص الكبيرة

قراءة وكتابة الملفات النصية

يمكنك قراءة وكتابة النصوص من وإلى الملفات باستخدام الدوال open()، read()، و write().

قراءة النص من ملف

```
with open('example.txt', 'r') as file:
  content = file.read()
  print(content(
```

مكتبة re للتعامل مع التعبيرات العادية

مكتبة re في بايثون تستخدم للتعامل مع التعبيرات العادية (Regular Expressions) والتي تتيح لك تنفيذ عمليات بحث واستبدال معقدة.

مثال على استخدام re :

البحث عن نمط معين

```
import re
text = "The rain in Spain"
pattern = r"\b\w{4}\b أحرف ٤ تحتوي على التي الكلمات عن البحث #  "
matches = re.findall(pattern, text(
print(matches(
```

استبدال نمط معين

```
import re
```

```
text = "The rain in Spain"

pattern = r"Spain"

replacement = "Italy"

new_text = re.sub(pattern, replacement, text(

print(new_text)
```

التعامل مع البيانات الرقمية

التعامل مع البيانات الرقمية هو جزء أساسي من البرمجة في بايثون. توفر بايثون مجموعة متنوعة من الأدوات والمكتبات للتعامل مع الأرقام وإجراء العمليات الحسابية والإحصائية.

العمليات الأساسية على الأرقام

١. العمليات الحسابية الأساسية

بايثون تدعم العمليات الحسابية الأساسية مثل الجمع، الطرح، الضرب، والقسمة

```
a = 10

b = 3


(a + b ,":جمع")print

(a - b ,":طرح")print

(a * b ,":ضرب")print

(a / b ,":قسمة")print

(a // b ,":قسمة صحيحة")print

(a % b ,":باقي القسمة")print

(a ** b ,":الأس")print
```

٢. العمليات الرياضية المتقدمة

يمكنك استخدام مكتبة math للقيام بالعمليات الرياضية المتقدمة مثل الجذر التربيعي، اللوغاريتم،

وحساب المثلثات

```
import math

x = 16

print ,":الجذر التربيعي")math.sqrt(x((

print ,":اللوغاريتم الطبيعي")math.log(x((

print ,":الجيب")math.sin(math.radians(30(((

print ,":جيبة التمام")math.cos(math.radians(60(((

print ,":ظل الزاوية")math.tan(math.radians(45(((
```

العمليات على المصفوفات الثنائية (D٢)

```
# إنشاء مصفوفة ثنائية الأبعاد

matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9([[

print\:المصفوفة الثنائية الأبعاد")n", matrix(


# العمليات الحسابية على المصفوفات الثنائية الأبعاد

print\:جمع ٢ إلى كل عنصر")n", matrix + 2(

print\:٣ ضرب كل عنصر في")n", matrix * 3(

print ,":متوسط الأعمدة")np.mean(matrix, axis=0((

print ,":متوسط الصفوف")np.mean(matrix, axis=1((
```

التعامل مع الأعداد الكبيرة والأعداد العشوائية

الأعداد العشوائية

يمكنك استخدام مكتبة random لتوليد الأعداد العشوائية.

```
import random
print ,":١٠ و ١ عدد صحيح عشوائي بين")random.randint(1, 10((
print ,":١ و ٠ عدد عشوائي بين")random.random(()

# توليد قائمة من الأعداد العشوائية
random_numbers = [random.randint(1, 100) for _ in range(10[(
print ,":قائمة من الأعداد العشوائية")random_numbers(
```

فرز وتصفية البيانات في بايثون

فرز وتصفية البيانات هي عمليات شائعة ومهمة في تحليل البيانات ومعالجتها. توفر بايثون ومكتباتها أدوات قوية للتعامل مع هذه العمليات.

فرز البيانات

باستخدام القوائم البسيطة

يمكنك استخدام دالة sorted() لفرز القوائم في بايثون.

```
numbers = [5, 2, 9, 1, 5, 6[
sorted_numbers = sorted(numbers(
print ,":الأعداد بعد الفرز")sorted_numbers(

# الفرز العكسي
sorted_numbers_desc = sorted(numbers, reverse=True(
print ,":الأعداد بعد الفرز العكسي")sorted_numbers_desc(
```

استخدام مكتبة Pandas

مكتبة pandas توفر أدوات قوية ومتقدمة لفرز وتصفية البيانات في الجداول (DataFrames).

إنشاء DataFrame

```
import pandas as pd
data} =
'    name': ['Alice', 'Bob', 'Charlie', 'David,['
'    grade': [88, 72, 90, 85,[
'    age': [20, 21, 19, 22[
{

df = pd.DataFrame(data(
print(df(
```

فرز البيانات

```
' فرز البيانات حسب العمود #grade'
sorted_df = df.sort_values(by='grade('
print' البيانات بعد الفرز حسب العمود")grade':\n", sorted_df(

' الفرز العكسي حسب العمود #grade'
sorted_df_desc = df.sort_values(by='grade', ascending=False(
print' البيانات بعد الفرز العكسي حسب العمود")grade':\n", sort-
ed_df_desc(
```

تصفية البيانات

```
'أكبر من ٨٠ #grade تصفية الصفوف حيث'

filtered_df = df[df['grade'] > 80[

print\:(الدرجات أكبر من ٨٠)"البيانات بعد التصفية (n", filtered_df(


'أقل من ٢١ #age تصفية الصفوف حيث'

filtered_df_age = df[df['age'] < 21[

print\:(العمر أقل من ٢١)"البيانات بعد التصفية (n", filtered_df_age(
```

ملاحظات:

مرونة Pandas: تعتبر مكتبة pandas من أكثر الأدوات مرونة وكفاءة في التعامل مع البيانات المهيكلة، مما يجعلها الخيار الأفضل في تحليل البيانات.

التعامل مع البيانات الكبيرة: توفر pandas ميزات متقدمة مثل التعامل مع البيانات الكبيرة والتكامل مع مكتبات أخرى لتحليل البيانات.

سهولة الاستخدام: بايثون توفر أدوات بسيطة وفعالة للتعامل مع البيانات النصية والرقمية، مما يسهل على المبرمجين معالجة البيانات بطرق متنوعة وفعالة.

تحويل البيانات (Data Transformation)

تحويل البيانات هو عملية تعديل وتغيير شكل البيانات لجعلها أكثر ملاءمة للتحليل أو الاستخدام. يوفر بايثون العديد من الأدوات والمكتبات التي تساعد في إجراء تحويلات البيانات بسهولة وفعالية

العمليات الأساسية لتحويل البيانات

١. تغيير نوع البيانات

يمكنك تحويل نوع البيانات في بايثون باستخدام الدوال المدمجة مثل int(), float(), str().

```
# تحويل نص إلى عدد صحيح
num_str = "123"
num_int = int(num_str(
print(type(num_int), num_int(
# تحويل عدد صحيح إلى عدد عشري
num_float = float(num_int(
print(type(num_float), num_float(
# تحويل عدد إلى نص
num_str_again = str(num_float(
print(type(num_str_again), num_str_again(
```

٢. استخدام تعبيرات الفلترة والتعيين

يمكنك استخدام تعبيرات الفلترة (List Comprehensions) لتغيير وتعديل البيانات في القوائم.

```
numbers = [1, 2, 3, 4, 5[
squared_numbers = [x**2 for x in numbers[
print ,":الأعداد المربعة")squared_numbers(

# تحويل قائمة من النصوص إلى أعداد صحيحة
str_numbers = ["1", "2", "3", "4", "5["
int_numbers = [int(x) for x in str_numbers[
print ,":الأعداد الصحيحة")int_numbers(
```

تحويل القيم باستخدام التعيينات

يمكنك تحويل القيم في الأعمدة باستخدام apply() ودوال مخصصة

تحويل الدرجات إلى تصنيفات

```
def grade_to_letter(grade:(
    if grade >= 90:
        return 'A'
    elif grade >= 80:
        return 'B'
    elif grade >= 70:
        return 'C'
    else:
        return 'D'

df['grade_letter'] = df['grade'].apply(grade_to_letter(
print\:بعد تحويل الدرجات إلى تصنيفات")n", df(
```

تحويل نوع البيانات

يمكنك تحويل نوع البيانات في الأعمدة باستخدام astype().

```
# تحويل الأعمار إلى أعداد صحيحة
df['age'] = df['age'].astype(int(
print\:صحيحة إلى أعداد الأعمار تحويل بعد")n", df(
```

التخزين المؤقت واسترجاع البيانات (Storing and Retrieving Data) في بايثون

تخزين البيانات المؤقتة واسترجاعها هو جزء مهم من العديد من التطبيقات والأنظمة. يمكن تخزين البيانات بعدة طرق منها الملفات النصية، الملفات بصيغة JSON، قواعد البيانات، وغيرها. سنستعرض هنا بعض الطرق الشائعة لتخزين واسترجاع البيانات في بايثون.

تخزين البيانات في ملفات نصية

كتابة البيانات إلى ملف نصي

يمكنك استخدام دالة open() لفتح ملف نصي ودالة write() لكتابة البيانات إليه.

```
# كتابة البيانات إلى ملف نصي
data = "Hello, world!\nThis is a test file".
with open("example.txt", "w") as file:
    file.write(data(
```

قراءة البيانات من ملف نصي

يمكنك استخدام دالة open() مع دالة read() لقراءة البيانات من ملف نصي.

```
# قراءة البيانات من ملف نصي
with open("example.txt", "r") as file:
    data = file.read()
    print(data(
```

ملاحظات ختامية حول تخزين واسترجاع البيانات في بايثون

اختيار طريقة التخزين

عند اختيار طريقة لتخزين البيانات، هناك عدة عوامل يجب أخذها في الاعتبار:

حجم البيانات:

إذا كنت تتعامل مع كميات صغيرة من البيانات مثل الإعدادات البسيطة أو النصوص، فيمكنك استخدام الملفات النصية أو ملفات JSON.

إذا كانت البيانات أكبر قليلاً ولكن لا تزال يمكن التعامل معها بسهولة، يمكن استخدام ملفات CSV.

إذا كنت تتعامل مع كميات كبيرة من البيانات أو تحتاج إلى عمليات معقدة مثل الفلترة والبحث، فمن الأفضل استخدام قواعد البيانات

SQLite.

❖ المراجع

١.	د ترجمة م.محمد شيخو ، ٢٠٢٠ م ، إبدا مع Python ، شعاع للنشر و العلوم
٢.	ميلاد وزان ترجمة د.علاء طعيمه ، ٢٠٢١ م، تعلم الآلة وعلم البيانات ، جامعة القادسية
٣.	John Paul Mueller,2018, Beginning Programming with Python, Willy
٤.	بايثون للجميع . تأليف: د. تشارلز سيفيرنس . ترجمة: منصة الكترونكس غو
٥.	بايثون عن طريق الأمثلة ترجمة و اعداد الدكتور علاء طعيمه


أكاديمية التعلم
Academy Of Learning
تحت إشراف
المؤسسة العامة للتدريب التقني والمهني
Technical and Vocational Training Corporation