

مقدمة في

البرمجة عن

طريق لغة

بايثون 

مقدمة في البرمجة عن طريق لغة بايثون

ألفه : محمد الغافلي

thebsom@hotmail.com

تم بحمد الله بتاريخ

١٤٣٣/٠٩/٠٤

٢٠١٢/٠٧/٢٣

هذا العمل منشور بموجب رخصة

المشاع الإبداعي: نسب العمل - المشاركة بالمثل ٣,٠



<http://creativecommons.org/licenses/by-sa/3.0/deed.en>

# الافتتاحية

انتشار الحواسب بمختلف أشكالها من الحواسب الشخصية إلى المحمولة إلى الهواتف الذكية زاد من اهتمام الناس بالبرمجة و حب تعلمها خصوصا بين أولئك الذين في المرحلة الثانوية ، من أكثر الأمور التي شدتني و غيري للبرمجة هي الألعاب الإلكترونية التي كنت ألعبها و أتساءل كثيرا كيف يتم إنشاءها.

بعد دخولي لعالم البرمجة عرفت أن المبرمج مسؤول عن الكثير من الأمور في مختلف المجالات و أنه يفكر بطريقة قد تختلف عما يراه مستخدم البرنامج ، من أهم فوائد البرمجة تنفيذ المهام بطريقة تلقائية دون القيام بها يدويا.

لم أتخيل أنني سأحتاج البرمجة يوما في الحياة العملية ، خلال دراستي الجامعية وصلت لمرحلة التدريب العملي و بدأت أعمل في إحدى شركات الاتصالات و لم تكن شركة برمجة ، يوما ما احتجنا ترتيب عدد هائل من الملفات ، لم تكن مشكلة معقدة و لكن عدد الملفات و طريقة ترتيبها يأخذ وقتا طويلا جدا و قيامنا بترتيبها يدويا يؤدي للكثير من الأخطاء ، استغرقت ساعتين في أول يوم لترتيب جزء منها و كان أمرا مملا جدا ، في اليوم التالي قررت أن أجعل الحاسب يقوم بهذه المهمة ، استغرقت في ذلك اليوم نصف ساعة فقط لكتابة برنامج صغير و التأكد من عمله بشكل صحيح يقوم بالبحث في الملفات و يقوم بترتيبها بالطريقة المطلوبة في وقت قصير و بأخطاء أقل ، عندها علمت أن كثيرا من المشاكل التي تواجه الناس و المؤسسات يستطيع المبرمج حلها بشكل أسرع و أكثر كفاءة.

في نهاية هذه الكلمات أحمد الله. الذي أنعم علي بمعرفة أسأله أن يوفقني لأنفع بها الناس ، أود أن أشكر كل من ساهم في هذا العمل ابتداء من والدتي الحبيبة التي وقفت معي في مختلف مراحل الحياة ، شكري و تقديرى أيضا

لأخي العزيز حسين المطوع الذي كان له فضل كبير في بداية مسيرتي البرمجية ، الشكر موصول أيضا للأخ الفاضل معاذ سقّان الذي يعود له الفضل في مراجعة الدروس لتظهر بأفضل صورة و للأخ العزيز محمد الشّمري لمساهمته في إزالة بعض العوائق التي وقفت في طريقي خلال كتابتي لهذه المادة ، أشكر أيضا كل من ساهم في تصميم الأدوات التي تم استخدامها في هذا الكتاب و منهم فريق برنامج ليبر أوفيس و مصمموا الخطوط المستخدمة و هي خطوط ديجافو و خطوط درويد و خط سالم ، لهم جميعا و لكل من ساهم ممن لم نذكره جزيل الشكر و خالص الامتنان.

محمد الغافلي

thebsom@hotmail.com

# نبذة عن الكاتب

بدايتي في البرمجة كانت في الفيچوال بيسك في عام ١٤٢٦ للهجرة الموافق ٢٠٠٥ للميلاد من أحد الكتب التي اشتريتها ، قفزتي في البرمجة كانت بلغة جافا في عام ٢٠٠٦ مع أحد أصدقائي الذي تعب معي و أدخلني عالم البرمجة ، تعلمت لغة سي ++ عام ٢٠٠٨ من أحد المواقع و بقيت أبرمج بها بين فترة و أخرى ، كتبت دروس سي ++ و نشرتها في شبكة الإنترنت نهاية عام ٢٠١٠ متخذا اسم "روح سامية" اسما مستعارا ، تعلمت برمجة تطبيقات أندرويد و بعدها تعلمت لغة بايثون بعد نصيحة من أحد أصدقائي ، من أعمالي خلال حياتي الجامعية تصميم معالج و تصميم لغة تجميع و مجمع (Assembler) له ، قمت بكتابة برامج عدة معظمها كان لتعلم برمجة مهام محددة منها برنامج صور ذات بعدين و مشغل صوتيات بسيط.

# مقدمة الكتاب

مقدمة في البرمجة عن طريق لغة بايثون ، أردنا أن نضع فكرتين في عنوان الكتاب ، الأولى هي "مقدمة في البرمجة" أي أن هذا الكتاب يعطي مقدمة في البرمجة و أفكارها ليس بلغة برمجة محددة بل أفكار مشتركة في مختلف لغات البرمجة ، الفكرة الثانية هي "عن طريق لغة بايثون" أي أن لغة بايثون ستكون وسيلة لإيصال هذه الأفكار.

يمكن أن تحل أي لغة أخرى محل لغة بايثون للتعريف بالبرمجة لكن لغة بايثون لها ميزات كثيرة كلغة أولى تساعد الدارس و الكاتب في التركيز على أفكار البرمجة أكثر من اللغة بحد ذاتها ، من هذه الميزات السهولة و الترتيب. على أن لغة بايثون لغة سهلة مقارنة بلغات أخرى مثل جافا و سي و سي++ إلا أنها تبقى قوية و مفيدة ، هناك جهات و مؤسسات تستخدم لغة بايثون لأغراض عدة و من أبرزها [بحسب موقع بايثون](#) شركة جوجل التي تحتل بايثون جزءا من محرك بحثها.

هذا الكتاب يستهدف المبتدئين في البرمجة و محتواه لا يفترض أي معرفة مسبقة بالبرمجة ، الكتاب يركز على الأفكار البرمجة و ليس على كل ميزات اللغة و سيصادف الدارس بعض الأوامر التي يمكن كتابتها بأكثر من طريقة بلغة بايثون لكن الكتاب لن يذكر إلا عددا محدودا من الطرق و ميزات اللغة لإيصال فكرة معينة.

الإصدار الثاني من لغة بايثون هو الأكثر استخدام وقت كتابة هذا الكتاب لكن الكتاب يستخدم الإصدار الثالث من اللغة لأنه أحدث إصدار و يعتبر مستقبل اللغة و هو الإصدار الذي نتوقع انتشاره في الفترة القادمة و نتوقع أن يستخدمه الدارس إذا قرر الاستمرار في تعلم لغة بايثون.

يبدأ الكتاب في الدروس الثلاثة الأولى بشرح تنصيب البرامج التي نحتاجها للبرمجة بلغة بايثون كتمهيد لبداية التعلم ، بعدها يبدأ بشرح بعض أوامر

البسيطة في اللغة تدريجها حتى يصل إلى كيفية التحكم بتنفيذ الأوامر ، أخيرا يعطي الكتاب فكرة عن كيفية ترتيب البرنامج فيما عدده خمسة عشر درسا ، ينتهي الكتاب بمشروع بسيط يساعد الدارس في تطبيق الأفكار التي تعلمها.

ينقسم كل درس في هذا الكتاب إلى عدة أقسام يتحدث كل قسم عن جزء من الفكرة العامة في الدرس ، قد تحتاج بعض الدروس ملفات يتم إرفاق روابط لها مع الدرس لتحميلها من الإنترنت ، يتم استخدام هذا الخط لتعليم النص الذي يحتوي على رابط ، يتم تعليم بعض الأمور المهمة في الدرس . بهذا الخط ، معظم الدروس ستحتوي على أوامر بلغة بايثون ، سيتم كتابة هذه الأوامر بالخط التالي :

```
print ( 'this is a python program' )
```

أسلوب التلوين المستخدم للأوامر سيكون مشابها للتلوين المستخدم في برنامج جي إدت الذي سيتم شرح تنصيبه في الدرس الثاني.

هذا الكتاب يعتبر خطوة أولى في المسيرة البرمجية للدارس ، بعد الانتهاء من هذه الدروس بإمكان الدارس الإكمال في تعلم خصائص لغة بايثون و كيفية الاستفادة منها أكثر و بإمكانه أيضا التحول إلى لغة أخرى و سيلاحظ التشابه الكبير بين الأفكار في لغة بايثون و في مختلف اللغات و يتعلم الاختلاف بينها ، سواء اختار الدارس التعمق في لغة بايثون أو تعلم لغة أخرى و استخدامها بإمكانه زيادة معرفته و مهاراته في البرمجة.

# الفهرس

- ١ - تنصيب بايثون.....١
- ١.١ - تنصيب بايثون في أنظمة أوبونتو.....٢
- ١.٢ - تنصيب بايثون في أنظمة ماك.....٦
- ١.٣ - تنصيب بايثون في أنظمة ويندوز.....١٠
- ١.٤ - تعلمنا في هذا الدرس.....١٢
- ٢ - تنصيب برنامج جي إدت.....١٣
- ٢.١ - تنصيب المحرر في أنظمة أوبونتو.....١٤
- ٢.٢ - تنصيب المحرر في أنظمة ماك.....١٤
- ٢.٣ - تنصيب المحرر في أنظمة ويندوز.....١٥
- ٢.٤ - استخدام المحرر.....١٦
- ٢.٥ - تعلمنا في هذا الدرس.....١٧
- ٣ - تشغيل بايثون.....١٨
- ٣.١ - تشغيل مفسر بايثون.....١٩
- ٣.٢ - تشغيل أوامر بايثون عن طريق المفسر.....٢١
- ٣.٣ - تشغيل أوامر بايثون من ملف.....٢١
- ٣.٤ - تعلمنا في هذا الدرس.....٢٢
- ٤ - العمليات الحسابية.....٢٣
- ٤.١ - الجمع و الطرح والضرب و القسمة.....٢٤
- ٤.٢ - القسمة الصحيحة و باقي القسمة.....٢٥
- ٤.٣ - أولوية تنفيذ العمليات الحسابية.....٢٦
- ٤.٤ - تعلمنا في هذا الدرس.....٢٧
- ٥ - الطباعة.....٢٨
- ٥.١ - أمر الطباعة.....٢٩
- ٥.٢ - النصوص.....٣٠
- ٥.٣ - رموز الإفلات.....٣٠



٥.٤	- كتابة برنامج في ملف	٣١
٥.٥	- تعلمنا في هذا الدرس	٣٢
٦	- المتغيرات	٣٣
٦.١	- المتغيرات	٣٤
٦.٢	- متغيرات النصوص	٣٦
٦.٣	- جمع النصوص	٣٦
٦.٤	- تعلمنا في هذا الدرس	٣٧
٧	- الإدخال	٣٨
٧.١	- الإدخال	٣٩
٧.٢	- إدخال الأعداد	٤٠
٧.٣	- تعلمنا في هذا الدرس	٤٠
٨	- التعامل مع الملفات	٤١
٨.١	- فائدة الملفات	٤٢
٨.٢	- فتح ملف	٤٢
٨.٣	- الكتابة في ملف	٤٣
٨.٤	- القراءة من ملف	٤٤
٨.٥	- تعلمنا في هذا الدرس	٤٦
٩	- الجمل الشرطية ١	٤٧
٩.١	- القيم المنطقية	٤٨
٩.٢	- عمليات المقارنة	٤٨
٩.٣	- عبارة if	٥٠
٩.٤	- تعدد الأوامر في عبارة if	٥١
٩.٥	- تعلمنا في هذا الدرس	٥٢
١٠	- الجمل الشرطية ٢	٥٣
١٠.١	- الأوامر المنطقية	٥٤
١٠.٢	- عبارة elif	٥٥
١٠.٣	- عبارة else	٥٦

٥٨.....	١٠.٤ - تعلمنا في هذا الدرس
٥٩.....	١١ - الحلقات ١
٦٠.....	١١.١ - أمر while
٦١.....	١١.٢ - مقاطعة أوامر الحلقة
٦٢.....	١١.٣ - تعلمنا في هذا الدرس
٦٣.....	١٢ - القوائم
٦٤.....	١٢.١ - القوائم
٦٥.....	١٢.٢ - التوصل لعناصر القائمة
٦٦.....	١٢.٣ - التوصل لأحرف العبارات النصية
٦٧.....	١٢.٤ - تعلمنا في هذا الدرس
٦٨.....	١٣ - الحلقات ٢
٦٩.....	١٣.١ - تعديل قيم القائمة باستخدام while
٧٠.....	١٣.٢ - معرفة عدد عناصر القائمة
٧٠.....	١٣.٣ - حذف العناصر من القائمة
٧١.....	١٣.٤ - تعلمنا في هذا الدرس
٧٢.....	١٤ - الدّوال
٧٣.....	١٤.١ - الدّوال
٧٤.....	١٤.٢ - إعطاء قيمة للدالة
٧٥.....	١٤.٣ - إرجاع قيمة من الدّالة
٧٦.....	١٤.٤ - تعلمنا في هذا الدرس
٧٧.....	١٥ - الوحدات
٧٨.....	١٥.١ - الوحدات
٧٨.....	١٥.٢ - استيراد الوحدات
٧٩.....	١٥.٣ - استخدام أوامر الوحدة
٨٠.....	١٥.٤ - تعلمنا في هذا الدرس
٨١.....	١٦ - مشروع السلسلة
٨٢.....	١٦.١ - فكرة المشروع

١٦.٢	- أقسام المشروع	٨٢.....
١٦.٣	- مهمة المشارك في المشروع	٨٧.....
١٦.٤	- ملخص المشروع	٨٨.....
١٦.٥	- حل المشروع	٨٩.....



# الدرس الأول

تنصيب بايثون

تحتاج برامج بايثون إلى برنامج خاص يقوم بتشغيلها ، لا يمكن تشغيل برامج بايثون إلا بوجود هذا البرنامج ، هذا الدرس يغطي كيفية تنصيب برنامج بايثون في كل من أنظمة أوبونتو و ماك و ويندوز ، بإمكانك الانتقال للقسم الذي يشرح طريقة التنصيب على نظامك ، يأتي هذا الدرس مع ملفات تساعدك في تنصيب و تشغيل بايثون لكل نظام ، بإمكانك تحميل الملفات الخاصة بنظام أوبونتو و ماك و ويندوز.

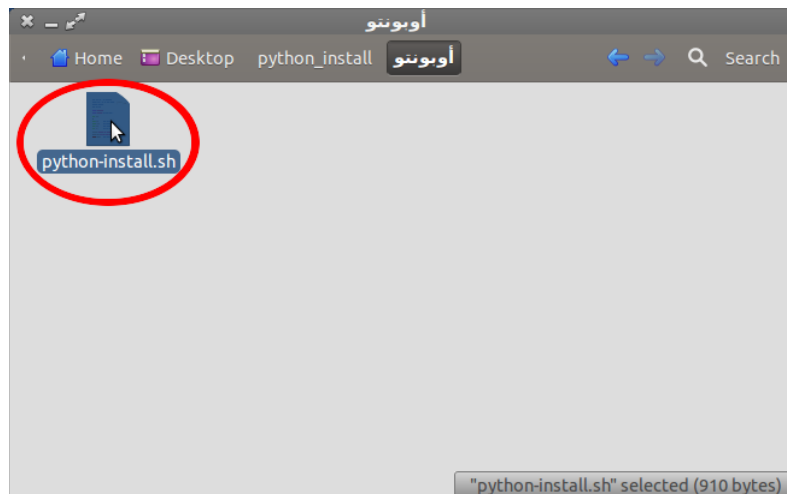
## ١.١ تنصيب بايثون في أنظمة أوبونتو

بإمكانك تنصيب الإصدار الثالث من بايثون من مستودعات البرامج في النظام باستخدام Ubuntu Software Center أو Synaptic ، تجد الإصدار الثالث من بايثون باسم python3.

بإمكانك أيضا تحميل ملف تنصيب بايثون من [هذا الرابط](#) و اتباع الخطوات التالية :

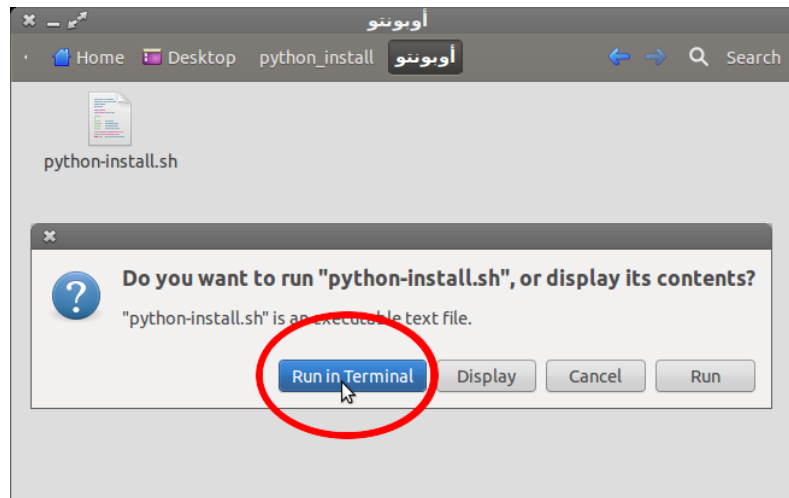
١. تأكد من اتصالك بالإنترنت.

٢. فك ضغط الملف الذي قمت بتحميله و ستجد فيه ملفا اسمه [python-install.sh](#) ، قم بتشغيله في الصورة رقم ١.



الصورة ١: انقر مرتين على الملف python-install.sh

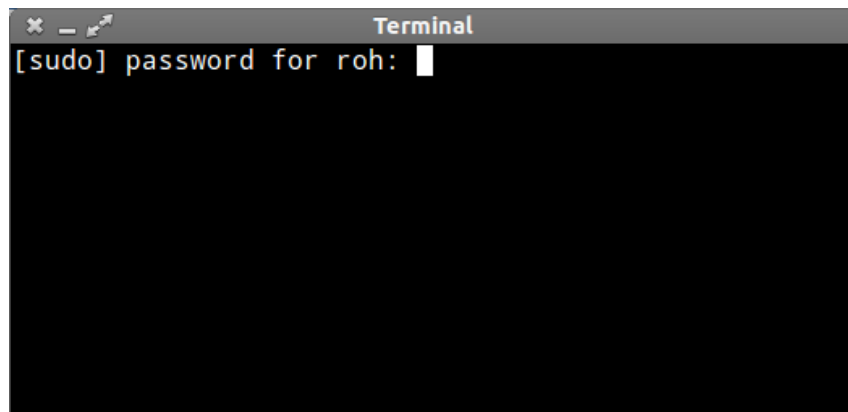
٣. ستظهر لك خيارات تشغيل الملف ، اختر الخيار *Run in Terminal* كما في الصورة رقم ٢.



الصورة ٢: اختر Run in Terminal

٤. سيتم طلب كلمة السر لمشرف الجهاز حتى يتم تنصيب البرنامج ، قم بإدخال كلمة السر ثم اضغط مفتاح الإدخال Enter كما في الصورة رقم ٣.

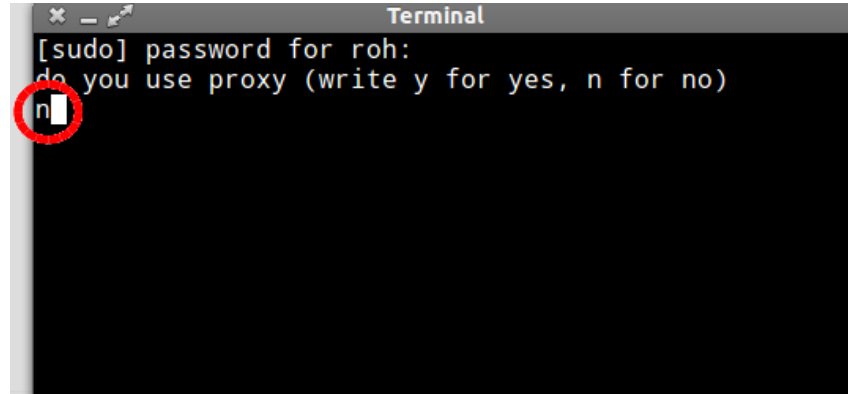
إذا كتبت كلمة السر فإن البرنامج يقرأها لكنه لا يظهر الأحرف بل بيقفيها مخفية.



الصورة ٣: اكتب كلمة السر ثم اضغط مفتاح Enter

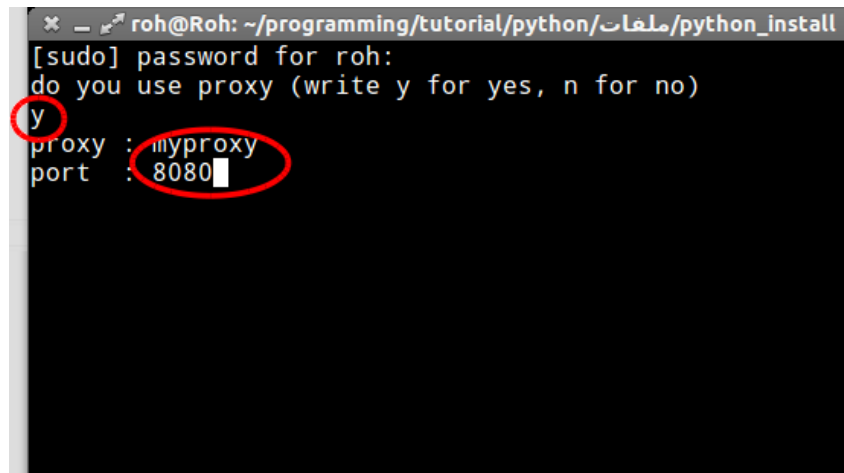
٥. سيسألك البرنامج إذا كنت تستخدم بروكسي لاتصالك أو لا.

إذا لم تكن تستخدم بروكسي اكتب الحرف n ثم اضغط مفتاح الإدخال Enter كما في الصور ٤.



الصورة ٤: اكتب الحرف n إذا لم تكن تستخدم بروكسي

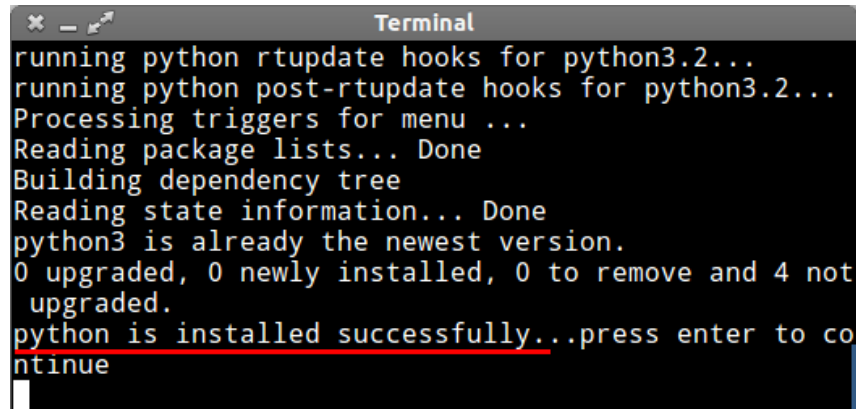
إذا كنت تستخدم بروكسي اكتب الحرف y ثم اضغط مفتاح الإدخال Enter ، سيسألك البرنامج عن عنوان البروكسي ثم منفذ البروكسي ، قم بإدخالها كما في الصورة ٥.



الصورة ٥: اكتب عنوان البروكسي و المنفذ

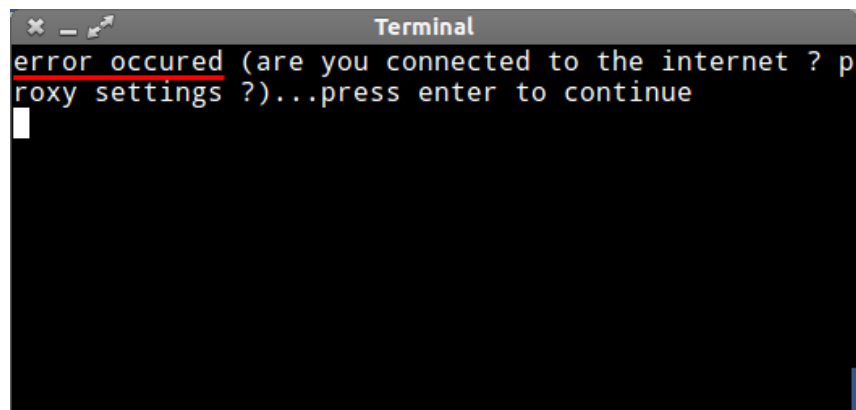
٦. انتظر حتى يتم تنصيب بايثون و تظهر لك رسالة إنهاء التنصيب بعدها اضغط مفتاح الإدخال Enter كما في الصورة ٦.

إذا ظهرت لك رسالة خطأ كما في الصورة ٧ فلم يتم تنصيب بايثون بسبب مشكلة على الأغلب تكون متعلقة باتصالك بالإنترنت.

A terminal window titled "Terminal" with a dark background and light green text. The text shows the process of updating Python hooks and installing Python 3.2. The final line, "python is installed successfully...press enter to continue", is underlined in red. A cursor is visible at the end of the line.

```
running python rtupdate hooks for python3.2...
running python post-rtupdate hooks for python3.2...
Processing triggers for menu ...
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3 is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 4 not
upgraded.
python is installed successfully...press enter to co
ntinue
```

الصورة ٦: رسالة إنهاء التنصيب

A terminal window titled "Terminal" with a dark background and light green text. The text shows an error message: "error occured (are you connected to the internet ? proxy settings ?)...press enter to continue". The first part of the message is underlined in red. A cursor is visible at the end of the line.

```
error occured (are you connected to the internet ? p
roxy settings ?)...press enter to continue
```

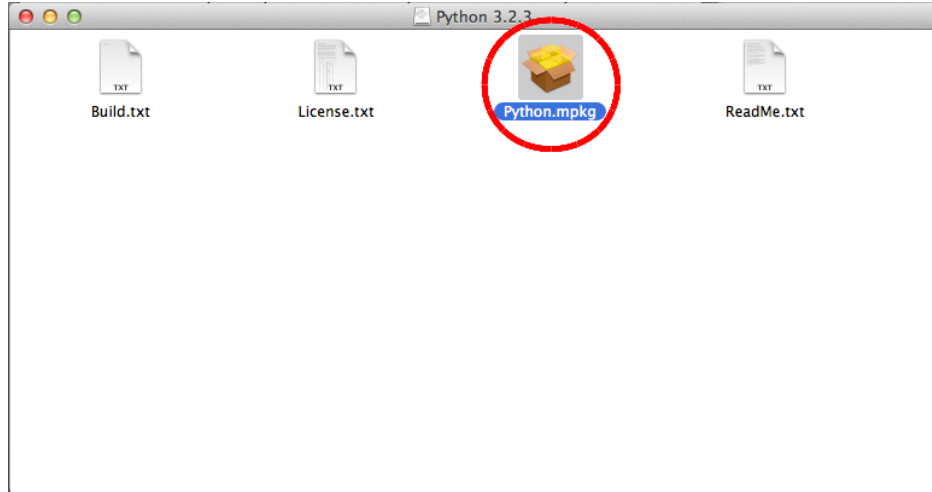
الصورة ٧: رسالة خطأ في التنصيب



## ١.٢ تنصيب بايثون في أنظمة ماك

بإمكانك تحميل ملف تنصيب بايثون من [هذا الرابط](#) و اتباع الخطوات التالية :

١. فك ضغط الملف الذي قمت بتحميله و ستجد ملفين للتنصيب :
  - [python-3.2.3-macosx10.3.dmg](#) : للتنصيب في أنظمة ماك ١٠.٣ إلى ١٠.٦
  - [python-3.2.3-macosx10.6.dmg](#) : للتنصيب في أنظمة ماك ١٠.٦ و ١٠.٧.
٢. شغل ملف التنصيب المناسب لنظامك بحسب ما تم شرحه في النقطة السابقة.
٣. ستظهر لك نافذة فيها عدة ملفات ، شغل الملف [Python.mpkg](#) كما في الصورة رقم ٨.



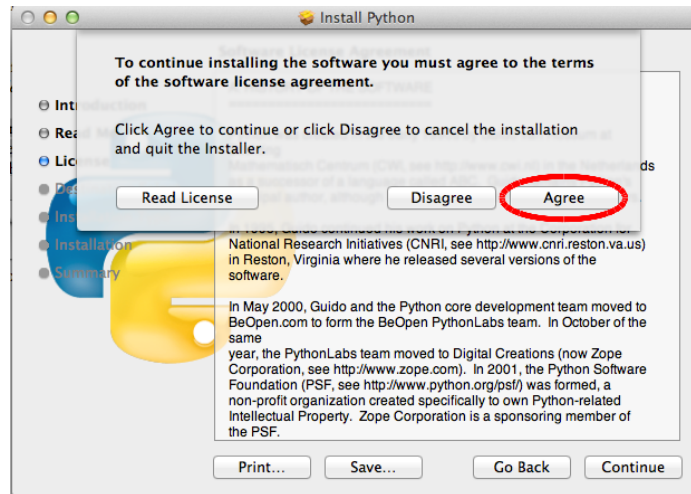
الصورة ٨: شغل الملف Python.mpkg

٤. اضغط على زر Continue في جميع النوافذ التالية التي ستظهر كما في الصورة رقم ٩ ، ستصل بعد عدة نوافذ إلى نافذة الموافقة على الرخصة.



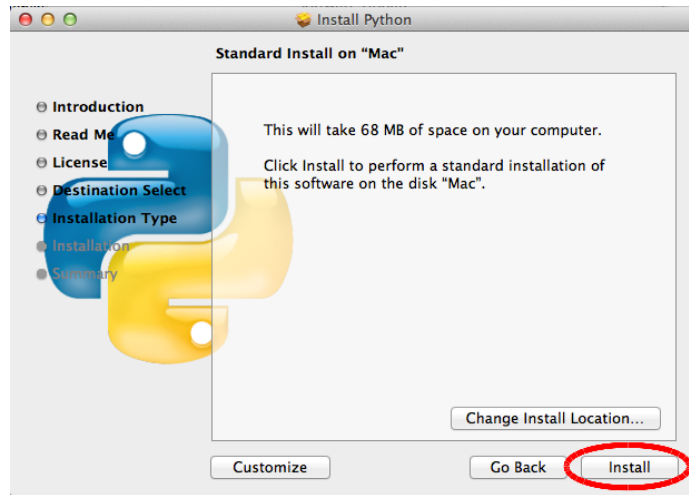
الصورة ٩: اضغط على Continue

٥. اضغط على زر Agree في نافذة الموافقة على الرخصة كما في الصورة رقم ١٠.



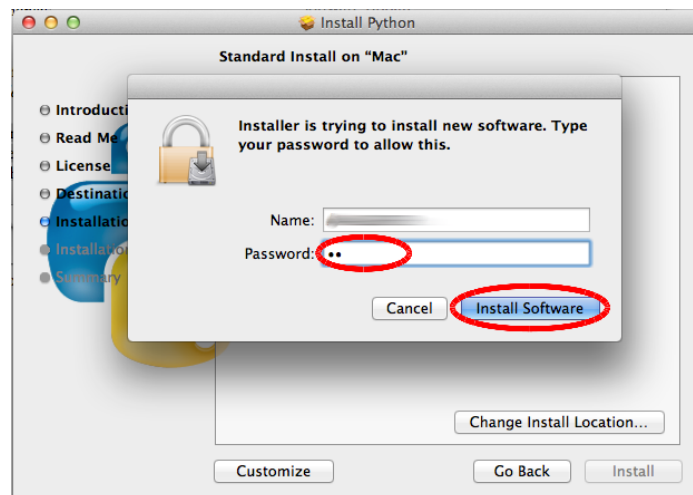
الصورة ١٠: اضغط على Agree

٦. اضغط على زر Install في النافذة التالية كما في الصورة ١١.



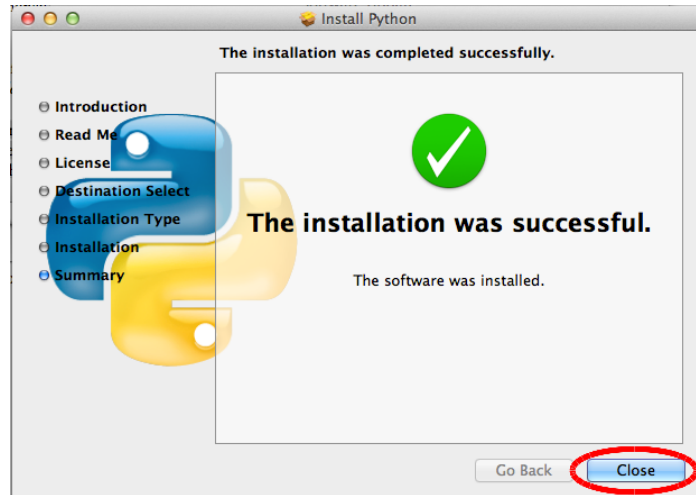
الصورة ١١: اضغط على Install

٧. سيطلب البرنامج منك كلمة السر لمشرف الجهاز ، اكتب كلمة السر و اضغط على زر Install Software كما في الصورة رقم ١٢.



الصورة ١٢: اكتب كلمة السر و اضغط  
Install Software

٨. اضغط على زر Close بعد انتهاء التنصيب كما في الصورة ١٣.



الصورة ١٣: اضغط Close

### ١.٣ تنصيب بايثون في أنظمة ويندوز

بإمكانك تحميل ملف تنصيب بايثون من [هذا الرابط](#) و اتباع الخطوات التالية :

١. فك ضغط الملف الذي قمت بتحميله و ستجد ثلاث ملفات :

○ [python-3.2.3.amd64.msi](#) : استخدم هذا الملف إذا كان نظامك ويندوز ٦٤ بت ، كثير من أنظمة ويندوز ٧ أنظمة ٦٤ و تحتاج هذا الملف.

○ [python-3.2.3.msi](#) : استخدم هذا الملف إذا كان نظامك ويندوز ٣٢ ، معظم أنظمة ويندوز إكس بي و فيستا أنظمة ٣٢ و تحتاج هذا الملف.

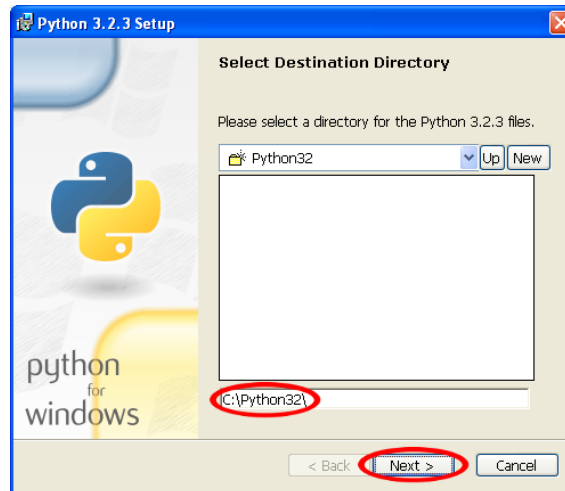
٢. شغل ملف التنصيب المناسب لنظامك بحسب ما تم شرحه في النقطة السابقة.

٣. اضغط زر التالي أو Next في النافذة الأولى كما في الصورة ١٤.



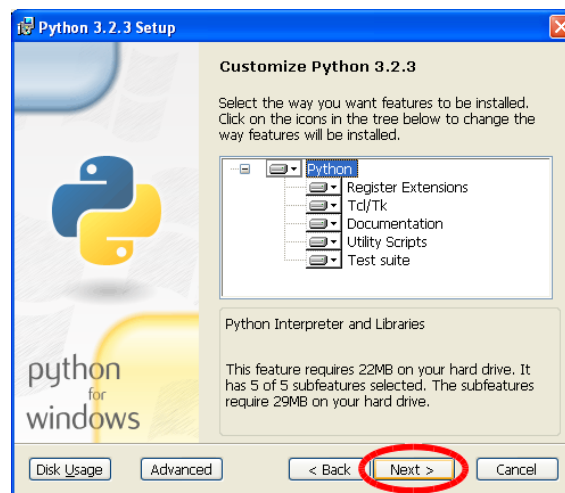
الصورة ١٤: اضغط Next

٤. في النافذة الثانية تأكد من اختيار "C:\Python32\" في مربع الكتابة  
ثم اضغط زر التالي أو Next كما في الصورة ١٥.



الصورة ١٥: تأكد من اختيار C:\Python32\

٥. اضغط زر التالي أو Next في النافذة التالية كما في الصورة ١٦.



الصورة ١٦: اضغط Next

٦. انتظر حتى ينتهي تنصيب بايثون ثم اضغط زر إنهاء أو Finish كما في الصورة ١٧.



الصورة ١٧: اضغط Finish

## ١.٤ تعلمنا في هذا الدرس

- تنصيب بايثون.

هذا الدرس كان تمهيدا لبداية تعلمنا للبرمجة ، في الدرس القادم سنقوم بتنصيب برنامج يساعدنا في كتابة برامج بايثون.



## الدرس الثاني

تنصيب برنامج جي إدت



نقوم بكتابة برامج بايثون باستخدام محرر نصوص مثل برنامج المفكرة ، برنامج المفكرة -على أنه يستطيع كتابة البرامج- يفتقر لميزات تسهل علينا كتابة البرامج كثيرا ، بعض محررات النصوص تجعل كتابة البرامج أسهل من غيرها ، بعضها يساعد في كتابة لغة محددة بينما يساعدنا البعض الآخر في كتابة أكثر من لغة ، يمكنك استخدام أي محرر نصوص تحب لكننا في هذا الدرس نقترح عليك استخدام برنامج gedit الذي يساعدنا في كتابة برامج بمختلف اللغات و منها لغة بايثون يأتي هذا الدرس مع ملفين :

- [gedit-3.2.6.dmg](#) : ملف تنصيب gedit لأنظمة ماك ، بإمكانك تحميله من [هذا الرابط](#).
- [gedit-setup-2.30.1-1.exe](#) : ملف تنصيب gedit لأنظمة ويندوز ، بإمكانك تحميله من [هذا الرابط](#).

## ٢.١ تنصيب المحرر في أنظمة أوبونتو

برنامج gedit يأتي منصبا بشكل افتراضي في أنظمة أوبونتو و لا تحتاج إلى تنصيبه ، بإمكانك البحث عنه بعبارة gedit أو Text Editor ، إذا كنت تستخدم إصدار أوبونتو قديما فهي قوائم للبرامج ستجده في قائمة Accessories باسم Text Editor.

## ٢.٢ تنصيب المحرر في أنظمة ماله

حمل ملف تنصيب ماك و شغله ، ستظهر لك نافذة فيها أيقونتان ، انقر على الأيقونة اليسرى ذات اللون الأحمر و اسحبها بالفأرة إلى أيقونة المجلد التي على يمين النافذة كما في الصورة رقم ١ ، انتظر حتى تنتهي عملية التنصيب.

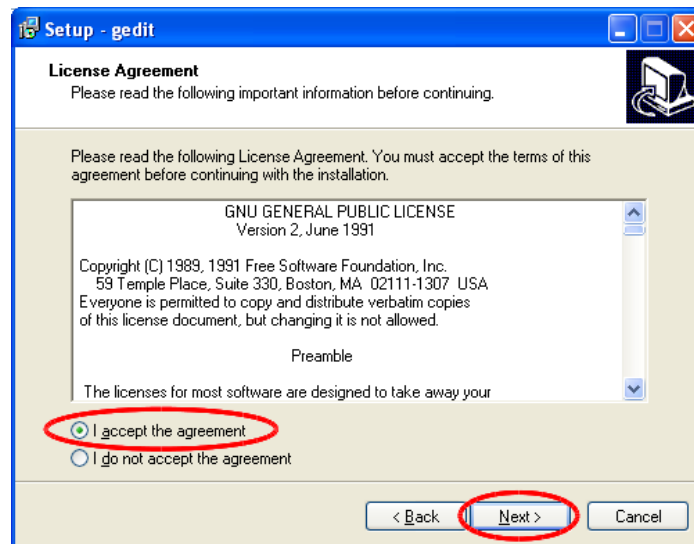


الصورة ١٨: اسحب الأيقونة اليسرى إلى أيقونة المجلد اليمنى

### ٢.٣ تنصيب المحرر في أنظمة ويندوز

حمل ملف تنصيب ويندوز و شغله ثم اتبع الخطوات التالية :

١. اضغط "التالي" أو "Next" في النافذة الأولى.
٢. حدد الخيار "I accept the agreement" ثم اضغط "التالي" أو "Next" كما في الصورة رقم ٢.



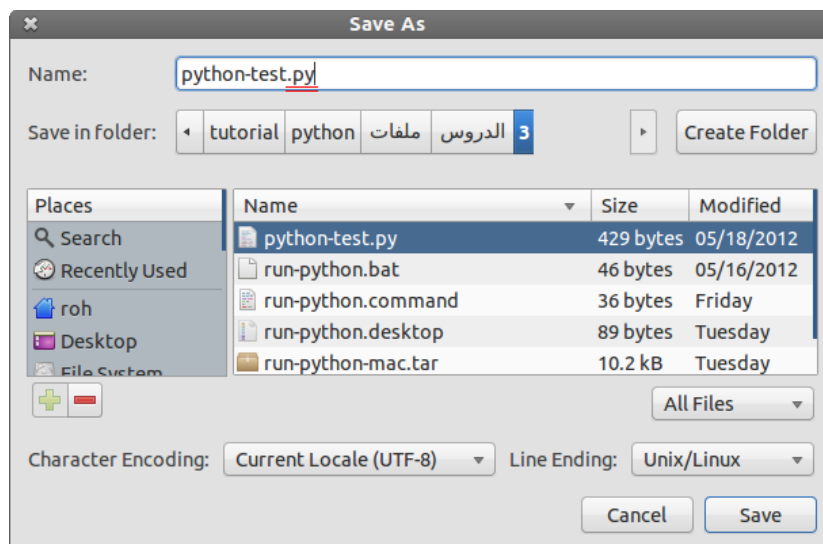
الصورة ١٩: حدد I accept the agreement ثم اضغط  
next

٣. استمر بالضغط على "التالي" أو "Next" في كل النوافذ التالية حتى تصل النافذة الأخيرة ثم اضغط الخيار "إنهاء" أو "Finish".

## ٢.٤ استخدام المحرر

برنامج gedit مثل أي محرر نصوص آخر يستطيع كتابة ملف نصي و حفظ الملف لكننا في هذا القسم نريد التنبيه على بعض النقاط التي تساعدنا في الدروس القادمة و هي :

١. قم بالحفظ الملف أولاً قبل أن تبدأ بكتابة أي شيء.
٢. حين تحفظ الملف تأكد اجعل اسم الملف الذي تريد حفظه ينتهي بامتداد py كما في الصورة رقم ٣.



الصورة ٢٠: احفظ الملف باسم ينتهي بالامتداد py.

الخطوات السابقة تجعل البرنامج يلون الكلمات التي لها معان في لغة بايثون مما يسهل علينا قراءة البرامج ، ستلاحظ هذا حين نقوم بكتابة برامج بايثون.

## ٢.٥ تعلمنا في هذا الدرس

- تنصيب برنامج gedit.
  - ملاحظات عن استخدام المحرر لكتابة برامج بايثون.
- في الدرس القادم سنتعلم تشغيل مفسر بايثون و كتابة الأوامر عليه و أيضا تشغيل البرامج المكتوبة بلغة بايثون. كيفية تشغيل ملفات تحوي أوامر بايثون.



## الدرس الثالث

تشغيل بايثون

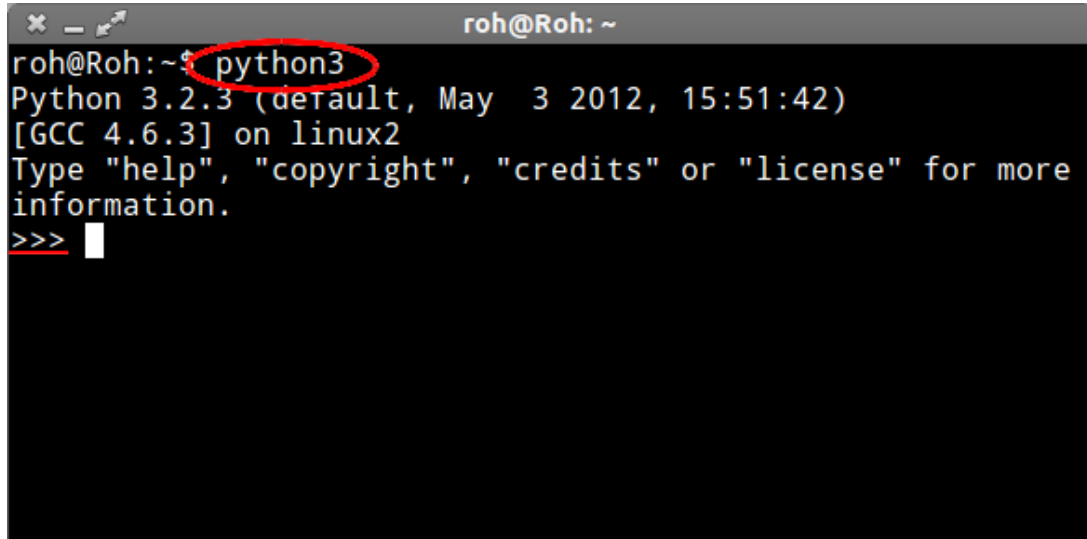
بعد تنصيب بايثون ننتقل في هذا الدرس إلى تشغيل بايثون ، هذا الدرس يوضح فكرة بايثون و طريقة استخدامها و البرمجة بها ، نشرح في هذا الدرس نقطتين رئيسيتين : تشغيل أوامر بايثون عن طريق المفسر و تشغيل أوامر بايثون من ملف ، يأتي هذا الدرس مع عدة ملفات :

- python-test.py : برنامج مكتوب بلغة بايثون سنقوم بتشغيله في هذا الدرس ، بإمكانك تحميله من هذا الرابط.
- run-python-ubuntu.tar : قم بتحميل هذا الملف إذا كنت تستخدم نظام أوبونتو من هذا الرابط ، قم بفك الملف بعد تحميله و ستجد أنه يحوي ملفا اسمه run-python.desktop ، احتفظ به لأننا سنستخدمه باستمرار ابتداء من هذا الدرس.
- run-python-mac.tar : قم بتحميل هذا الملف إذا كنت تستخدم نظام ماك من هذا الرابط ، قم بفك الملف بعد تحميله و ستجد أنه يحوي ملفا اسمه run-python.command ، احتفظ به لأننا سنستخدمه باستمرار ابتداء من هذا الدرس.
- run-python.bat : قم بتحميل هذا الملف إذا كنت تستخدم نظام ويندوز من هذا الرابط ، احتفظ به لأننا سنستخدمه باستمرار ابتداء من هذا الدرس.

### ٣.١ تشغيل مفسر بايثون

١. قم بتشغيل ملف run-python الذي قمت بتحميله لنظامك و سيتم فتح طرفية لكتابة الأوامر.

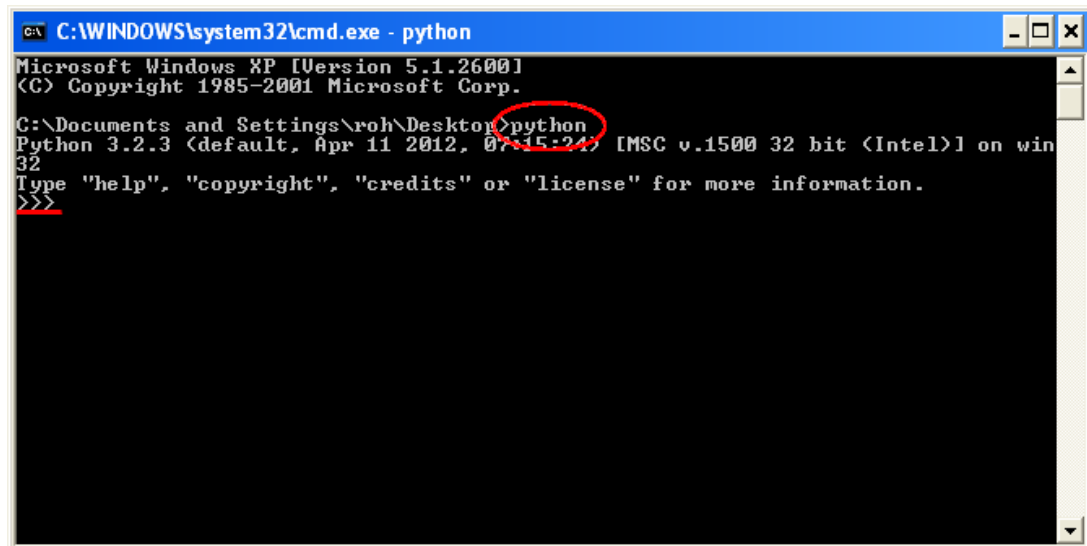
٢. إذا كنت تستخدم نظام أوبونتو أو ماك اكتب الأمر python3 و سيتم تشغيل المفسر، إذا اشتغل المفسر ستجد في بداية السطر الأخير ثلاث علامات أكبر من >>> كما في الصورة رقم ١.



```
roh@Roh: ~  
roh@Roh:~$ python3  
Python 3.2.3 (default, May 3 2012, 15:51:42)  
[GCC 4.6.3] on linux2  
Type "help", "copyright", "credits" or "license" for more  
information.  
>>>
```

الصورة ٢١: تشغيل المفسر في أنظمة أوبونتو و ماك

٣. إذا كنت تستخدم نظام ويندوز اكتب الأمر python و سيتم تشغيل المفسر، إذا اشتغل المفسر ستجد في السطر الأخير ثلاث علامات أكبر من >>> كما في الصورة رقم ٢.

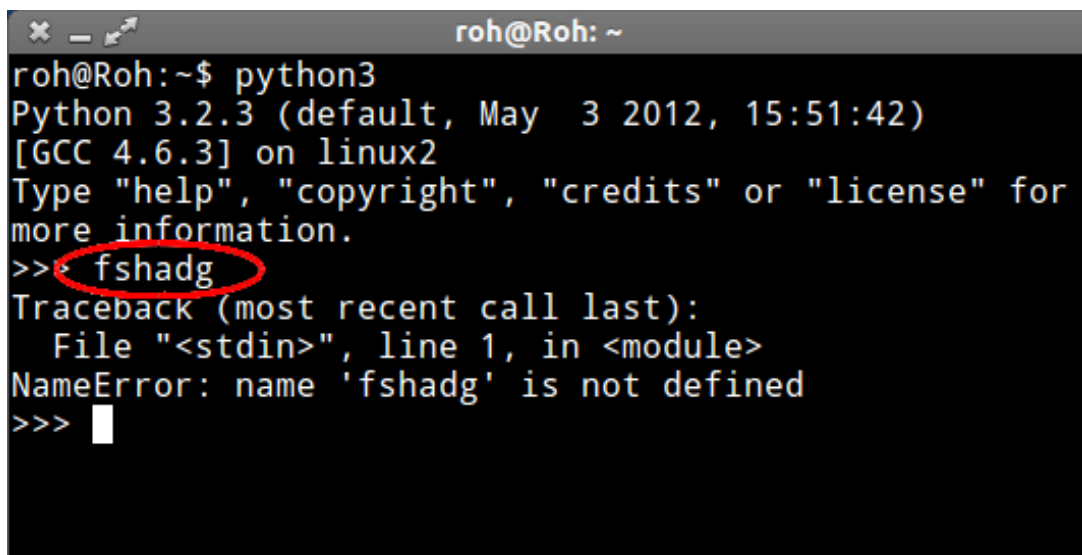


```
C:\WINDOWS\system32\cmd.exe - python  
Microsoft Windows XP [Version 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
C:\Documents and Settings\roh\Desktop>python  
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win  
32  
Type "help", "copyright", "credits" or "license" for more information.  
>>>
```

الصورة ٢٢: تشغيل المفسر في نظام ويندوز

## ٣.٢ تشغيل أوامر بايثون عن طريق المفسر

المفسر يتيح لنا كتابة أوامر لبايثون و برنامج بايثون يقوم بتنفيذها ،  
شغل المفسر و اكتب أي شيء ثم اضغط زر الإدخال Enter كما في الصورة  
رقم ٣.



```
roh@Roh: ~  
roh@Roh:~$ python3  
Python 3.2.3 (default, May 3 2012, 15:51:42)  
[GCC 4.6.3] on linux2  
Type "help", "copyright", "credits" or "license" for  
more information.  
>>> fshadg  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'fshadg' is not defined  
>>> 
```

الصورة ٢٣: كتابة أمر للمفسر

بعد كتابتنا للأمر السابق تم تنفيذ الأمر ، ظهرت لنا رسالة خطأ لأن  
العبارة التي كتبناها خاطئة و لا معنى لها في لغة بايثون ، خلال الدروس  
سنتعلم عددا من الأوامر في لغة بايثون تمكنا من القيام بالعديد من المهام.

## ٣.٣ تشغيل أوامر بايثون من ملف

في القسم السابق قمنا بتشغيل المفسر و كتابة أمر ، كثيرا ما نحتاج  
تنفيذ العديد من الأوامر و يصعب علينا كتابتها واحدا واحدا في المفسر ،  
يقوم المبرمجون عادة بكتابة كل الأوامر في ملف ثم يتم تشغيل كل الأوامر  
الموجودة في الملف ، في هذا القسم من الدرس سنقوم بتشغيل الملف  
[python-test.py](#) و هو برنامج بايثون ، قبل القيام بتشغيل الملف تأكد أنه  
في نفس المجلد الذي فيه الملف [run-python](#) الذي قمنا بتحميله لفتح  
الطرفية ، افتح الطرفية باستخدام ملف [run-python](#) و اكتب أمر تشغيل



بايثون (python3) إذا كنت تستخدم أوبونتو أو ماك و python إذا كنت تستخدم ويندوز) بعده مسافة ثم اسم الملف الذي تريد تشغيله ، الملف الذي نريد تشغيله هو ***python-test.py*** لذلك كتبنا اسمه بعد أمر بايثون كما في الصورة رقم ٤ ، البرنامج يرسم كلمة "بايثون" في الطرفية ، هذا برنامج بايثون بسيط جدا و خلال الدروس سنتعلم كتابة برامج تقوم بمهام أكثر.

```

roh@Roh:~$ python3 python-test.py
          .Q
        a^a
      -W-W`
a
-W`
]f <mWQ. :# )E d( jf
]f =@' Q. =# =E d( ]f
]f jf Q. :m =# d( jf
_/ ]f j[ Q =# )E d( jf
j[ ]f ]6. Q :W =E d( jf
j[ ]f 4QmQQQQQQWQQE dQQQQ[
3[ j[ Q
]6 _m' _ . <@ :Q.Q. j[
9QQ@ ( )WQD' ! ? ''
roh@Roh:~$

```

الصورة ٢٤: تشغيل ملف أوامر بايثون

## ٣.٤ تعلمنا في هذا الدرس

- تشغيل أوامر بايثون عن طريق المفسر.
- تشغيل أوامر بايثون من ملف.

هذا الدرس كان تمهيدا لبداية تعلمنا للبرمجة ، في الدرس القادم سنتعلم تنفيذ العمليات الحسابية التي تشكل أمرا أساسيا في أي لغة برمجة و منها بالتأكيد لغة بايثون.



## الدرس الرابع

العمليات الحسابية

البرنامج عبارة عن عدد من الأوامر يتم تنفيذها و لتعلم البرمجة يجب علينا تعلم هذه الأوامر ، ابتداء من هذا الدرس سنبدأ البرمجة بلغة بايثون و تعلم عدد من الأوامر في كل درس ، نتعلم في هذا الدرس عمليات الحسابية مهمة حتى في لغات البرمجة الأخرى ، سنتطرق في الدرس لست عمليات حسابية بعضها مشهور و هي الجمع و الطرح و الضرب و القسمة ، البعض الآخر من العمليات الحسابية قد لا يكون مشهورا و هي القسمة الصحيحة و باقي القسمة ، هناك عمليات أخرى في بايثون لكننا لن نتطرق لها في هذا الدرس و قد نتعلمها و نستخدمها في دروس لاحقة.

## ٤.١ الجمع و الطرح والضرب و القسمة

شغل مفسر بايثون كما تم شرحه في الدرس السابق ، اكتب الأمر التالي ثم اضغط زر الإدخال :

```
>>> 7+4
```

إذا أدخلت السطر السابق سيكتب المفسر في السطر الذي يليه العدد ١١ ، إذا أدخلنا أي عملية حسابية يقوم المفسر بتنفيذها و طباعة الناتج ، جرب جمع أعداد أكثر في سطر واحد ، مثلا اكتب :

```
>>> 8+4+12+3
```

سيقوم المفسر بطباعة نتيجة جمع الأعداد ، هناك العديد من العمليات في لغة بايثون نعطي هنا بعضها مع رمز كل عملية و مثال عليها و ناتجها :

• عملية الجمع ( + ) :

```
>>> 7 + 4
11
```

• عملية الطرح ( - ) :

```
>>> 7 - 4
3
```

• عملية الضرب ( \* ) :

```
>>> 7 * 4
28
```

• عملية القسمة (/) :

```
>>> 7 / 4
1.75
```

لاحظ أننا نستخدم رموزا للضرب و القسمة تختلف عن الرموز المستخدمة عادة في الرياضيات ، في الضرب نستخدم رمز النجمة \* بدل الرمز × و في القسمة نستخدم الخط المائل / بدل استخدام الرمز ÷ لكن معنى العملية لا يتغير فالضرب و القسمة في بايثون و الرياضيات لهما نفس المعنى تماما.

## ٤.٢ القسمة الصحيحة و باقي القسمة

هناك عمليتان حسابيتان مفيدتان في البرمجة إضافة إلى العمليات السابقة و هما القسمة الصحيحة و باقي القسمة.

• القسمة الصحيحة : هي قسمة عددين مع تجاهل أي قيمة بعد الفاصلة ، إذا قسمنا مثلا ٧ على ٤ في القسمة العادية نجد أن الناتج هو ١.٧٥ ، في القسمة الصحيحة يكون الناتج ١ مع تجاهل كل القيم بعد الفاصلة ، رمز القسمة الصحيحة هو خطان مائلان // ، الأمر التالي مثال على القسمة الصحيحة و بعده يظهر ناتج العملية :

```
>>> 7 // 4
1
```

• باقي القسمة : هذه العملية تعطي باقي قسمة العدد الأول على العدد الثاني ، نستخدم رمز النسبة المئوية % للقيام بعملية باقي القسمة ، مثلا ٧ % ٤ يساوي ٣ لأن العدد ٣ هو باقي قسمة ٧ على ٤ ، الأمر التالي مثال على عملية باقي القسمة و بعده يظهر ناتج العملية :

```
>>> 7 % 4
3
```

### ٤.٣ أولوية تنفيذ العمليات الحسابية

بإمكاننا كتابة أكثر من عملية في أمر واحد ، مثلا نستطيع كتابة الأمر التالي :

```
>>> 3 % 2 + 5 * 2
```

يتم تنفيذ العمليات الحسابية بترتيب معين فبعض العمليات لها أولوية أعلى من الأخرى و يتم تنفيذها أولا ، في هذا الدرس سنذكر ترتيب العمليات الستة التي تم ذكرها في هذا الدرس لكن هناك عمليات أخرى قد يكون لها أولوية أعلى أو أقل من العمليات المذكورة ، أولوية تنفيذ العمليات التي تطرقنا لها في الدرس كما يلي :

١. الضرب و القسمة و القسمة الصحيحة و باقي القسمة.

٢. الجمع و الطرح.

يتم تنفيذ كل العمليات من المستوى الأول قبل تنفيذ عمليتي الجمع و الطرح اللتين في المستوى الثاني ، في السطر السابق نجد ثلاث عمليات و هي باقي القسمة و الجمع و الضرب ، يتم تنفيذ باقي القسمة و الضرب أولا قبل تنفيذ الجمع ، بعد حساب ناتجي باقي القسمة و الضرب يتم بعد ذلك تنفيذ الجمع بين الناتجين.

إذا وجدت أكثر من عملية بنفس مستوى التنفيذ يتم تنفيذ العمليات من اليسار إلى اليمين ، مثلا :

```
2 * 7 // 3
```

لأن الأمر السابق يحوي عمليتين من نفس المستوى و هما الضرب و القسمة الصحيحة يتم تنفيذ العملية التي على اليسار أولا و هي ضرب ٢ × ٧ و يتم حساب النتيجة ، بعد ذلك تتم عملية القسمة الصحيحة بين نتيجة الضرب و العدد ٣.

بإمكاننا تجميع بعض العمليات بين قوسين ليتم تنفيذها قبل العمليات

الأخرى بغض النظر عن أولوية تنفيذها ، مثلا :

$$5 * ( 2 + 1 )$$

مع أن عملية الضرب تتم قبل عملية الجمع إلا أن عملية الجمع في الأمر السابق تتم أولا لأنها داخل قوسين.

## ٤.٤ تعلمنا في هذا الدرس

• بعض العمليات الحسابية :

- الجمع.
- الطرح.
- الضرب.
- القسمة.
- القسمة الصحيحة.
- باقي القسمة.

• أولوية تنفيذ العمليات الحسابية.

في هذا الدرس تعلمنا الحساب بلغة بايثون و تعاملنا بشكل أساسي مع المفسر ، قمنا بكتابة أوامر صحيحة و مشاهدة ناتجها ، في الدرس القادم سنتعلم أوامر لكتابة جمل على الطرفية و هذا من أبسط أشكال التفاعل مع المستخدم.



# الدرس الخامس

الطباعة

سنتعلم في هذا الدرس معنى الطباعة في البرمجة و نتعرف على أمر الطباعة ، سنستخدم أمر الطباعة في مفسر بايثون ثم نكتب أول برنامج لنا في ملف و سنستخدم فيه أيضا أمر الطباعة.

الطباعة في البرمجة تعني الكتابة على شاشة الأوامر ، هذا مختلف كثيرا عن الطباعة بالطابعة ، إذا سمعت عبارة "طباعة" في أي لغة برمجة فهي تعني الطباعة على شاشة الأوامر.

## ٥.١ أمر الطباعة

شغل مفسر بايثون و اكتب الأمر التالي ثم اضغط زر الإدخال :

```
>>> print ( 'hello world' )
```

كل ما سيفعله المفسر هو كتابة العبارة "hello world" في الطرفية ، تذكر طريقة كتابة الأمر : نكتب print بعدها قوسان و نكتب بينهما ما نريد طباعته مثل الجملة السابقة "hello world" ، بإمكاننا طباعة الأرقام أيضا ، اكتب السطر التالي في مفسر بايثون :

```
>>> print ( 29 )
```

سيقوم المفسر بطباعة العدد ٢٩ ، جرب كتابة أي عدد و ستجد أنه سيقوم بطباعته ، نستطيع أيضا طباعة ناتج العمليات الحسابية ، مثلا :

```
>>> print ( 11 + 7 )
```

سيقوم المفسر بطباعة ناتج العملية الحسابية تماما كما حصل في الدرس الماضي حين قمنا بالعمليات الحسابية ، أمر الطباعة يستطيع أن يطبع أكثر من شيء في السطر الواحد ، خذ هذا المثال :

```
>>> print ( 'I am', 12, 'years old' )
```

في السطر السابق قمنا بطباعة عبارة تخبر أن عمرنا ١٢ سنة ، قمنا بطباعة جملتين و رقم ، بإمكاننا طباعة أي عدد من العبارات و الأعداد و العمليات الحسابية و غيرها بأن نفصل بين كل شيء نطبعه و الذي يليه بفاصلة ، أمر الطباعة يفصل بين كل أمر و آخر بمسافة واحدة ، في مثالنا السابق مثلا



نجد عند الطباعة في شاشة الأوامر أن العبارة "I am" مفصولة بمسافة عن العدد ١٢ و العدد أيضا مفصول بمسافة عن العبارة "years old".

## ٥.٢ النصوص

في القسم السابق طبعنا جملة على الشاشة و حين كتبنا الجملة في بايثون وضعناها بين علامتي اقتباس مفردتين ( ' ) ، كل الكلمات الموجودة بدون علامات اقتباس تفسر على أنها أوامر بايثون و أي عبارة بين علامتي اقتباس تعتبر عبارة نصية و ليست أمرا ، لهذا السبب تجد أننا إذا أردنا طباعة أي جملة نضعها بين علامتي اقتباس ، بإمكاننا أيضا استخدام علامات الاقتباس المزدوجة ( " ) ، نستطيع أن نكتب مثلا :

```
>>> print ( "hello world" )
```

علامات الاقتباس المفردة و المزدوجة لها نفس المعنى تماما في لغة بايثون و بإمكاننا أن نستخدم أي نوع من علامات الاقتباس لكتابة نص معين ، المهم في الأمر أن ننهي النص بنفس علامة الاقتباس التي فتحنا النص بها ، إذا بدأنا النص بعلامة اقتباس مفردة يجب أن ننهيه بعلامة اقتباس مفردة و إذا بدأنا النص بعلامة اقتباس مزدوجة يجب أن ننهيه بعلامة اقتباس مزدوجة ، خلال الدروس سنستخدم علامات الاقتباس المفردة لتحديد النصوص و لن نستخدم علامات الاقتباس المزدوجة.

## ٥.٣ رموز الإفلات

قمنا في القسم السابق بطباعة نص معين بأن كتبناه في أمر الطباعة بين علامتي اقتباس مفردتين ، هناك رموز خاصة تسمى رموز الإفلات لها معان خاصة ، سنتعلم في هذا القسم رمزا واحدا فقط و هو \n ، خذ هذا الأمر مثلا و الجملة التي يطبعها :

```
>>> print ( 'Hello\nI am Mohammad' )
hello
I am Mohammad
```

لاحظ أننا كتبنا رمز الإفلات \n بعد كلمة "Hello" مباشرة ، هذا الرمز يعني سطرًا جديدًا أي أن أمر الطباعة سينزل سطرًا بعد كلمة "Hello" مباشرة ، نستطيع استخدام أمر الطباعة مرتين لطباعة الجمل في أسطر مختلفة دون استخدام رمز الإفلات :

```
>>> print ( 'Hello')
hello
>>> print ( 'I am Mohammad')
I am Mohammad
```

## ٥.٤ كتابة برنامج في ملف

قد تتساءل عن وظيفة أمر الطباعة فنحن نطبع كلامًا نعرفه ، أيضًا قد تتساءل عن فائدة طباعة الأرقام و العمليات الحسابية مع أن المفسر يقوم بطباعتها تلقائيًا ، سنستغل هذا الدرس لنستفيد قليلًا من أمر الطباعة و نتعلم كيفية كتابة برنامج بايثون في ملف بدل أن نكتبه في المفسر ، سنستفيد أكثر من أمر الطباعة في الدروس القادمة.

افتح برنامج محرر النصوص جي إدت لكتابة برنامج بايثون ، الأفضل أن تقوم بحفظ الملف أولاً قبل أن تبدأ الكتابة فيه ، احفظ الملف و سمه بأي اسم لكن تأكد أن الاسم يتكون من حروف أو أرقام انجليزية و ينتهي بـ **"py"** ، تستطيع مثلاً أن تسميه "lesson1.py" ، اكتب في الملف النص التالي :

```
print ( 'hello world')
```

هذا نفس الأمر الذي كتبناه في بداية الدرس ، بإمكانك كتابة أي أوامر أخرى ، احفظ الملف بعد التعديل و شغل الطرفية ، تأكد أن ملف تشغيل الطرفية و ملف بايثون الذي كتبته في نفس المجلد كما تم شرحه في الدرس الثاني.

شغل الطرفية و بعدها شغل الملف كما تم شرحه في الدرس الثاني :

- إذا كنت تستخدم أوبونتو أو ماك : اكتب python3 بعدها مسافة بعدها اسم الملف ، مثلا :

```
python3 lesson1.py
```

- إذا كنت تستخدم ويندوز : اكتب python بعدها مسافة بعدها اسم الملف ، مثلا :

```
python lesson1.py
```

تلاحظ أن البرنامج طبع عبارة "hello world" ثم توقف ، تستطيع كتابة عدة أوامر طباعة تظهر في الطرفية كلما شغلت الملف و لن تظهر الأوامر في الطرفية ، أيضا إذا كتبت عمليات حسابية في الملف فلن يتم طباعة ناتجها كما هو الحال في المفسر بل يجب أن تكتبها في أمر الطباعة لكي تظهر في شاشة الأوامر.

## ٥.٥ تعلمنا في هذا الدرس

- أمر الطباعة : print بعدها قوسان بينهما جملة أو رقم نريد طباعته.
  - رمز الإفلات \n و معناه سطر جديد.
  - كتابة برنامج بايثون في ملف.
- في هذا الدرس تعلمنا الطباعة في لغة بايثون ، سنستفيد أكثر و أكثر من هذا الأمر أكثر في الدروس القادمة ، الدرس القادم يغطي المتغيرات و هي من أهم مبادئ البرمجة.



## الدرس السادس

المتغيرات

تعلّمنا في الدرسين السابقين تنفيذ العمليات الحسابية و طباعة الجمل و الأعداد ، كل الجمل و الأعداد التي تعاملنا معها هي قيم ثابتة لا تتغير ، أحيانا نريد حفظ عدد أو عبارة معينة في البرنامج لنستخدمها لاحقا و هنا تأتي فائدة المتغيرات ، ستلاحظ أن استخدام المتغيرات يشبه جدا استخدام الجمل و الأرقام التي استخدمناها في الدروس السابقة ، لن نستفيد من المتغيرات كثيرا في هذا الدرس لكننا سنستفيد منها أكثر فأكثر خلال الدروس القادمة.

## 1.1 المتغيرات

شغل مفسر بايثون و اكتب الأمر التالي ثم اضغط زر الإدخال :

```
>>> var = 13
```

في الدروس السابقة تعلّمنا بعض الكلمات التي لها معنى في لغة بايثون مثل أمر الطباعة print ، كلمة var ليس لها أي معنى في لغة بايثون لكننا في السطر السابق عرفنا متغيرا و أسميناه var و حددنا أن قيمته ١٣ ، في الأمر السابق أعطينا معنى لكلمة var و نستطيع استخدامها في الأوامر التالية على أنها رقم مثل أي رقم آخر ، في الدروس السابقة قمنا بعمليات على الأرقام و طباعة الأرقام ، نستطيع أيضا أن نفعل هذا مع المتغير var :

```
>>> var
13
>>> var * 2 + 1
27
>>> var * var
169
>>> print ( var )
13
```

نستخدم المتغير لحفظ قيمة معينة ، لاحظ في الأسطر السابقة أننا حفظنا قيمة ١٣ في المتغير و قمنا باستخدام المتغير مثلما نستخدم أي رقم في سواء في الحساب أو الطباعة.

بإمكاننا تغيير قيمة المتغير في أي وقت بتعريفه مرة أخرى كما فعلنا في

المرّة السابقة ، أدخل السطر التالي في المفسر :

```
>>> var = 5
```

في السطر السابق غيرنا قيمة var إلى ٥ ، إذا استخدمنا var بعد هذا السطر فسيأخذ القيمة الجديدة ، المتغير يأخذ قيمة واحدة و بتغيير قيمته تختفي قيمته السابقة و لا يمكننا أن نستخرجها من المتغير بعد ذلك ، قم بطباعة المتغير و شاهد قيمته بنفسك :

```
>>> var  
5
```

بإمكاننا أيضا تغيير قيمة var اعتمادا على قيمته السابقة ، مثلا لو أردنا أن نضاعف قيمة var مرتين نستطيع كتابة الأمر التالي :

```
>>> var = var * 2
```

في السطر السابق جعلنا قيمة var هي قيمت var مضروبة في ٢ ، دائما يتم حساب الطرف الأيمن في الأمر أولا قبل أن تتغير قيمة المتغير ، في السطر السابق مثلا يتم حساب var ضرب ٢ و النتيجة تصبح ١٠ ، بعد الحساب يتم تعيين القيمة للمتغير فتصبح قيمة var الجديدة هي ١٠ ، بإمكانك طباعته لتتأكد من قيمته.

بإمكاننا اختيار أي اسم للمتغير لكن علينا مراعاة بعض الشروط :

- لا يبدأ برقم ، يمكن أن يبدأ بحرف أو رمز سفلي ( \_ ) فقط.
- لا يحوي علامات حسابية أو علامات ترقيم مثل الفاصلة و إشارة الجمع و لا يحوي المسافات ، يمكن أن يحوي حروفا و رمزا سفليا و أرقاما.
- أن لا يكون كلمة لها معنى في لغة بايثون.

## ١.٢ متغيرات النصوص

في القسم السابق أعطينا مثالا لمتغير أعطيناه قيمة عددية ، الأعداد ليست القيم الوحيدة التي يمكن أن نخزنها في متغير بل نستطيع أن نحفظ الجمل و النصوص أيضا ، خذ هذا المثال :

```
>>> name = 'Mohammad'
```

في الأمر السابق عرفنا المتغير name و خزنا فيه جملة نصية و ليس رقما ، يمكننا أن نعامل هذا المتغير مثل أي عبارة نصية أخرى ، بإمكاننا طباعته مثلا :

```
>>> print ( name )
Mohammad
```

## ١.٣ جمع النصوص

درسنا عدة عمليات و قمنا باستخدامها مع الأعداد ، بعض العمليات يمكن استخدامها أيضا مع النصوص و منها عملية الجمع ، جمع الأعداد معروف لكن جمع النصوص يعني فقط إنتاج نص مكون من النصين المجموعين ، مثلا :

```
>>> 'hello' + 'world'
'helloworld'
```

لاحظ أن جمع العبارتين تم بدون الفصل بينهما بأي مسافة ، للفصل بين الكلمتين علينا أن نضع المسافة في عملية الجمع ، في الأمر التالي مثلا أضفنا مسافة بعد كلمة hello :

```
>>> 'hello ' + 'world'
'hello world'
```

بما أننا نستطيع أن نعامل المتغير الذي يحوي نصا مثل أي متغير فإمكاننا جمع المتغير name مع النصوص :

```
>>> 'hello ' + name + ', how are you ?'
'hello Mohammad, how are you ?'
```

لكن انتبه ! عملية الجمع تحصل بين النصوص مع بعضها و مع الأرقام مع بعضها لكن لا يمكن جمع أرقام و نصوص معا ، إذا قمت بجمع عدد و نص معا ستظهر لك رسائل أخطاء ، بإمكانك تجربة هذا :

```
>>> name + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

## 1.4 تعلمنا في هذا الدرس

- المتغيرات : تستخدم لحفظ قيمة معينة.
  - متغيرات النصوص.
  - جمع النصوص : جمع النصوص يعطي نصا يحوي كل النصوص المجموعة.
- في الدرس القادم بإذن الله سنتعلم التفاعل مع المستخدم ، سنكتب برنامجا يطلب من المستخدم إدخال بيانات عن نفسه و نحفظها في البرنامج و نقوم بطاعتها و استخدامها.





## الدرس السابع

الإدخال

قمنا في كل الدروس السابقة بطباعة قيم كتبناها في البرنامج ، كثير من البرامج تتفاعل مع المستخدم بمعنى أنها تطلب منه إدخال قيم معينة ، في برامج التصفح على سبيل المثال يكتب المستخدم رابط الموقع الذي يريد زيارته ، سنتعلم في هذا الدرس كيف نجعل البرنامج يطلب عبارة من المستخدم و يقوم المستخدم بكتبتها عن طريق لوحة المفاتيح ، مثلا قد يطلب البرنامج من المستخدم أن يكتب اسمه.

## ١.٧ الإدخال

بإمكاننا طلب عبارة من المستخدم عن طريق أمر الإدخال `input ()` ، على سبيل المثال :

```
>>> var = input ()
```

الأمر السابق يطلب من المستخدم كتابة عبارة بلوحة المفاتيح و يخزنها على شكل نص في المتغير `var` ، إذا كتب المستخدم اسمه مثلا فإن قيمة `var` تصبح اسم المستخدم و بإمكاننا طباعة المتغير `var` للتأكد من ذلك :

```
>>> var = input ()
mohammad
>>> print ( var )
mohammad
```

بإمكاننا حين نستخدم أمر `input` أن نكتب جملة نصية بين الأقواس لتظهر عند تنفيذ الأمر ، هذا مفيد حين نريد ذكر رسالة معينة قبل طلب الإدخال من المستخدم ، مثلا قد نريد أن نظهر رسالة "ما هو اسمك : " و بعدها ننتظر من المستخدم أن يكتب اسمه ، للقيام بهذا نكتب الأمر التالي :

```
>>> var = input ( 'what is your name : ' )
what is your name : mohammad
```

في المثال السابق أظهرنا رسالة طلب الاسم للمستخدم و افترضنا أنه أدخل "mohammad" فأصبحت الكلمة "mohammad" مخزنة في المتغير `var`.

## ٧.٢ إدخال الأعداد

الأمر input يقوم بإدخال الجمل فقط ، نستطيع أن ندخل أرقاما لكننا لا نستطيع أن نقوم بعمليات حسابية بها كالضرب و القسمة ، نستطيع فقط أن نجمعها مع الجمل الأخرى مثل أي عبارة نصية ، إذا أردنا إدخال عدد نحتاج استخدام الأمر ( int ) لإدخال أعداد صحيحة بدون فواصل عشرية أو الأمر ( float ) إذا أردنا إدخال عدد بفاصلة عشرية ، نكتب الأمر int أو float و نكتب داخل الأقواس الأمر input في المثال التالي سندخل عددا صحيحا و نطبع نتيجة جمعه مع ١ :

```
>>> var = int ( input () )
33
>>> print ( var + 1 )
34
```

إذا لم نستخدم الأمر int حاولنا طباعة نتيجة جمع المتغير مع ١ ستظهر لنا رسالة خطأ :

```
>>> var = input ()
33
>>> print ( var + 1 )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

## ٧.٣ تعلمنا في هذا الدرس

- الإدخال : نستخدم الأمر input لطلب الإدخال من المستخدم.
  - إدخال الأعداد : نستخدم الأمر int أو float مع الأمر input لإدخال الأعداد.
- في الدرس القادم سنتعلم التعامل مع الملفات ، سنقرأ محتويات ملف و نقوم بطباعتها و سنقوم أيضا بكتابة عبارات نصية داخل ملف.



## الدرس الثامن

التعامل مع الملفات

معظم البرامج التي نتعامل معها يوميا تتعامل مع الملفات ، في هذا الدرس سنتعلم كيفية فتح ملفات في لغة بايثون ، سنتعلم كيف نقرأ محتويات الملف و كيف نكتب في الملف.

## ٨.١ فائدة الملفات

تعلمنا في الدروس السابقة كيفية إدخال قيمة من لوحة المفاتيح من قبل المستخدم ، إذا انتهى البرنامج تختفي هذه القيمة و لا يمكن أن نعرفها ، تخيل مثلا أن المستخدم أدخل اسمه في البرنامج ، إذا انتهى البرنامج تختفي هذه القيمة و لا يمكن استرجاعها ، أحيانا نريد أن نسترجع بعض البيانات في وقت لاحق حتى بعد انتهاء البرنامج ، هنا تأتي فائدة الملفات ، الملفات تحفظ البيانات من البرنامج و يمكن بعد ذلك قراءتها حتى بعد إغلاق البرنامج أو إطفاء الحاسب.

## ٨.٢ فتح ملف

لفتح ملف نستخدم الأمر `open()` ، الصيغة العامة لأمر `open` هي :

```
open ( 'file-name' )
```

نكتب بين القوسين عدة أمور سنذكر في هذا الدرس اثنين منها :

١. اسم الملف ، هذا القسم إجباري و يجب أن نكتب اسم الملف الذي نريد فتحه ، لو أردنا مثلا فتح ملف اسمه "example.txt" نكتب ما يلي :

```
>>> open ( 'example.txt' )
```

بالتأكيد نريد أن نتعامل مع أي ملف نقوم بفتحه ، لا يمكن التعامل مع الملف الذي نفتحه إلا إذا حفظناه في متغير لذلك عادة نقوم عادة باستخدام أمر `open` في تعريف متغير ، مثلا :

```
>>> var = open ( 'example.txt' )
```

بإمكاننا بعدها استخدام المتغير `var` لقراءة الملف و التعامل معه ، سيتم شرح ذلك في الأقسام التالية.

٢. وضع فتح الملف ، هناك وضعان رئيسيان عند فتح الملف : وضع الكتابة و وضع القراءة ، عند فتح ملف بوضع الكتابة فيمكننا فقط الكتابة في الملف و لا يمكننا القراءة منه ، أما إذا فتحنا ملفا بوضع القراءة يمكننا فقط القراءة من الملف و لا يمكننا الكتابة فيه ، لتحديد الوضع الذي نريده نضع فاصلة بعد اسم الملف و نكتب بعدها حرفا يحدد الوضع :

◦ إذا أردنا تحديد وضع الكتابة نضع الحرف "w" ، مثلا :

```
>>> var = open ( 'example.txt', 'w' )
```

◦ إذا أردنا تحديد وضع القراءة نضع الحرف "r" ، مثلا :

```
>>> var = open ( 'example.txt', 'r' )
```

تذكر أن تكتب الوضع بين علامتي اقتباس فلا يجوز أن نكتب الحرف r أو w بدون علامات اقتباس.

إذا لم نحدد وضعاً يتم فتح الملف بوضع القراءة لذلك لا فرق أبداً بين فتح الملف بوضع القراءة و بين فتحه بدون تحديد الوضع ، الأمران التاليان متماثلان تماماً :

```
>>> var = open ( 'example.txt', 'r' )
>>> var = open ( 'example.txt' )
```

## ٨.٣ الكتابة في ملف

للكتابة في ملف علينا أولاً فتح الملف بوضع الكتابة كما تم شرحه في القسم السابق ، افتح مفسر بايثون و اكتب الأمر التالي :

```
>>> wf = open ( 'example.txt', 'w' )
```

هكذا عرفنا المتغير wf و نستطيع أن نستخدمه للكتابة في الملف example.txt ، للكتابة في الملف نستخدم الأمر **write()** و صيغته كالتالي :

```
>>> wf.write ( 'text' )
```

نكتب اسم المتغير بعده نقطة بعدها كلمة **write** و بعدها قوسان ، نكتب بين القوسين العبارة التي نريد كتابتها كعبارة نصية بين علامتي اقتباس ،

نستطيع استخدام أمر write أكثر من مرة ، بإمكاننا مثلا كتابة ما يلي :

```
>>> wf.write ( 'hello' )  
>>> wf.write ( 'i am mohammad' )
```

الأوامر الثلاثة السابقة تكتب العبارة التالية داخل الملف :

```
hello i am mohammad
```

تلاحظ من العبارة الموجودة في الملف أن أمر write لا يفصل بين الجمل بمسافات و لا يكتب كل جملة في سطر ، إذا أردنا أن نفصل بين الجمل بمسافات أو نقط أو ما شاكل علينا أن نكتب المسافات و الأسطر في الجملة التي نكتبها في الملف ، الأوامر التالية هي نفس الأوامر السابقة لكنها تزيد سطرا جديدا في كل جملة يتم كتابتها :

```
>>> wf.write ( 'hello\n' )  
>>> wf.write ( 'i am mohammad\n' )
```

الأسطر السابقة تكتب الآتي في الملف :

```
hello  
i am mohammad
```

بعد كتابة كل ما نريده في الملف علينا أن نغلق الملف بأمر () close ، صيغته مثل صيغة write لكننا نضع القوسين بدون أي شيء بينهما ، اكتب الأمر التالي بعد الانتهاء من كتابة العبارات السابقة في الملف :

```
wf.close ()
```

بعد إغلاق الملف لا يمكننا الكتابة فيه و استخدام أمر write بعد الإغلاق يسبب أخطاء في البرنامج.

## ٨.٤ القراءة من ملف

للقراءة من ملف علينا أولا فتح الملف بوضع القراءة كما تم شرحه في القسم الأول ، في القسم السابق كتبنا ملفا اسمه example.txt ، لفتح الملف و نقرأ محتوياته ، اكتب الأمر التالي في مفسر بايثون :

```
>>> rf = open ( 'example.txt' )
```

قمنا بتعريف المتغير rf و نستطيع الآن استخدامه للقراءة من الملف ، أمر

القراءة من ملف هو `readline()` ، هذا الأمر يقرأ سطرا واحدا من الملف ،  
صيغة الأمر مثل صيغة أمر `write()` لكننا لا نكتب شيئا بين الأقواس ،  
بإمكاننا استخدام الأمر `readline` كما يلي :

```
>>> text = rf.readline ()
```

في الأمر السابق عرفنا متغيرا اسمه `text` و خزنا فيه سطرا واحدا من الملف ،  
بإمكاننا بعد ذلك طباعة المتغير `text` لنعرف الجملة المكتوبة :

```
>>> print ( text )  
helloi am mohammadhello
```

العبارات السابقة كانت عبارات كتبناها دون أن نكتب سطرا جديدا ، إذا  
استخدمنا الأمر `readline` مرة أخرى سيعطينا اسطر الثاني من الملف :

```
>>> text = rf.readline ()  
>>> print ( text )  
i am mohammad
```

إذا كانت هناك أسطر أكثر فإن أمر `readline` يقرأ لنا سطرا جديدا في كل مرة  
نستخدمه ، إذا وصلنا لنهاية الملف و استخدمنا الأمر `readline` فإن الأمر  
يعطينا نصا فارغا ، في الأوامر السابقة قمنا بقراءة كل الأسطر من الملف  
الذي فتحناه ، لنجرب استخدام أمر القراءة مرة أخرى :

```
>>> text = rf.readline ()  
>>> rf = print ( text )
```

لاحظ أن أمر الطباعة لم يطبع شيئا على الإطلاق لأننا وصلنا لنهاية الملف.  
بعد الانتهاء من القراءة من الملف علينا أن نتذكر إغلاقه بالأمر `close()` تماما  
مثلا فعلنا حين أغلقنا الملف بعد الكتابة :

```
>>> rf.close ()
```

بعد إغلاق الملف لا يمكننا القراءة منه و استخدام أمر `readline` بعد الإغلاق  
يسبب أخطاء في البرنامج.



## ٨.٥ تعاملنا في هذا الدرس

- فتح الملف : نستخدم الأمر ( open ) :

◦ وضع الكتابة :

```
var = open ( 'example.txt', 'w' )
```

◦ وضع القراءة :

```
var = open ( 'example.txt', 'r' )
```

- الكتابة في ملف : نستخدم الأمر ( write ) :

```
var.write ( 'text' )
```

- القراءة من ملف : نستخدم الأمر ( readline ) :

```
text = var.readline ( )
```

- إغلاق الملف : نستخدم الأمر ( close ) :

```
var.close ( )
```

في الدرس القادم سنتعلم كيفية التحكم في البرنامج بشكل أكبر و كتابة أوامر يتم تنفيذها في حالات معينة فقط بينما يتم تجاهلها في حالات أخرى.



## الدرس التاسع

الجمل الشرطية ١

في هذا الدرس سنتعلم التحكم بشكل أكبر في البرنامج ، سنقوم بتحديد أوامر يقوم بها البرنامج في حالات معينة و لا يقوم بها في حالات أخرى ، مثلا قد نريد من البرنامج أن يطلب درجة ، إذا كانت الدرجة أكبر من ٩٠ نخبر الطالب أنه ممتاز ، إذا كانت الدرجة أقل من ذلك لا نطبع شيئا.

## ٩.١ القيم المنطقية

في البرمجة توجد عدة أنواع للقيم ، تعلمنا في الدروس السابقة القيم العددية و هي الأعداد مثل ١ و ٢ ، تعلمنا أيضا القيم النصية و هي الجمل و كنا نضعها بين علامات اقتباس و من أمثلتها عبارة "hello world".

بالإضافة للقيم العددية و النصية هناك نوع آخر من القيم و هي القيم المنطقية ، القيم المنطقية يمكن أن تكون واحدة من اثنتين : إما قيمة صواب أو خطأ ، هذا النوع من القيم يصف تعبيراً ما بأنه صحيح أو غير صحيح ، مثلا عبارة ١٥ أكبر من ٩ ، العبارة السابقة صحيحة إذا قيمتها المنطقية صائبة ، تعبير آخر يمكن أن نستخدمه هو أن ١٥ أصغر من ٩ ، العبارة السابقة ليست صحيحة فالعدد ١٥ ليس أصغر من العدد ٩ ، إذا فالقيمة المنطقية للعبارة خاطئة.

## ٩.٢ عمليات المقارنة

درسنا في الدرس الرابع العمليات الحسابية مثل الجمع و الطرح ، هناك عمليات أخرى في لغة بايثون و هي عمليات المقارنة ، عمليات المقارنة تستخدم لمقارنة الأعداد و نتيجتها قيمة منطقية أي إما أن تكون صائبة أو خاطئة ، سنأخذ مثلا على هذه العمليات :

```
>>> 15 > 9
True
```

الأمر السابق يقارن العدد ١٥ و ٩ ، الرمز > يعني أكبر من ، بإمكاننا تخيل الأمر السابق كعبارة ١٥ أكبر من ٩ و هي نفس العبارة التي شرحناها في القسم

السابق و نتيجة هذا الأمر صائبة ، نجد أن مفسر بايثون طبع نتيجة العملية و هي كلمة True و تعني صائب ، نستطيع استخدام عملية مقارنة أخرى مثل < و تعني أصغر من :

```
>>> 15 < 9  
False
```

الأمر السابق يعني أن ١٥ أصغر من ٩ ، هذه المقارنة خاطئة لذلك يعطينا المفسر نتيجة False و تعني خاطئ.

هناك عدة عمليات مقارنة في لغة بايثون و هي :

- == : علامتا يساوي تعنيان أن العددين متساويين ، هذه العملية تعطي نتيجة صائبة إذا كان العددان متساويين حقا و تعطي نتيجة خاطئة إذا لم يكونا متساويين ، مثلا :

```
>>> 15 == 9
```

تعطي نتيجة خاطئة لأن العدد ١٥ لا يساوي العدد ٩.

- != : علامة تعجب ثم يساوي تعني عدم المساواة ، هذه العملية هي عكس العملية السابقة و تعطي نتيجة صائبة إذا كان العددان غير متساويين بينما تعطي نتيجة خاطئة إذا كان العددان متساويين ، مثلا :

```
>>> 15 != 9
```

تعطي نتيجة صائبة لأن العدد ١٥ فعلا لا يساوي العدد ٩.

- > : أكبر من ، تعطي نتيجة صائبة فقط إذا كان العدد الأول ( الذي على اليسار ) أكبر من الثاني ( الذي على اليمين ).
- < : أصغر من ، تعطي نتيجة صائبة فقط إذا كان العدد الأول أصغر من الثاني.
- >= : أكبر من أو يساوي ، تعطي نتيجة صائبة إذا كان العدد الأول أكبر من الثاني أو يساويه.

- <=: أصغر من أو يساوي ، تعطي نتيجة صائبة إذا كان العدد الأول أصغر من الثاني أو يساويه.

### ٩.٣ عبارة if

أمر if يجعلنا نتحكم في تنفيذ بعض الأوامر اعتمادا على قيمة منطقية ، يتم تنفيذ الأوامر فقط إذا كانت القيمة المنطقية صائبة ، صيغة عبارة if هي كما يلي :

```
if condition :
    instruction
rest_of_program
```

نكتب الأمر if بعده شرط و هو قيمة منطقية ثم نقطتين رأسيتين ، في السطر الذي يليه نكتب مسافات قبل كتابة الأمر ، بإمكاننا كتابة أي عدد من المسافات حتى لو كانت مسافة واحدة على الأقل ، الأمر الذي نكتب قبله مسافات يتم تنفيذه فقط إذا كان الشرط صائبا ، أما إذا كان الشرط خاطئا فلا يتم تنفيذ أي الأمر ، إذا كتبنا أمرا بدون أن نضع مسافات قبله يعتبر خارج عبارة if و يتم تنفيذه سواء كان الشرط خاطئا أو صائبا ، لنأخذ مثال درجة الطالب الذي ذكرناه في بداية الدرس ، بإمكانك كتابة البرنامج التالي في ملف ثم تشغيله :

```
mark = int ( input ( 'what is your mark ? ' ) )
if mark > 90 :
    print ( 'you are excellent ^_^' )
print ( 'good bye' )
```

في المثال السابق يدخل المستخدم درجته ، بعدها يفحص البرنامج إذا كانت الدرجة أكبر من أو تساوي ٩٠ أو لا ، إذا كذلك يطبع البرنامج عبارة "you are excellent ^\_^" ، فحص الدرجة هو السطر الثاني :

```
if mark >= 90 :
```

استخدمنا أمر if بعده الشرط ، الشرط هنا هو عملية مقارنة ، المتغير mark أكبر من أو تساوي العدد ٩٠ ، إذا كانت قيمة عملية المقارنة صائبة يتم تنفيذ

الأمر التالي أما إذا كانت خاطئة لا يتم تنفيذه ، لاحظ أننا زدنا مسافات قبل كتابة الأمر الذي يلي سطر if :

```
if mark > 90 :  
    print ( 'you are excellent ^_^' )
```

جرب تشغيل البرنامج و أدخل العدد ٧٠ ، ستجد أن البرنامج لن يطبع العبارة "you are excellent ^\_^" لأن المتغير mark ليس أكبر من ٩٠ ، شغل البرنامج مرة أخرى و أدخل هذه المرة العدد ٩٥ ، في هذه الحالة ستجد أن البرنامج سيطبع تلك العبارة لأن الشرط صائب فقيمة المتغير mark أكبر من ٩٠ ، الأمر الأخير في البرنامج يبدأ من بداية السطر لذلك يتم تنفيذه دائما و لا علاقة له بعبارة if و لا بالشرط.

## ٩.٤ تعدد الأوامر في عبارة if

بإمكاننا أن نكتب أكثر من أمر داخل عبارة if ، يجب أن نكتب قبل كل الأوامر داخل عبارة if مسافات متساوية ، هذا نفس البرنامج السابق لكننا زدنا أمرا آخر داخل عبارة if :

```
mark = int ( input ( 'what is your mark ? ' ) )  
if mark > 90 :  
    print ( 'you are excellent ^_^' )  
    print ( 'keep up the good work' )  
print ( 'good bye' )
```

لاحظ في المثال السابق أننا زدنا أمرا داخل عبارة if ، بإمكاننا أن نضع أي عدد من الأوامر داخل عبارة if سواء كان ثلاث أوامر أو أربع أو أكثر ، المهم أن نكتب نفس عدد المسافات قبل كل أمر ، لا يجوز مثلا أن نكتب أربع مسافات قبل الأمر الأول و ثلاث مسافات قبل الأمر الثاني.

## ٩.٥ تعلمنا في هذا الدرس

- القيم المنطقية : إما صائبة و إما خاطئة.
- عمليات المقارنة.
- عبارة if :

```
if condition :  
    instruction  
rest_of_program
```

- تعدد الأوامر في عبارة if :

```
if condition :  
    instruction  
    instruction  
    instruction  
...  
rest_of_program
```

في الدرس القادم سنتعلم إضافات إلى عبارة if يمكننا من التحكم بشكل أكبر في البرنامج.



## الدرس العاشر

الجملة الشرطية ٢



تعلمنا في الدرس السابق كيف نتحكم في تنفيذ بعض الأوامر اعتمادا على شرط معين ، ماذا لو كان معنا أكثر من شرط واحد و أردنا القيام بأوامر مختلفة اعتمادا على الشرط الذي تحقق ، مثلا قد نريد أن نطبع عبارة إذا كانت درجة المستخدم أكبر من ٩٠ لكننا نريد أن نطبع عبارة أخرى إذا كانت الدرجة أقل من ٦٠ ، في هذا الدرس سنتعلم كيف ننفذ أوامر اعتمادا على عدة شروط و ليس شرطا واحدا ، سنتعلم أيضا كيف نستخدم عدة شروط و ننفذ أمرا معينا اعتمادا على الشرط الذي يتحقق منها.

## ١.١ الأوامر المنطقية

تعلمنا القيم المنطقية و أنها إما تكون صائبة أو خاطئة ، استخدمنا عمليات المقارنة لتحديد بعض الشروط و تعلمنا كيف أن عمليات المقارنة تعطي نتيجة منطقية ، أحيانا يكون لدينا عدة شروط و نريد أن نتأكد أنها جميعا صائبة ، أحيانا أخرى يكون لدينا عدة شروط و نريد أن نفصح إذا كان واحد منها على الأقل صائبا.

هناك عدة أوامر منطقية في لغة بايثون تفيدنا في التحكم بالشروط :

- or : هذا الأمر يأخذ شرطين ، إذا كان أحد الشرطين صائبا تكون نتيجة الأمر صائبة ، لو أردنا مثلا أن نفحص ما إذا كان متغير ما ، مثلا x ، يساوي ٣ أو ٧ نكتب :

```
if x == 3 or x == 7 :  
    print ( 'x equals 3 or 7' )
```

الأمر السابق يفحص إذا كان x يساوي ٣ أو ٧ و ينفذ أمر الطباعة إذا كان أي من الشرطين صائبا.

- and : هذا الأمر يأخذ شرطين و يعطي نتيجة صائبة إذا كان كلا الشرطين صائبان ، لو أردنا مثلا أن نفحص أن متغيرا ، مثلا x ، أكبر من ٠ لكنه في نفس الوقت لا يساوي ٣ نكتب الآتي :

```
if x > 0 and x != 3 :
    print ( 'x is more than 0 and is not 3' )
```

يتم تنفيذ أمر الطباعة السابق فقط إذا كان المتغير x أكبر من ٠ و في نفس الوقت لا يساوي ٣.

- not : هذا الأمر يأخذ شرطا واحدا فقط و يعطي نتيجة معاكسة لنتيجة الشرط ، إذا كان الشرط صائبا فإن نتيجة أمر not تصبح خاطئة ، أما إذا كان الشرط خاطئا تكون نتيجة أمر not صائبة ، مثلا إذا أردنا فحص ما إذا كان المتغير x ليس أكبر من ٠ نكتب :

```
if not x > 0 :
    print ( 'x is not more than 0' )
```

يتم تنفيذ أمر الباعة فقط إذا لم يكن المتغير x أكبر من ٠.

## ١.١.٢ عبارة elif

أحيانا يكون لدينا شرطان ، نريد أن ننفذ أوامر إذا تحقق الشرط الأول لكننا نريد أن ننفذ أوامر أخرى إذا تحقق الشرط الثاني ، في مقدمة الدرس ذكرنا مثلا ، لنفترض أننا نريد أن نخبر المستخدم أنه ممتاز إذا كانت الدرجة أكبر أو تساوي ٩٠ لكننا نريد أن نخبره أنه كسول إذا كانت الدرجة أقل من ٦٠ ، أمر elif يمكننا من القيام بهذا ، هذا الأمر مشابه لأمر if إلى حد كبير فلديه شرط و لديه أوامر يتم تنفيذها إذا كان الشرط صائبا ، لنأخذ مثلا على هذا الأمر ، بإمكانك كتابة البرنامج التالي في ملف و تشغيله :

```
mark = int ( input ( 'what is your mark ? ' ) )
if mark >= 90 :
    print ( 'you are excellent ^^' )
elif mark < 60 :
    print ( 'you are very bad x_x' )
```

البرنامج السابق يطلب درجة من المستخدم ، في عبارة if في السطر الثاني يفحص البرنامج إذا كانت الدرجة أكبر أو تساوي ٩٠ ، إذا كان ذلك صحيحا يطبع البرنامج عبارة "you are excellent ^\_^" و لا يقوم بتنفيذ عبارة elif ، أما إذا كان الشرط خاطئا ينتقل البرنامج إلى عبارة elif و يفحص شرطها و هو أن الدرجة أقل من ٦٠ ، إذا كان الشرط صائبا يقوم البرنامج بتنفيذ أمرها و هو طباعة عبارة "you are very bad x\_x" ، أما إذا كان الشرط خاطئا لا يقوم البرنامج بطباعة أي شيء.

لاحظ أن أمر elif يبدأ في بداية السطر مثل الأمر if ، أيضا نكتب قبل أوامر elif مسافات تماما مثل أوامر if ، الفرق بين الأمرين أن أمر elif لا يتم فحص شرطه و لا تنفيذ أوامره إلا إذا كان الشرط في أمر if خاطئا.

بإمكاننا كتابة عدة أوامر elif و يتم فحص شروطها بالترتيب ، يتم تنفيذ أوامر أول عبارة يكون شرطها صائبا و تجاهل كل العبارات التالية ، هذا البرنامج مثال على ذلك :

```
mark = int ( input ( 'what is your mark ? ' ) )
if mark >= 90 :
    print ( 'you are excellent ^_^' )
elif mark >= 80 :
    print ( 'you are good' )
elif mark < 60 :
    print ( 'you are very bad x_x' )
```

يتم أولا فحص شرط if ، لا يتم فحص شرط elif الأولى إلا إذا كان شرط if خاطئا و لا يتم فحص شرط عبارة elif الثانية إلا إذا كانت كل الشروط في عبارة if و elif التي فوقها خاطئة.

### ١.٣ عبارة else

إذا كان لدينا عبارة if و أردنا تنفيذ أمر إذا كان شرطها خاطئا نستخدم أمر else ، أمر else مثل أمر elif لكنه لا يأخذ شرطا بل يتم تنفيذ أوامره إذا كانت شروط العبارات السابقة خاطئة.

لنفترض مثلا أننا نريد أن نفحص إذا كانت الدرجة أكبر من ٩٠ أو تساوي ٩٠ و نخبر المستخدم أنه ممتاز ، نخبره أنه سيء إذا كانت الدرجة أقل من ٦٠ ، لدينا أيضا عبارة نطبعها في كل الحالات الأخرى أي إذا لم تكن الدرجة أكبر أو تساوي ٩٠ و لم تكن أيضا أقل من ٦٠ ، نستطيع القيام بهذا باستخدام أمر else كما يلي :

```
mark = int ( input ( 'what is your mark ? ' ) )
if mark >= 90 :
    print ( 'you are excellent ^_^' )
elif mark < 60 :
    print ( 'you are very bad x_x' )
else :
    print ( 'you are ok' )
```

تلاحظ أن عبارة else مماثلة لعبارة elif و if باستثناء أن الأمر هو else و أن الشرط غير موجود ، بإمكاننا استخدام أمر else حتى لو لم نستخدم أمر elif أي أن نكتب if بعدها else ، مثلا :

```
mark = int ( input ( 'what is your mark ? ' ) )
if mark >= 90 :
    print ( 'you are excellent ^_^' )
else :
    print ( 'you are not excellent' )
```

## ١.٤.٤ تعلمنا في هذا الدرس

- الأوامر المنطقية :

- or

- and

- not

- عبارة elif :

```
if condition :  
    instruction  
elif condition :  
    instruction
```

- عبارة else :

```
if condition :  
    instruction  
else :  
    instruction
```

في الدرس القادم سنتعلم كيفية التحكم في تكرار الأوامر و تنفيذها أكثر من مرة.



# الدرس الحادي عشر

الحلقات ١

تعلّمنا في الدرسين السابقين كيفية التحكم في تنفيذ الأوامر اعتماداً على شروط معينة ، في هذا الدرس سنتعلّم كيفية استخدام الشروط لتكرار عدد من الأوامر أكثر من مرة ، مثلاً قد نكتب برنامجاً يطلب كلمة سر من المستخدم ، إذا أدخل المستخدم كلمة سر خاطئة يعيد البرنامج طلب كلمة السر ، لا يتوقف البرنامج عن طلب كلمة السر حتى يدخل المستخدم كلمة سر صحيحة ، هذا مثال على تكرار الأوامر.

## 11.1 أمر while

أمر if يقوم بتنفيذ أوامر اعتماداً على شرط ، أمر while يقوم أيضاً بتنفيذ أوامر اعتماداً على شرط لكن بدل تنفيذ الأوامر مرة واحدة يقوم بتكرارها حتى تصبح قيمة الشرط خاطئة ، تسمى جملة while حلقة لأن أوامرها تتكرر عدة مرات بخلاف جملة if التي تُنفذ مرة واحدة فقط ، قبل أن نأخذ مثالاً على while لتعلّم صيغته و هي :

```
while condition :  
    instruction  
rest_of_program
```

صيغة أمر while مماثلة تماماً لصيغة أمر if باستثناء كتابتنا لكلمة while بدل if ، يمكننا أيضاً كتابة أكثر من أمر للحلقة تماماً مثل جملة if.

لنطبق مثال كلمة السر الذي ذكرناه في مقدمة الدرس ، لنفترض أن البرنامج يريد كلمة سر من المستخدم ، كلمة السر الصحيحة هي "secret" ، إذا أدخل المستخدم كلمة أخرى يعيد البرنامج السؤال عن كلمة السر حتى يدخل المستخدم كلمة السر الصحيحة ، في هذا البرنامج تكرار حيث أنه يكرر السؤال عن كلمة السر ، نستطيع استخدام أمر while في مثل هذه الحالة :

```
password = ''  
while password != 'secret' :  
    password = input ( 'what is the password ? ' )  
print ( 'you entered the correct password' )
```

البرنامج السابق يتأكد من الشرط أولاً و هو أن password لا يساوي "secret" ، المتغير password في بداية البرنامج هو نص فارغ لذلك فالشرط صائب ، ينفذ البرنامج أوامر while و فيها أمر واحد : طلب إدخال كلمة السر و يتم حفظها في المتغير password ، بعدها تنتهي الحلقة و يفحص البرنامج الشرط مرة أخرى ، إذا أدخل المستخدم كلمة "secret" يصبح الشرط خاطئاً و تنتهي الحلقة و ينفذ البرنامج الأمر التالي و هو طباعة العبارة "you entered the correct password" ، إذا لم تكن كلمة السر "secret" يعيد البرنامج طلب كلمة السر و لا يتوقف حتى يدخل المستخدم كلمة السر الصحيحة.

## ١١.٢ مقاطعة أوامر الحلقة

يمكننا داخل الحلقة أن نقاطعها قبل انتهاء تنفيذ الأوامر ، هناك أمران لمقاطعة تنفيذ الأوامر في الحلقة :

- break : أمر break يوقف الحلقة و يخرج منها ، أي أمر بعد break داخل حلقة while لا يتم تنفيذه و يستمر البرنامج في تنفيذ الأوامر التي بعد الحلقة.

- continue : أمر continue يقاطع الحلقة و يعود إلى بدايتها ، يتم فحص الشرط مرة أخرى و تنفيذ أوامر الحلقة إذا كان الشرط صحيحاً.

سنذكر مثالا على استخدام الأمر break فقط في هذا الدرس ، لنفترض أن البرنامج يسمح للمستخدم بأن يتوقف عن إدخال كلمة السر إذا أدخل عبارة "cancel" ، نستطيع أن نقطع الحلقة إذا أدخل المستخدم العبارة "cancel" ، بإمكاننا القيام بهذا بكتابة جملة if داخل الحلقة و يكون شرطها أن password يساوي "cancel" ، ننفذ أمر break الذي يقطع الحلقة داخل جملة



if ، المثال التالي تطبيق لما سبق :

```
password = ''
while password != 'secret' :
    if password == 'cancel' :
        break
    password = input ( 'what is the password ? ' )
```

إذا أدخل المستخدم كلمة "cancel" يتم تنفيذ أمر break و بهذا تتوقف الحلقة و يكمل البرنامج بعدها.

## ١١.٣ تعلمنا في هذا الدرس

- أمر while :

```
while condition :
    instruction
rest_of_program
```

- أمر break.

- أمر continue

في الدرس القادم سنتعلم القوائم في لغة بايثون ، القوائم تنفعنا إذا كان لدينا عدد كبير من المتغيرات.



# الدرس الثاني عشر

القوائم

كثيرا ما نحتاج حفظ عدد كبير من القيم يكون بينها أمر مشترك ، مثلا قد نريد حفظ أعمار عدد كبير من الأشخاص ، تخيل أن لدينا ١٠٠ شخص نريد حفظ أعمارهم في متغيرات ، من الصعب تعريف ١٠٠ متغير و استخدام قيمها ، القوائم تحل هذه المشكلة ، قبل بدايتنا للدرس ننوه أن كثيرا من لغات البرمجة تستخدم تعبير "المصفوفات" للإشارة إلى القوائم لكن المصفوفات في لغة بايثون لها ميزات أكثر من المصفوفات في تلك اللغات و إن كان المبدأ مشابها ، إذا صادفت مصطلح "المصفوفات" فتذكر القوائم في بايثون.

## ١٢.١ القوائم

القائمة في لغة بايثون عبارة عن مجموعة قيم ، يمكننا حفظ مجموعة القيم هذه في متغير واحد و يمكننا استخراج أي قيمة نريد من هذا المتغير ، نكتب كل القيم في القائمة بين قوسين مربعين ( [ ] ) ، فصل بين كل قيمة و أخرى بفاصلة ( , ) ، في المثال التالي نعرف متغيرا و نحفظ فيه قائمة بأعداد صحيحة :

```
>>> var = [ 10, 2, -31 ]
```

المتغير var أصبح قائمة فيها ثلاثة أعداد و هي ١٠ و ٢ و ٣١ ، بإمكاننا وضع أعداد أكثر لو أردنا ذلك ، بإمكاننا أيضا حفظ عبارات نصية في القائمة :

```
>>> text = [ 'hello', 'football', 'nice cat' ]
```

المتغير text أصبح يحوي العبارات النصية "hello" و "football" و "nice cat" ، يمكننا أيضا في لغة بايثون أن نضع أعدادا و عبارات نصية في نفس القائمة لو أردنا ذلك :

```
>>> text = [ 1, 'hello', 2, 3, 'nice cat' ]
```

بهذه الطريقة نستطيع حفظ العديد من القيم في متغير واحد و تغييرها باستخدام ذلك المتغير.

## ١٢.٢ التوصل لعناصر القائمة

ذكرنا أن وظيفة القوائم هي حفظ عدة قيم في متغير واحد و تعلمنا في القسم السابق من الدرس كيف ننشئ قائمة و نخزن فيها قيما ، في هذا القسم سنتعلم كيف نغير قيم العناصر في القائمة و نستخدمها في العمليات الحسابية.

نكتب اسم المتغير و بعده قوسان مربعان ( [ ] ) لتحديد العنصر الذي نريد التوصل إليه ، نكتب رقم العنصر الذي نريده بين القوسين ، الأمر التالي يعطي مثلا لتغيير قيمة عنصر في قائمة اسمها var :

```
>>> var = [ 4, 10, 12 ]
>>> var [ 0 ] = 33
```

الأمر السابق يغير قيمة أول عنصر في القائمة var إلى ٣٣ ، قيمة العنصر الأول قبل التغيير كانت ٤ ، لاحظ أن رقم العنصر الأول هو ٠ وليس ١ ، يبدأ تعداد عناصر أي قائمة بالرقم ٠ ، بإمكاننا أيضا استخدام عناصر القائمة في العمليات الحسابية ، المثال التالي يطبع ناتج ضرب العنصر الثاني في العدد ٣ :

```
>>> var [ 1 ] * 3
30
```

بإمكاننا تحديد العنصر باستخدام متغير عددي بدل كتابة العدد مباشرة ، مثلا :

```
>>> x = 1
>>> var [ x ] * 3
30
```

في المثال السابق حددنا العنصر x من القائمة ، يتم تحديد العنصر الذي رقمه يساوي قيمة المتغير x ، لأن قيمة المتغير x هي ١ تم تحديد العنصر رقم ١ ، تذكر أن رقم العنصر الأول هو ٠ و رقم العنصر الثاني هو ١.

بإمكاننا إضافة عناصر للقائمة باستخدام الأمر append ، نكتب اسم القائمة

بعده نقطة بعدها كلمة **append** ثم **قوسين** ، نكتب بين القوسين **القيمة التي نريد إضافتها للقائمة** ، المثال التالي يضيف العدد ٩ إلى القائمة var :

```
>>> var.append ( 9 )
```

بإمكاننا إضافة الأعداد و العبارات النصية و أي قيمة نريد ، نستطيع أيضا إضافة المتغيرات و لسنا ملزمين بإضافة قيمة عددية مباشرة كما في المثال السابق ، المثال التالي يضيف المتغير x للقائمة :

```
x = 14  
var.append ( x )
```

لأن قيمة x هي ١٤ يتم إضافة العدد ١٤ إلى القائمة.

### ١٢.٣ التوصل لأحرف العبارات النصية

يمكننا التوصل لأحرف العبارات النصية كما نتوصل لعناصر القائمة ، مثلا لو كان لدينا المتغير التالي :

```
>>> var = 'hello'
```

بإمكاننا التوصل للحرف الأول من النص كما نتوصل للعنصر الأول من القائمة ، المثال التالي يطبع الحرف الأول :

```
>>> print ( var [ 0 ] )  
h
```

بإمكاننا طباعة جزء من النص باستخدام عملية الجمع بين الأحرف التي نريدها ، المثال التالي يوضح هذا بطباعة أول ثلاثة أحرف من النص :

```
>>> print ( var [ 0 ] + var [ 1 ] + var [ 2 ] )
```

جمعنا في المثال السابق الحرف الأول و الثاني و الثالث من النص فنتج لنا نص يحوي الأحرف الثلاثة الأولى منه و قمنا بطباعتها.

## ١٢.٤ تعلمنا في هذا الدرس

- القوائم.
- التوصل لعناصر القائمة.
- إضافة عناصر للقائمة.

في الدرس القادم سنتعلم طريقة نستفيد بها من أمر while في التعامل مع القوائم.



## الدرس الثالث عشر

الحلقات ٢

في الدرس السابق تعلمنا القوائم و كيفية تعديل عناصرها ، أحيانا نريد تغيير قيم كل عناصر القائمة ، تخيل أن لدينا قائمة فيها ١٠٠ عنصر و نريد تغيير قيمها كلها ، من الصعب تغيير قيم العناصر واحدا واحدا و كتابة أمر لكل عنصر ، إذا قمنا بذلك فعلينا كتابة ١٠٠ أمر لتغيير قيم العناصر ، الحلقات تفيدنا في مثل هذه الحالة لتعديل قيم عناصر كثيرة في القائمة دون كتابة أمر لكل عنصر ، في هذا الدرس سنتعلم القيام بذلك.

### ١٣.١ تعديل قيم القائمة باستخدام while

تعلمنا في الدرس الحادي عشر حلقة while ، نضع فيها شرطا و يتم تنفيذ الأوامر إذا كان الشرط صائبا و يتم إعادة تنفيذ الأوامر حتى يصبح الشرط خاطئا ، بإمكاننا استخدام هذه الحلقة لتعديل عناصر قائمة ، للقيام بذلك نستخدم متغيرا عدديا ، تخيل أن لدينا قائمة فيها ثلاثة عناصر هي ١٠ و ٢ و ٣١ ، نريد تعديل قيمة كل عنصر إلى العدد ٥٠٠ ، المثال التالي يوضح كيفية القيام بذلك باستخدام while :

```
var = [ 10, 2, -31 ]
x = 0
while x < 3 :
    var [ x ] = 500
    x = x + 1
```

في المثال السابق لدينا قائمة اسمها var و فيها ثلاث عناصر ، لدينا أيضا متغير اسمه x قيمته ٠ ، المتغير x يمثل عددا يزداد كل مرة يتم تنفيذ الحلقة فيها ، نستطيع استخدام هذا العداد لنمر على كل عنصر من القائمة ، ينفذ البرنامج حلقة while و شرطها أن x أصغر من ٣ ، الشرط صائب في أول مرة يتم تنفيذ الحلقة فيها ، وأمر الحلقة هي تغيير قيمة العنصر x من القائمة إلى ٥٠٠ ، لأن x يحمل القيمة ٠ يتم تغيير العنصر رقم ٠ من القائمة ، بعدها يتم زيادة المتغير x بقيمة ١ ، تصبح قيمة المتغير x ١ و يبقى شرط الحلقة صائبا و يتم تنفيذها مرة أخرى لكن هذه المرة يتم تغيير العنصر ١ من القائمة لأن



قيمة x أصبحت ١ ، يزداد المتغير x و يصبح ٢ و يتم تغيير العنصر رقم ٢ من القائمة ، بعد أن يحمل المتغير x قيمة ٣ يصبح شرط الحلقة خاطئا و تتوقف الحلقة ، هكذا نكون قد غيرنا قيم كل عناصر القائمة إلى ٥٠٠.

بإمكاننا القيام بأمور أكثر تعقيدا ، مثلا نستطيع أن نجعل قيمة كل عنصر ضعف قيمته الأولى ، نقوم بذلك بأن نضرب كل عنصر في ٢ ، البرنامج التالي يظهر هذا الأمر :

```
var = [ 10, 2, -31 ]
x = 0
while x < 3 :
    var [ x ] = var [ x ] * 2
    x = x + 1
```

في المثال السابق بدل أن نغير قيمة العناصر إلى ٥٠٠ غيرناها إلى نفس قيمة العنصر القديمة مضروبة في ٢ و بهذا نضاعف كل العناصر.

## ١٣.٢ معرفة عدد عناصر القائمة

أحيانا تكون لدينا قائمة لا نعرف عدد عناصرها بالتحديد ، ربما نضيف إليها عددا من العناصر و نحذف منها عددا آخر بشكل عشوائي ، أمر len يسمح لنا بمعرفة عدد عناصر القائمة ، نكتب كلمة len بعدها قوسان و بينهما اسم القائمة التي نريد عدد عناصرها ، في المثال التالي نعرف قائمة فيها ثلاثة عناصر و نقوم بحساب عدد العناصر :

```
>>> var = [ 10, 2, -31 ]
>>> len ( var )
3
```

## ١٣.٣ حذف العناصر من القائمة

نستطيع حذف عناصر من القائمة باستخدام أمر pop ، صيغة الأمر هي كتابة اسم القائمة بعده نقطة بعدها كلمة pop بعدها قوسان ، نكتب بين القوسين رقم العنصر الذي نريد حذفه ، إذا حذفنا العنصر يمكننا حفظ قيمته في متغير ، نقوم بذلك بكتابة أمر pop بعد عملية تعيين للمتغير الثاني

باستخدام العملية **=** في المثال التالي نعرف قائمة فيها ثلاث عناصر ثم نحذف العنصر الثاني منها و نحفظه في المتغير x ، بعد ذلك نقوم بطباعة المتغير x و القائمة لمعرفة القيم المخزنة في كل منهما :

```
>>> var = [ 10, 2, -31 ]
>>> x = var.pop ( 1 )
>>> print ( var )
[10, -31]
>>> print ( x )
2
```

تلاحظ أن الأمر pop حذف العنصر الثاني من القائمة و خزنه في المتغير x.

## ١٣.٤ تعلمنا في هذا الدرس

- تعديل قيم القائمة باستخدام while.
- معرفة عدد عناصر القائمة.
- حذف العناصر من القائمة.

في الدرس القادم سنتعلم نوعا جديدا من الحلقات يسهل علينا التعامل مع القوائم.



# الدرس الرابع عشر

الأوال

في الدروس السابقة تعلمنا كتابة الأوامر ، في هذا الدرس سنتعلم كيفية تقسيم برامجنا بشكل مرتب و إنشاء أوامر أكثر ، سنتعلم كتابة الدوال التي تساعدنا في هذا الأمر.

## ١٤.١ الدوال

الدالة في لغة البرمجة مجموعة أوامر يتم تنفيذها معا ، استخدمنا الكثير من الدوال في الدروس الماضية ، قبل إعطاء مثال على الدوال سنتعلم الصيغة العامة للدالة و هي :

```
def name () :  
    instructions
```

نكتب كلمة **def** بعدها اسم الدالة التي نريد كتابتها ، بإمكاننا اختيار أي اسم كما نختار أسماء المتغيرات تماما ، بعد اسم الدالة نكتب قوسين ثم نقطتين. رأسيتين ، في السطر التالي نكتب أوامر الدالة ، يجب أن نكتب مسافات قبل أوامر الدالة تماما مثلما نفعل مع جمل if و while ، إذا كتبنا سطرا فارغا بدون مسافات فهذا يعني أن تعريف الدالة انتهى ، بإمكاننا كتابة أكثر من أمر في الدالة و هو ما نقوم به عادة ، المثال التالي هو مثال على دالة اسمها func تقوم بطباعة ثلاثة جمل :

```
>>> def func () :  
...     print ( 'hello' )  
...     print ( 'my name is Mohammad' )  
...     print ( 'how are you ?' )  
...
```

بعد تعريفنا للدالة نستطيع تنفيذ أوامرها ، يسمى تنفيذ أوامر الدالة استدعاء الدالة ، نستدعي الدالة بأن نكتب اسم الدالة بعده قوسان ، نستطيع مثلا استدعاء دالة func التي عرفناها في المثال السابق كما يلي :

```
>>> func ()  
hello  
my name is Mohammad  
how are you ?
```

بكتابتنا للأمر السابق نكون استدعينا الدالة السابقة func و ينفذ البرنامج

أوامر الدالة ، أوامر الدالة هي ثلاث أوامر طباعة بثلاث عبارات لذلك تجد أن البرنامج قام بطباعة العبارات الثلاث في الدالة.

في تعريف و استدعاء الدالة كتبنا قوسين بعد اسم الدالة ، لم نكتب بين الأقواس شيئاً ، كل الدوال تكتب بهذه الطريقة و كل الأوامر التي بعدها قوسان تكون دوالاً ، استخدمنا دوال كثيرة في الدروس السابقة مثل دالة print و input ، تلاحظ أننا دائماً نكتب قوسين فارغين بعد الأمر و أحياناً نكتب شيئاً داخل القوسين ، سنتعلم هذا في القسم التالي من هذا الدرس.

## ١٤.٢ إعطاء قيمة للدالة

أحياناً نحتاج إدخال قيمة للدوال ، مثلاً دالة print تأخذ العبارات التي نريد طباعتها ، على سبيل المثال :

```
print ( 'hello' )
```

العبرة "hello" كانت القيمة التي أعطيناها للدالة و الدالة تقوم بطباعتها ، كثيراً ما نحتاج كتابة دالة تأخذ قيمة ، لتعريف دالة تأخذ قيمة نكتب في تعريف الدالة بين القوسين أسماء متغيرات ، بإمكاننا استخدام أي اسم متغير نريد ، مثلاً قد نريد كتابة دالة ترحب بالأشخاص كما يلي :

```
>>> def greet ( name ) :  
...     print ( 'hello', name )  
...
```

تأخذ الدالة قيمة واحدة و أسميناها name ، بإمكاننا استخدام هذا المتغير داخل الدالة ليعني القيمة التي تم إدخالها للدالة ، تقوم الدالة بطباعة كلمة "hello" بعدها اسم الشخص الذي أعطيناه للدالة ، إذا أردنا استدعاء الدالة يجب علينا إعطاؤها قيمة تحل محل المتغير name ، هذا مثال على استدعاء الدالة greet و إعطائها الاسم "Mohammad" :

```
>>> greet ( 'Mohammad' )  
hello Mohammad
```

لاحظ كيف طبعت الدالة الاسم الذي أعطيناه إياها ، بإمكاننا تجربة أي اسم آخر :

```
>>> greet ( 'Ahmad' )  
hello Ahmad  
>>> greet ( 'Arwa' )  
hello Arwa
```

بإمكاننا أن نعرف دالة تأخذ أكثر من قيمة ، للقيام بذلك علينا في تعريف الدالة أن نكتب أسماء متغيرات بين القوسين و نفصل بين كل متغير و آخر **بفاصلة** ، بإمكاننا اختيار أي أسماء نريدها لكن يجب أن تكون مختلفة ، مثلاً قد نريد كتابة دالة تطبع ناتج ضرب عددين ، نكتب الدالة بهذه الطريقة :

```
>>> def multiply ( a, b ) :  
...     print ( a * b )  
...
```

الدالة السابقة تأخذ عددين و تسميهما a و b ، تقوم الدالة بطباعة ناتج ضربهما ، يجب إعطاء قيمتين للدالة حين استدعائها ، مثلاً :

```
>>> multiply ( 3, 7 )  
21  
>>> multiply ( 22, 35 )  
770
```

بإمكاننا كتابة دالة تأخذ أعداد أكثر من القيم ، علينا فقط أن نعطي كل قيمة اسماً و نفصل بين كل اسم و آخر بفاصلة في تعريف الدالة.

### ١٤.٣ إرجاع قيمة من الدالة

أحياناً نحتاج كتابة دالة ترجع قيمة معينة ، على سبيل المثال نستخدم دالة input عادة بهذا الشكل :

```
>>> var = input ( )
```

لاحظ أننا استخدمنا الدالة input في إعطاء قيمة لمتغير ، هذا لأن دالة input ترجع لنا العبارة التي يدخلها المستخدم ، هذا مثال على دالة ترجع قيمة ، أمر return يقوم بإرجاع قيمة ، قبل شرح صيغة هذا الأمر سنعطي

مثالا على دالة تستخدمه :

```
>>> def double ( a ) :  
...     return 2 * a  
...
```

الدالة السابقة تأخذ قيمة واحدة و ترجع ضعف القيمة ، بإمكاننا استخدام الدالة لنخزن قيمة في متغير كما يلي :

```
>>> n = double ( 5 )
```

الأمر السابق يخزن قيمة ١٠ في المتغير n ، هذا لأننا استدعينا الدالة و أعطيناها العدد ٥ ، تقوم الدالة بإرجاع ناتج ضرب العدد ٥ مع ٢ ، من هذا المثال تلاحظ أن صيغة الأمر return هي أن نكتب الكلمة return و بعدها القيمة التي نريد إرجاعها :

```
return value
```

القيمة يمكن أن تكون قيمة مباشرة كأن نكتب العدد ٢ و يمكن أن تكون متغيرا أو عملية رياضية ، يمكن أيضا أن تكون القيمة جملة نصية.

## ١٤.٤ تعلمنا في هذا الدرس

- الدّوال.
- إعطاء قيمة للدّالة.
- إرجاع قيمة من الدّالة.

في معظم الدروس السابقة كتبنا أوامر تقوم بمهام بسيطة ، في الدرس القادم سنتعلم كيفية القيام بأوامر أكثر تعقيدا و سنكتب برنامجا يقوم بضغط الملفات بصيغة zip.



# الدرس الخامس عشر

الوحدات



في الدروس السابقة تعلمنا عددا من الأوامر الأساسية في لغة بايثون لكننا لم نكتب برنامجا له وظيفة محددة ، في هذا الدرس سنتعلم القيام ببعض المهام المعقدة و سنكتب برنامجا يقوم بضغط الملفات بصيغة zip ، البرنامج الذي سنكتبه في هذا الدرس يقوم فقط بإضافة ملفات إلى ملف zip و لا يستطيع استخراجها و لا حذفها ، البرنامج لا يستطيع أيضا إضافة المجلدات.

هذا الدرس يأتي مع ملفين : الأول هو write\_zip.py و تجده في [هذا الرابط](#) ، الملف الثاني هو lesson15.bmp و تجده في [هذا الرابط](#) ، قم بتحميل الملفين قبل البدء في الدرس ، تأكد أن جميع الملفات موجودة في نفس المجلد الذي فيه ملف تشغيل بايثون الذي تستخدمه لتشغيل المفسر.

## ١٥.١ الوحدات

في الدروس السابقة تعلمنا عدة أوامر تقوم بالطباعة و كتابة الملفات و التحكم بالحلقات و غيرها ، كل تلك الأوامر كانت أوامر موجودة في لغة بايثون ، هذه الأوامر مع أنها مفيدة لكنها لا تعطينا قدرات كافية لكتابة برامج معقدة ، هنا تأتي فائدة الوحدات ، الوحدات هي مجموعة دوال جاهزة تعطينا قدرة للقيام بأمر معين ، الدوال مكتوبة مسبقا بمعنى أن هناك من برمجها و كل ما علينا فعله هو استخدامها في برنامجنا ، في هذا الدرس سنستخدم وحدة تساعدنا على ضغط الملفات بصيغة zip.

## ١٥.٢ استيراد الوحدات

قبل استخدام أوامر أي وحدة علينا استيرادها ، استيراد الوحدات يتم بالأمر import ، إذا أردنا استيراد وحدة اسمها module فإن صيغة أمر استيرادها هي :

```
>>> import module
```

نكتب الأمر `import` بعده اسم الوحدة التي نريد استيرادها ، الملف `write_zip.py` الذي قمنا بتحميله هو وحدة ، لاستيراده نكتب أمر `import` ثم اسم الملف لكننا لا نكتب مقطع .py الذي ينتهي به الملف :

```
>>> import write_zip
```

لا يمكننا استخدام أوامر الوحدة قبل استيرادها.

### ١٥.٣ استخدام أوامر الوحدة

بعد عملية الاستيراد نستطيع استخدام الدوال التي توفرها لنا الوحدة ، كل وحدة توفر دوال مختلفة و طريقة استخدام كل دالة قد تختلف عن الأخرى ، الصيغة العامة لاستدعاء دالة من وحدة هي كما يلي :

```
>>> module.function ()
```

نكتب أولاً اسم الوحدة بعده نقطة بعدها اسم الدالة ، وحدة `write_zip` التي استوردناها توفر لنا دالة واحدة فقط و هي `compress ()` ، هذه الدالة تقوم بضغط الملفات بصيغة `zip` ، صيغة الدالة هي :

```
>>> write_zip.compress ( 'filename.ext' )
```

مثل الصيغة العامة نكتب أولاً اسم الوحدة و هو `write_zip` بعده نقطة بعدها اسم الدالة `compress` ثم قوسين ، هذه الدالة تأخذ قيمة واحدة و هي اسم الملف الذي نريد ضغطه لذلك نكتب الاسم بين القوسين ، جرب ضغط ملف الصورة `lesson15.bmp` الذي أتى مع الدرس بكتابة الأمر التالي :

```
>>> write_zip.compress ( 'lesson15.bmp' )
```

تذكر دائماً أن تضع اسم الملف كاملاً مع امتداده فمثلاً في الملف `lesson15.bmp` كتبنا المقطع `.bmp` الذي ينتهي به الملف.

إذا نفذت أمر ضغط الملف فسيظهر ملف جديد اسمه `lesson15.zip` بحجم أصغر من حجم ملف الصورة الأصلي ، إذا قمنا بضغطه ستجد الصورة.

في المثال السابق قمنا بضغط ملف محدد ، نستطيع أن نسأل المستخدم عن الملف الذي يريد أن يقوم بضغطه ، في الدرس السابع درسنا الأمر `input` ،

بإمكاننا استخدامه لكتابة برنامج يسأل المستخدم عن الملف الذي يريد ضغطه :

```
name = input ( 'file name : ' )  
>>> write_zip.compress ( name )
```

بهذه الطريقة نستطيع ضغط أي ملف يريده المستخدم.

تعلمنا في هذا القسم استخدام وحدة write\_zip التي أتت مع الدرس ، هناك الكثير من الوحدات الموجودة في لغة بايثون و التي تساعدنا للقيام بالكثير من المهام ، وحدة write\_zip نفسها تستخدم وحدات أخرى للقيام بالمهمة ، لن نتطرق هذه السلسلة للوحدات التي توفرها بايثون لكن معظم الوحدات لها شروح خاصة بها و بإمكان الدارس البحث عنها و قراءتها.

## ١٥.٤ تعلمنا في هذا الدرس

- الوحدات.
- استيراد الوحدات
- import** module
- استخدام دوالّ الوحدة.

هذا الدرس كان آخر درس في هذه السلسلة ، بعد هذا الدرس سيتم طرح مشروع يعطي تجربة أكثر واقعية من الأمثلة التي تم طرحها خلال السلسلة ، سيطبق الدارس في المشروع عددا من المفاهيم التي تعلمها خلال هذه السلسلة.



مشروع السلسلة

بعد نهاية الدروس في هذه السلسلة نقدم مشروعا برمجة بلغة بايثون يقوم به الدارس ، سيعطي هذا المشروع تجربة أكثر واقعية من الأمثلة الموجودة ضمن السلسلة ، المشروع يتطلب ملفين : الأول هو [dlserver.pyc](#) ، بإمكانك تحميله من [هذا الرابط](#) ، الثاني هو [dlc.pyc](#) و بإمكانك تحميله من [هذا الرابط](#).

في هذا الشرح سنذكر ثلاثة نقاط ، بداية فكرة المشروع ، بعدها سنذكر أقسام المشروع ، أخيرا سنتحدث عن المطلوب من الدارس في هذا المشروع.

## ١٦.١ فكرة المشروع

فكرة المشروع هي مدير تحميل يقوم بتحميل الملفات من الإنترنت ، يستكمل أيضا تحميل الملفات إذا كنا بدأنا تحميلها و توقفنا في وقت سابق ثم أردنا إكمال التحميل.

## ١٦.٢ أقسام المشروع

المشروع مقسم إلى ثلاثة أقسام :

١. مدير التحميل :

القسم الأول هو برنامج مدير التحميل [dlserver.pyc](#) ، هذا القسم هو الذي يقوم فعليا بتحميل الملفات من الإنترنت إلى الجهاز ، أيضا يستكمل التحميلات التي توقفت قبل أن تنتهي ، نستطيع تشغيل هذا البرنامج مثلما شغل أي برنامج بايثون ، نشغل الطرفية و نكتب الأمر التالي :

```
python dlserver.pyc
```

إذا كنت تستخدم نظام أوبونتو أو ماك فلا تنس أن تستخدم أمر python3 بدل python كما شرحنا في الدرس الثالث ، لا يقوم

البرنامج بأي شيء حين نشغله و لا يبدأ تحميل أي ملف ، بعد التشغيل ينتظر مدير التحميل أوامر من برنامج آخر يرسل له الروابط و الملفات اللي يحملها ، البرنامج الثاني هو القسم الثاني من المشروع.

٢. برنامج التحكم :

القسم الثاني من المشروع هو برنامج التحكم ، وظيفة برنامج التحكم هي أخذ أوامر من المستخدم و إرسالها إلى مدير التحميل ، الأوامر تطلب من مدير التحميل أن يبدأ التحميلات جديدة أو يوقف تحميلات جارية ، في هذا المشروع خمسة أوامر :

○ get :

أولا أمر التحميل get ، هذا الأمر يطلب من مدير التحميل أن يحمل رابطا ، نستخدم هذا الأمر بأن نكتب الأمر get بعده مسافة ثم رابط الملف الذي نريد تحميله ، مثلا :

```
get http://www.google.com/images/logo.gif
```

مدير التحميل يدعم التحميل باستخدام بروتوكول http و https فقط ، بالإضافة إلى ذلك فإن سرعة تحميل الملف الواحد محددة بقيمة ١٠ كيلو بايت لكل ثانية كحد أقصى حتى يسهل على الدارس ملاحظة الملفات حال تحميلها.

○ info :

الأمر الثاني هو info ، هذا الأمر يجعل مدير التحميل يرسل رسالة لبرنامج التحكم فيها معلومات عن الحميلات اللي تعمل ، يعطينا رقم كل تحميل و اسم الملف و نسبة التحميل المنتهية ، يعطينا أيضا إجمالي عدد التحميلات الجارية ، استخدام هذا الأمر هو مجرد كتابة كلمة info :

```
info
```

◦ del :

الأمر الثالث هو أمر الحذف del ، هذا الأمر يحذف تحميل ، نكتب بعد الأمر مسافة ثم رقم التحميل ، أرقام التحميلات تبدأ من العدد ٠ ، هذا يعني أن أول تحميل نضيفه يكون رقمه ٠ و التحميل الثاني يكون رقمه ١ ، إذا أردنا حذف التحميل الأول مثلاً نكتب :

```
del 0
```

إذا أرسلنا أمر الحذف فإن مدير التحميل [يرسل لبرنامج التحكم رسالة](#) يذكر فيها إذا تم حذف التحميل بنجاح أم أن هناك خطأ.

نستطيع أن نعطي أمر الحذف اسم الملف أو رابط الملف بدل الرقم ، مثلاً إذا قمنا بتحميل الصورة من الرابط <http://www.google.com/images/logo.gif> فإن اسم الملف هو logo.gif ، الأمران التاليان يقومان بنفس الشيء تماماً :

```
del http://www.google.com/images/logo.gif  
del logo.gif
```

◦ disconnect :

الأمر الرابع في المشروع هو أمر الفصل disconnect ، هذا الأمر يغلق برنامج الأوامر لكن مدير التحميل يكمل عمليات التحميل و لا يتوقف عن العمل ، صيغة الأمر هي كتابة كلمة disconnect فقط :

```
disconnect
```

إذا أغلقنا برنامج التحكم يمكن تشغيله مرة أخرى لإرسال أوامر أخرى.

◦ exit :

الأمر الأخير هو أمر الخروج exit ، هذا الأمر يوقف كل التحميلات و يغلق مدير التحميل و برنامج التحكم معا ، إذا حاولنا تشغيل برنامج التحكم و مدير التحميل مغلق تظهر لنا رسائل خطأ ، صيغة

هذا الأمر هي كتابة كلمة exit فقط :

```
exit
```

٣. وحدة الاتصال بمدير التحميل

القسم الثالث من مشروعنا هو وحدة في الملف dlc.pyc ، هذي الوحدة توفر دوال للاتصال بمدير التحميل و إرسال الأوامر له و استقبال الرسائل منه ، نستخدم هذه الوحدة في برنامج التحكم ، قبل استخدام أي دالة في الوحدة يجب أن نستوردها باستخدام أمر import كما يلي :

```
import dlc
```

توفر لنا الوحدة أربعة دوال :

١. connect :

الدالة الأولى هي دالة الاتصال بمدير التحميل ، يجب أن نستدعي هذي الدالة قبل إرسال أو استقبال أي شيء بين برنامج التحكم و مدير التحميل ، لا تأخذ أي قيمة و يمكننا استدعاءها بهذا الشكل :

```
dlc.connect ()
```

٢. send :

الدالة الثانية هي دالة الإرسال send ، هذه الدالة ترسل أمرا لمدير التحميل و مدير التحميل ينفذ الأمر ، الدالة تأخذ العبارة التي نريد إرسالها لمدير التحميل ، نستطيع مثلا إرسال الأمر info بهذا الاستدعاء :

```
dlc.send ( 'info' )
```

٣. recv :

الدالة الثالثة دالة الاستقبال ، نستخدم هذه الدالة لاستقبال رسالة من مدير التحميل ، كما شرحنا سابقا فإن لدينا خمسة أوامر في المشروع ، في أمرين من هذه الأوامر الخمسة يأتي رد من مدير التحميل ، الأمر info يرسل إلينا ردا يحوي معلومات عن التحميلات الجارية ، أمر del



يرسل إلينا ردا يخبرنا إن كان حذف الرابط تم بنجاح أو لا ، هذه الدالة لا تأخذ أي قيمة ، نستطيع حفظ الرسالة في متغير التي نتلقاها من مدير التحميل في متغير ، نستطيع تخزين الرد في متغير اسمه message بهذا الشكل :

```
message = dlc.recv ()
```

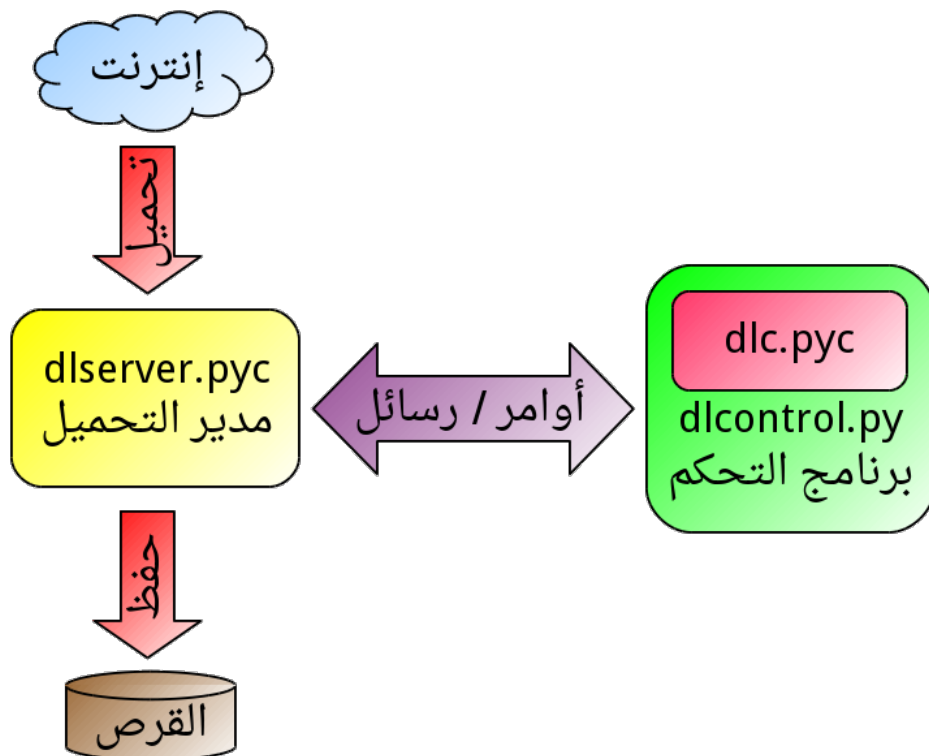
بعدها يحوي المتغير message الرسالة التي أرسلها مدير التحميل.

٤. disconnect :

الدالة الرابعة في الوحدة هي دالة الفصل ، هذه الدالة تفصل برنامج الأوامر عن مدير التحميل و لا يمكن التواصل مع مدير التحميل بعد استدعاء هذه الدالة ، نستدعي هذه الدالة قبل الخروج من برنامج الأوامر ، لا تأخذ هذه الدالة أي قيمة و يمكن استدعاءها بهذا الشكل :

```
dlc.disconnect ()
```

بإمكاننا تمثيل التواصل بين أجزاء المشروع بهذا الرسم :



الصورة ٢٥: تمثيل التواصل بين أجزاء المشروع و الإنترنت و الحاسب

### ١١.٣ مهمة المشارك في المشروع

مهمة المشارك هي برمجة القسم الثاني من المشروع و هو برنامج التحكم ، سنعطيك برنامج مدير التحميل جاهزا و قابلا للتشغيل لكنه كما شرحنا لا يعمل إلا بأوامر تأتي من برنامج التحكم ، أيضا وحدة dlc ستكون جاهزة ليستخدمها المشارك خلال كتابته لبرنامج التحكم في إرسال الأوامر و استقبال الرسائل ، بإمكان المستخدم تحميل أي ملف إذا كان البروتوكول المستخدم هو http أو https لكن لتسهيل الأمر على الدارس تم رفع ثلاث ملفات صور بإمكان الدارس تحميلها للتأكد من عمل المشروع تجدونها في الروابط التالية :

[الصورة الأولى](#)

[الصورة الثانية](#)

[الصورة الثالثة](#)

## ١٦.٤ ملخص المشروع

- مدير تحميل.
- أقسام المشروع :
  - مدير التحميل.
  - برنامج الأوامر.
  - وحدة الاتصال مع مدير التحميل.
- أوامر المشروع :
  - get
  - info
  - del
  - disconnect
  - exit
- مهمة المشارك هي كتابة برنامج الأوامر.

## ١٦.٥ حل المشروع

في هذا الجزء سيتم عرض حل المشروع في أقسام ، كل قسم يحوي جزء من الحل.

١. إرسال الأوامر :

أول شيء نحتاجه هو إرسال الأوامر ، علينا أولاً أخذ الأوامر من المستخدم بعدها نرسل الأوامر باستخدام الدالة send من الوحدة dlc ، تذكر أننا لا نستطيع استخدام أي دالة من الوحدة إلا بعد استيرادها بالأمر import ، أيضاً لا نستطيع إرسال و استقبال أي شيء مع مدير التحميل إلا بعد الاتصال به باستخدام الدالة connect ، استيراد الوحدة و الاتصال بها يجب أن يكونا أول شيئين في برنامجنا ، نستطيع ترتيب خطوات برنامجنا كما يلي :

- استيراد وحدة dlc.
- الاتصال بمدير التحميل.
- طلب الأمر من المستخدم.
- إرسال الأمر لمدير التحميل.

نستطيع ترتيب الخطوات السابقة في البرنامج التالي :

```
import dlc
dlc.connect ()
message = input ()
dlc.send ( message )
```

لا يقوم البرنامج باستقبال الرسائل من مدير التحميل و لا يستمر بطلب الأوامر من المستخدم بل يطلب أمراً واحداً و يرسله ثم يتوقف ، علينا تعديل البرنامج ليدعم استقبال الرسائل و ليستمر بطلب الأوامر من المستخدم حتى يكتب المستخدم أمراً للخروج.

## ٢. استقبال الرسائل و قطع الاتصال :

لدينا خمسة أوامر في البرنامج ، أمر get يتم إرساله فقط و يتكفل مدير التحميل بالتنفيذ دون أي عملية أخرى من برنامج الأوامر ، علينا في بقية الأوامر تنفيذ عمليات أخرى في برنامج الأوامر ، الأمران del و info يجعلان مدير التحميل يرسل رسالة لبرنامج التحكم ، علينا استقبال الرسالة و طباعتها إذا صادفنا أحدهما ، الأمران exit و disconnect يقومان بالخروج من برنامج التحكم ، علينا أن نخرج من برنامج التحكم إذا صادفنا أحدهما ، لاحظ في حالة استخدام أمر الحذف أن المستخدم يكتب الأمر del و يكتب بعده رقم التحميل أو اسم الملف أو رابط الملف لكننا نريد فقط التأكد أن أول ثلاثة أحرف هي الأمر del ، نستطيع ترتيب الكلام السابق في الخطوات التالية :

○ إذا كان الأمر info أو كان يبدأ بالأحرف del :

▪ استقبال رسالة.

▪ اطبع الرسالة.

○ إذا كان الأمر exit أو disconnect :

▪ اقطع الاتصال.

نقوم بالأوامر السابقة بعد أخذ الأمر من المستخدم ، لاحظ أننا في الأمر del لا نفحص كل الأمر بل فقط نفحص أنه يبدأ بالأحرف del ، نستطيع القيام بذلك بأن نعامل العبارة التي يدخلها المستخدم كمصفوفة و نجمع أول ثلاثة أحرف فيها ، بهذا الشكل نكون قد أخذنا أول ثلاثة أحرف فقط من العبارة و نفحص إذا كانت del و نحدد ما إذا احتجنا استقبال رسالة أم لا.

نستطيع إضافة الخطوات السابقة في البرنامج الذي كتبناه في القسم

الأول كما يلي :

```
import dlc
dlc.connect ()
message = input ()
dlc.send ( message )

start = message[0] + message[1] + message[2]
if message == 'info' or start == 'del' :
    receive = dlc.recv ()
    print ( receive )
elif message == 'exit' or message == 'disconnect' :
    dlc.disconnect ()
```

لاحظ أننا عرفنا متغيراً و حفظنا فيه أول ثلاثة أحرف من العبارة التي أدخلها المستخدم لفحص إذا كان الأمر del أم لا.

بقيت مشكلة أن البرنامج يرسل أمراً واحداً و يتوقف حتى لو لم يكن الأمر exit و لا disconnect ، علينا أن نجعل البرنامج يستمر بطلب الأمر من المستخدم حتى يكتب المستخدم الأمر exit أو disconnect.

٣. الاستمرار في طلب الأوامر من المستخدم :

في القسم السابق جعلنا البرنامج يستقبل الرسائل أو يخرج في الأوامر التي تناسب كل عملية ، المشكلة الباقية هي الاستمرار في طلب الأوامر من المستخدم ، نتوقف عن طلب الأوامر من المستخدم فقط إذا صادفنا الأمر disconnect أو الأمر exit ، الحلقات تستخدم للتكرار و بما أننا نكرر عملية ما و هي طلب أمر من المستخدم فعلينا استخدام حلقة while ، ما نكرره هو طلب الأمر و إرساله و استقبال الرسائل في حال كان الأمر del أو info ، باختصار نكرر كل العمليات التي قمنا بها في القسم السابق من الحل باستثناء استيراد الوحدة و الاتصال بمدير التحميل ، إذا علينا أن نضع كل الأوامر الباقية داخل حلقة while ، يبقى لنا شرط الحلقة ، نريد أن نتوقف الحلقة إذا كان الأمر disconnect أو exit ، الشرط الذي نكتبه في الحلقة يجعلها

تستمر لكن الشرط الذي ذكرناه هو شرط التوقف ، نستطيع أن نقول  
أننا نريد أن تستمر الحلقة إذا لم يكن الأمر disconnect ولا exit ،  
نستطيع كتابة شرط استمرار الحلقة في لغة بايثون بهذا الشكل :

```
message != 'disconnect' and message != 'exit'
```

يكون البرنامج بعد التعديل كما يلي :

```
import dlc
dlc.connect ()

message = ''
while message != 'disconnect' and message != 'exit' :
    message = input ()
    dlc.send ( message )

    start = message[0] + message[1] + message[2]
    if message == 'info' or start == 'del' :
        receive = dlc.recv ()
        print ( receive )
    elif message == 'exit' or message == 'disconnect' :
        dlc.disconnect ()
```

لاحظ أننا عرفنا المتغير message قبل الحلقة ، هذا لأننا قمنا  
باستخدام المتغير في شرط الحلقة و لا يمكن استخدام متغير إلا بعد  
تعريفه.

هكذا يستمر البرنامج في طلب الأوامر من المستخدم طوال الوقت حتى  
يكتب المستخدم أحد أمري الخروج disconnect أو exit ، الحل السابق هو  
طريقة تعطي النتيجة المطلوبة و قد يختلف الحل من مبرمج لآخر لكن  
الفكرة مقاربة على الأغلب بين مختلف الحلول.

# خاتمة الكتاب

هذا الكتاب حاول التركيز على الأفكار الأساسية للبرمجة بشكل عام و لم تتطرق الدروس لميزات تخص لغة بايثون بالتحديد ولم تشرح الوحدات المتوفرة في هذه اللغة ، على الأغلب يتساءل الدارس في نهاية هذه الدروس : ماذا بعد ؟ ما هي الخطوة التالية ؟ كيف أطور من نفسي في مجال البرمجة ؟

بعد تعلمك لأساسيات البرمجة فالخطوة التالية هي إتقان إحدى لغات البرمجة ، أكمل تعلمك للغة بايثون و تعلم ميزاتها ، هذه السلسلة لم تذكر الكثير من أوامر بايثون و الميزات المتوفرة فيها ، لن تحتاج لدروس مفصلة بعد هذا فهذه الدروس غطت الأمور الأساسية و كل ما تحتاجه هو دروس مختصرة تذكر الميزات و كيفية استخدامها بشكل سريع يوفر وقتك ، تستطيع أن تختار لغة أخرى غير بايثون و إن كانت بايثون أفضل حيث أنك بدأت تعلمها لكن في كل الأحوال تذكر أن تركز على لغة برمجة واحدة فقط ، لا تشتت تفكيرك في عدة لغات تجعلك تفكر في الاختلاف بينها بدل أن تركز في الأفكار و كيفية بناء البرنامج و إنجاز المهام فلغات البرمجة على اختلافها تحوي شبيها كبير و كلها تؤدي نفس الوظيفة.

بعد إتقانك لإحدى لغات البرمجة ابدأ تعلم الوحدات المتوفرة فيها ، خلال الدروس عرضنا كيفية استخدام الوحدات ، فكر في مهمة معينة و ابحث عن الوحدة التي تقوم بهذه المهمة ، تعلم تلك الوحدة و استخدمها للقيام بهذه المهمة ، في مشروع السلسلة مثلا كان مدير التحميل يحمل الملفات من الإنترنت و قد تم استخدام وحدة urllib للقيام بذلك ، بإمكانك تعلم كيفية تحميل الملفات باستخدامها.

بعد تعلم عدد من الوحدات بإمكانك الانتقال للغات البرمجة المختلفة و تعلمها ، تعلمك للغة برمجة يعتمد على ما تريد القيام به فبعض اللغات تكون



مصممة للقيام بمهمة معينة و تكون أسهل من اللغات الأخرى في تلك المهمة  
بالتحديد.

في أمان الله  
أخوكم محمد الغافلي