

لغة البرمجة بايثون (Python Programming Language)

الدرس الأول: مقدمة عامة

قبل البدء في شرح اساسيات البرمجة بلغة بايثون دعونا نتصفح بعضاً من مميزاتها ونوضح كل منها وكما يلي:

ظهرت لغة بايثون لأول مرة عام ١٩٨٩ بعد اطلاقها للبرمجة من قبل مخترعها الهولندي (غويدو فان روسوم) وتمتاز هذه اللغة بأنها تركز على قابلية القراءة (Readability) باستخدامها لكلمات مفتاحية مشابهة للغة البشر تسهل قراءتها من قبل المختصين وغيرهم. كما تتميز هذه اللغة بسرعتها لكونها لغة مفسرة (interpreted) أي انها تنفذ مباشرة بدون الحاجة الى ترجمة (not compiled) وهي ايضاً لغة نماذج أولية سريعة (rapid prototyping language) يمكن استخدامها لفحص النماذج الأولية بدون الكثير من المقدمات والاعدادات المسبقة فهي لا تحتاج الكثير من الوقت للتطوير فهي اسرع من بقية لغات البرمجة الأخرى مثل ال (C++, Java,...etc.) كما ان من مميزاتها المرونة العالية في استخدام المتغيرات بدون اعلان (no variable declaration) مما يقلل وقت البرمجة والتطوير الى الحد الأدنى. كغيرها من لغات المستوى العالي، تمتاز لغة بايثون انها موجهة نحو الهدف (object oriented) وهي لغة إجرائية (procedural) تسمح بتطوير تطبيقات متعددة الأغراض (general purpose) لإدارة الذاكرة (memory management) وتطوير مكتبات النظام القياسية (standard libraries). يتوافر مفسر اللغة (language interpreter) للتنصيب على مختلف نظم التشغيل من الويندوز والماك واللينكس ويمكن تنزيل النسخة المناسبة لكل نظام من الموقع التالي (<https://www.python.org/downloads/>) والذي عند فتحه تظهر الواجهة التالية:

[About](#)
[Downloads](#)
[Documentation](#)
[Community](#)
[Success Stories](#)
[News](#)
[Events](#)

Download the latest version for Windows

[Download Python 3.4.3](#)
[Download Python 2.7.9](#)

Wondering which version to use? [Here's more about the difference between Python 2 and 3.](#)

Looking for Python with a different OS? Python for [Windows](#), [Linux/UNIX](#), [Mac OS X](#), [Other](#)

Want to help test development versions of Python? [Pre-releases](#)



Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.4.3	2015-02-25	Download	Release Notes
Python 2.7.9	2014-12-10	Download	Release Notes
Python 3.4.2	2014-10-13	Download	Release Notes
Python 3.3.6	2014-10-12	Download	Release Notes
Python 3.2.6	2014-10-12	Download	Release Notes
Python 2.7.8	2014-07-02	Download	Release Notes
Python 2.7.7	2014-06-01	Download	Release Notes

من هذه الواجهة نختار الإصدار المناسب (ويفضل ان يكون الأخير لاحتوائه على اخر التحديثات) وهو الان (وقت كتابة الدرس) الإصدار (python 3.4.3) فننقر عليه لتظهر الواجهة التالية:

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		4281ff86778db65892c05151d5de738d	19554643	SIG
XZ compressed source tarball	Source release		7d092d1bba6e17f0d9bd21b49e441dd5	14421964	SIG
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later	548f79e55708130c755bb0f1ddd921c	24734803	SIG
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	86b29d7ddd60b4b3fc5848de55ca704	23170148	SIG
Windows debug information files	Windows		b3d8752e74a502db97bd0c6ef30ac60f	36900012	SIG
Windows debug information files for 64-bit binaries	Windows		6c1be415ae552e190ef0b06a5de9473	24301250	SIG
Windows help file	Windows		d5703787758eb1a674101ee2b0bc28be	7405996	SIG
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not Itanium processors	f6ade29acaf8fcd0463e69a6e7ccf87	25550848	SIG
Windows x86 MSI installer	Windows		cb450d1cc616bfc8f7a2d6bd88780bf6	24846336	SIG

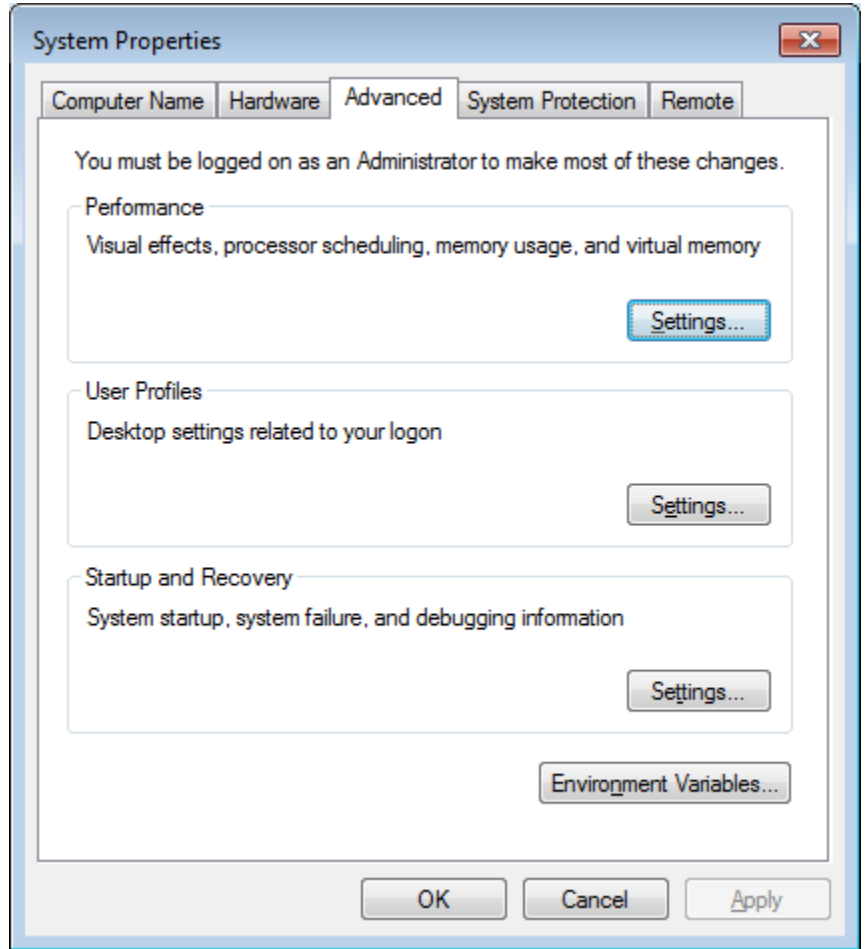
نجد هذه الملفات في أسفل صفحة ونقوم بتنزيل النسخة المناسبة لنظام تشغيلنا (32 or 64) ونقوم بتنصيبها وهي سهلة التنصيب لا تحتاج أي معلومات مسبقة لإكمال تنصيبها.

والان بعد اكمال التنصيب نذهب الى قائمة (Start) ونكتب في خانة البحث (idle) لتظهر لنا ضمن النتائج ال (IDLE Python GUI) فنقوم بفتحها لتظهر الواجهة التالية:

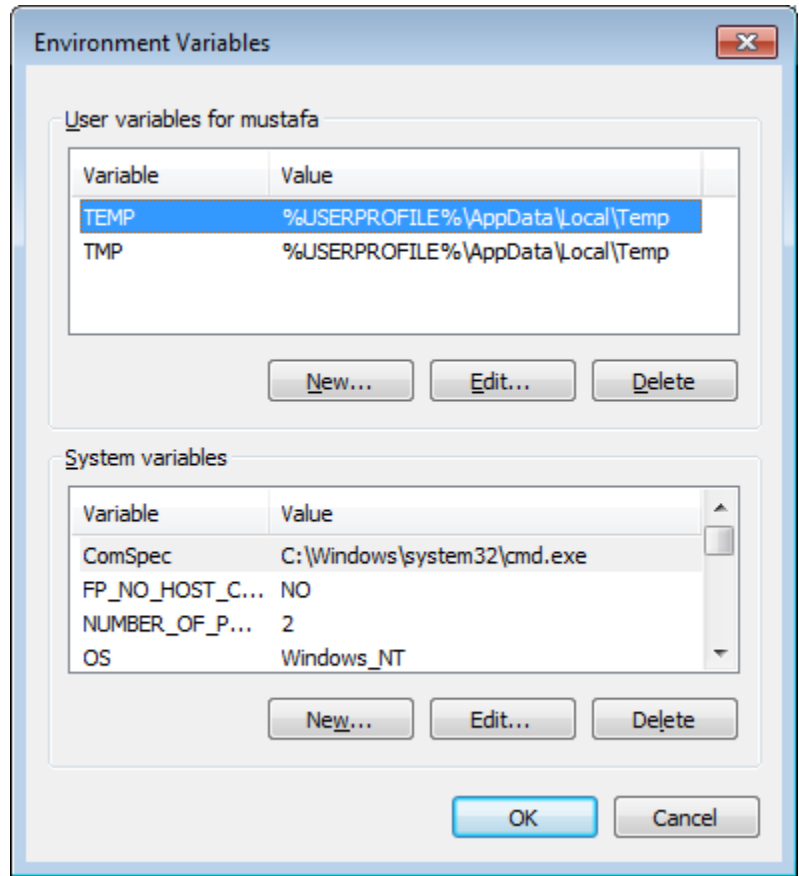
```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

وهذه الواجهة تمثل الواجهة الرسومية لتطوير وتطبيق برامجنا التي سنعمل عليها في هذا الدرس والدروس القادمة ان شاء الله تعالى.

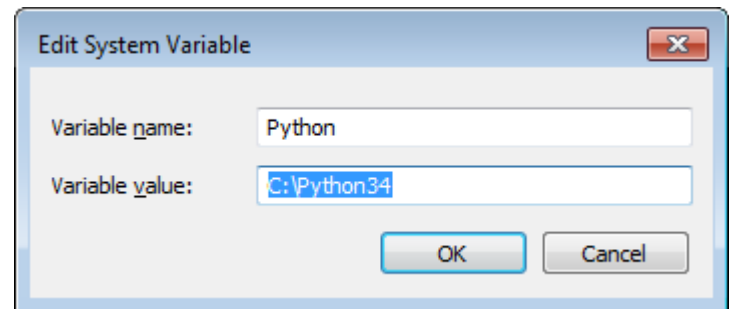
الان ننتقل الى الخطوة المهمة الأخرى وهي ضبط بيئة الويندوز للعمل على البايثون خارج ال (IDLE) الخاص به في سطر الأوامر (Command Prompt) او غيرها ويتم ذلك بالنقر نقرة يمين على ايقونة جهاز الكمبيوتر ثم الذهاب الى properties ثم (advanced system settings) لتظهر النافذة التالية:



الان ننقر على (environment variables) لتظهر النافذة التالية:



الآن نذهب إلى (system variables) وننقر على (new) لتظهر الواجهة التالية والتي نملأها بالمعلومات المبينة في أدناه:



وهنا قمنا بتعريف متغير جديد للويندوز اسمه (python) وامتداده هو امتداد تنصيب المفسر الذي قمنا بتنزيله وتنصيبه قبل قليل وهو في الوضع الطبيعي (c:\python34).

بعدها ننقر على (ok) وبذلك تنتهي عملية تهيئة بيئة الويندوز للتعامل مع البايثون في التطبيقات المختلفة.

الدرس الثاني برنامج (hello world):


بعد ان شرحنا كيفية تنزيل وتنصيب البرمجيات اللازمة للبدء بالعمل على لغة بايثون في الدرس السابق، وشرحنا ايضاً كيفية اعداد بيئة نظام تشغيل الويندوز للتعامل مع هذه اللغة. نأتي اليوم الى البدء بالتعامل مع هذه اللغة وكيفية كتابة اول برنامج باستخدامها ولكن قبل ذلك لا بد من ذكر ملاحظة مهمة جداً وهي اننا اعتمدنا في شرحنا لهذه الدروس على تطبيق (IDLE Python GUI) ولكن هناك الكثير من التطبيقات الأخرى التي يمكن استخدامها لكتابة برامج بايثون واختبار أداؤها وتطويرها ومن أهمها هو (Notepad ++) والذي يمكن تنزيله من الرابط التالي: <https://notepad-plus-plus.org> ورغم اننا لن نتعامل معه في شروحاتنا الا انه من الأفضل ذكره لمن احترقوا استخدامه في تطوير برمجيات بقية لغات البرمجة سابقاً او يحاولون التعامل معه الان بدءاً من لغة بايثون علماً انه يتمتع بمميزات اكثر بكثير من مفسر بايثون التلقائي الذي قمنا بتنصيبه في الدرس السابق وهو (IDLE) الذي سنعمل عليه في دورتنا هذه.

الان نبدأ درسنا على بركة الله:

نذهب الى قائمة (Start) ونقوم بفتح تطبيق (IDLE Python GUI) الذي قمنا بتنصيبه سابقاً لتظهر النافذة التالية:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

الان نرى محرك الأوامر (>>>) وهو محرك الأوامر الخاص بلغة البايثون ويعني اننا مستعدون للبدء بكتابة برنامجنا الأول والذي سيكون كما في كل لغات البرمجة لطباعة عبارة (hello world) على الشاشة والذي يتكون في لغة بايثون من سطر واحد فقط على خلاف بقية لغات البرمجة الأخرى مثل السي بلس بلس والجافا واليكم مقارنة بسيطة بينها بخصوص هذا البرنامج:



Python is simple

```

print "Hello World!"

```

Python

```

#include <iostream.h>
int main()
{
cout << "Hello World!";
}

```


C++

```

public class helloWorld
{
    public static void main(String [] args)
    {
        System.out.println("Hello World!");
    }
}

```

Java

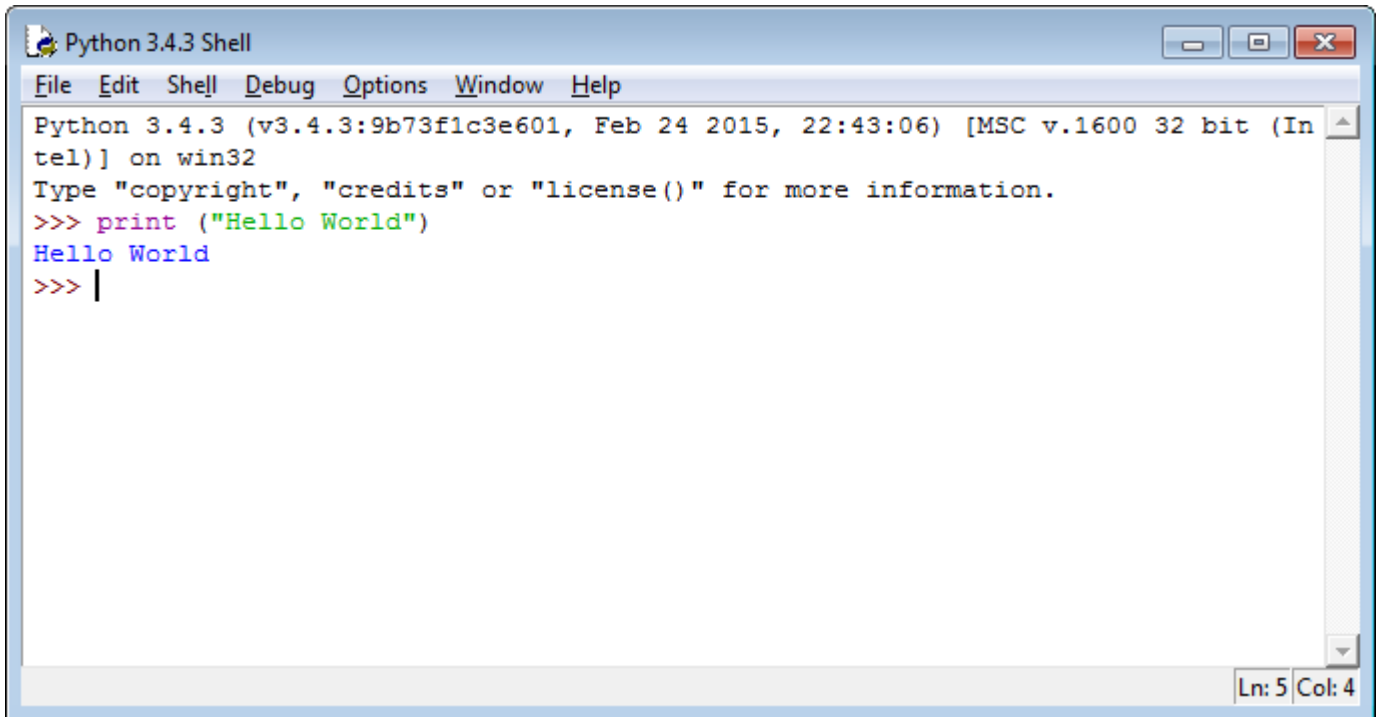


ملاحظة: الصورة أعلاه تحتوي خطأ حيث تم استخدام (print) للغة بايثون بدون اقواس وهو الحال للبايثون ٢،٧ وما قبلها واما بايثون ٣ وما بعدها فتتطلب الاقواس بشك ضروري لذا يجب الانتباه الى النسخة التي تستخدمها وفروقات متطلباتها.

وكما لاحظنا هنا فان طباعة عبارة (Hello world) تتطلب سطرًا واحدًا فقط وهو

`print ("hello world")`

والذي يتضح في مفسر اللغة كما يلي:

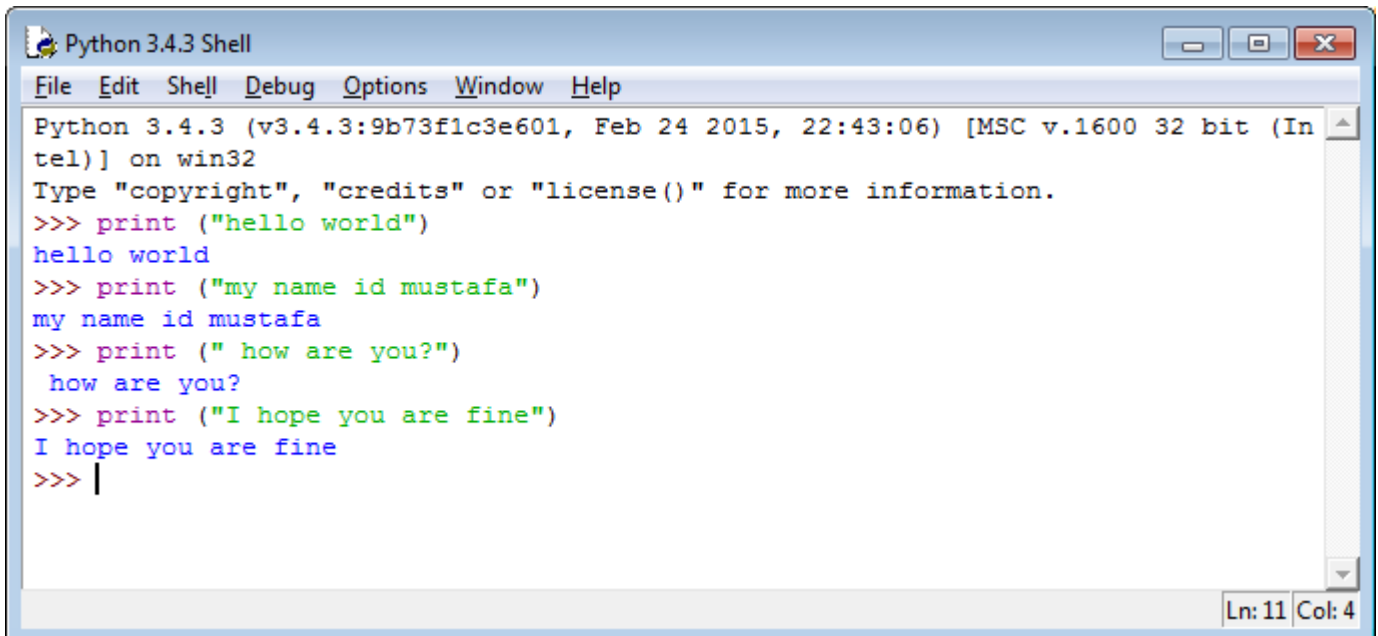


```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print ("Hello World")
Hello World
>>> |
Ln: 5 Col: 4

```

الان عرفنا كيفية طباعة سطر واحد من الكلمات ويمكن طباعاً طباعة أكثر من سطر وذلك بتكرار عبارة (Print) وكما في ادناه:



```

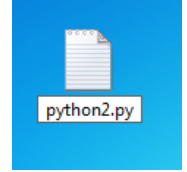
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print ("hello world")
hello world
>>> print ("my name id mustafa")
my name id mustafa
>>> print (" how are you?")
 how are you?
>>> print ("I hope you are fine")
I hope you are fine
>>> |
Ln: 11 Col: 4

```

الان قد يتساءل البعض عن كيفية طباعة عدة أسطر دفعة واحدة او تنفيذ مجموعة من الأوامر دفعة واحدة (كما هو الحال في بقية لغات البرمجة) والحل لذلك سهل جداً وهو سياقنا الذي سنتبعه في الشرح للدروس القادمة:

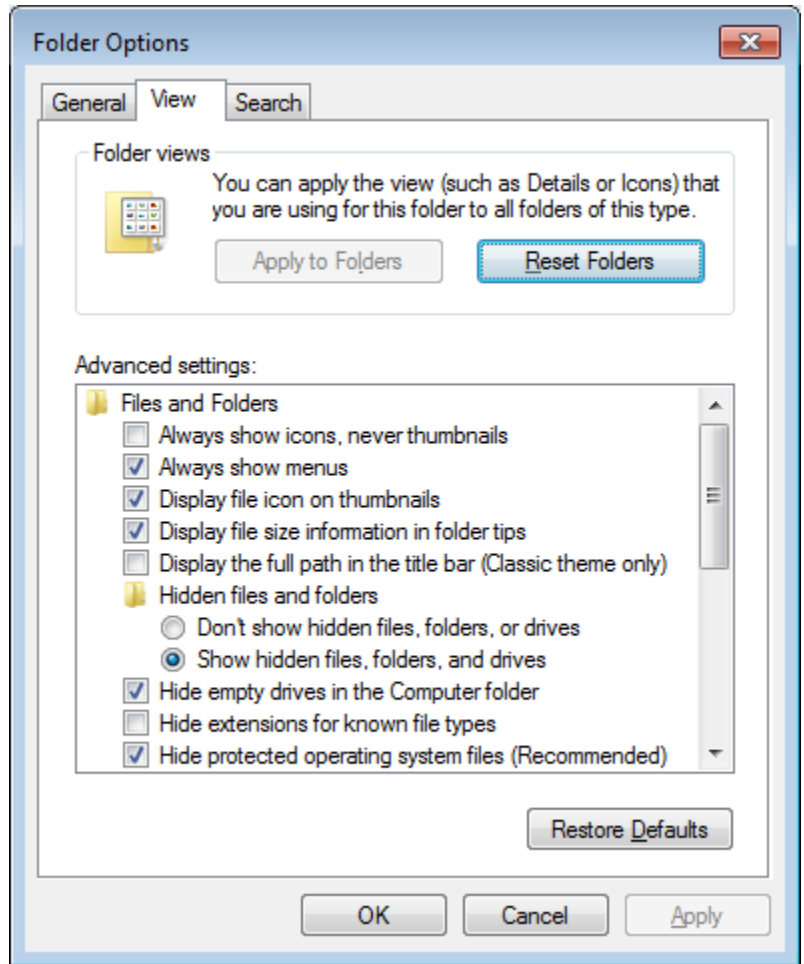
نبدأ خطوات ذلك الان:

نقوم بالنقر على مكان فارغ في سطح المكتب نقرة يمين ثم نختار (new) ثم (text document) ونسمي الملف باسم معين ونغير امتداده الى (.py) وهو امتداد ملفات البايثون وكما في ادناه:



ولمن لا يعرف كيفية عرض امتدادات الملفات فهو امر سهل جداً:

نفتح (my computer) ثم نذهب الى قائمة (tools) ومنها نختار (folder options) ثم (view) لتظهر النافذة التالية:

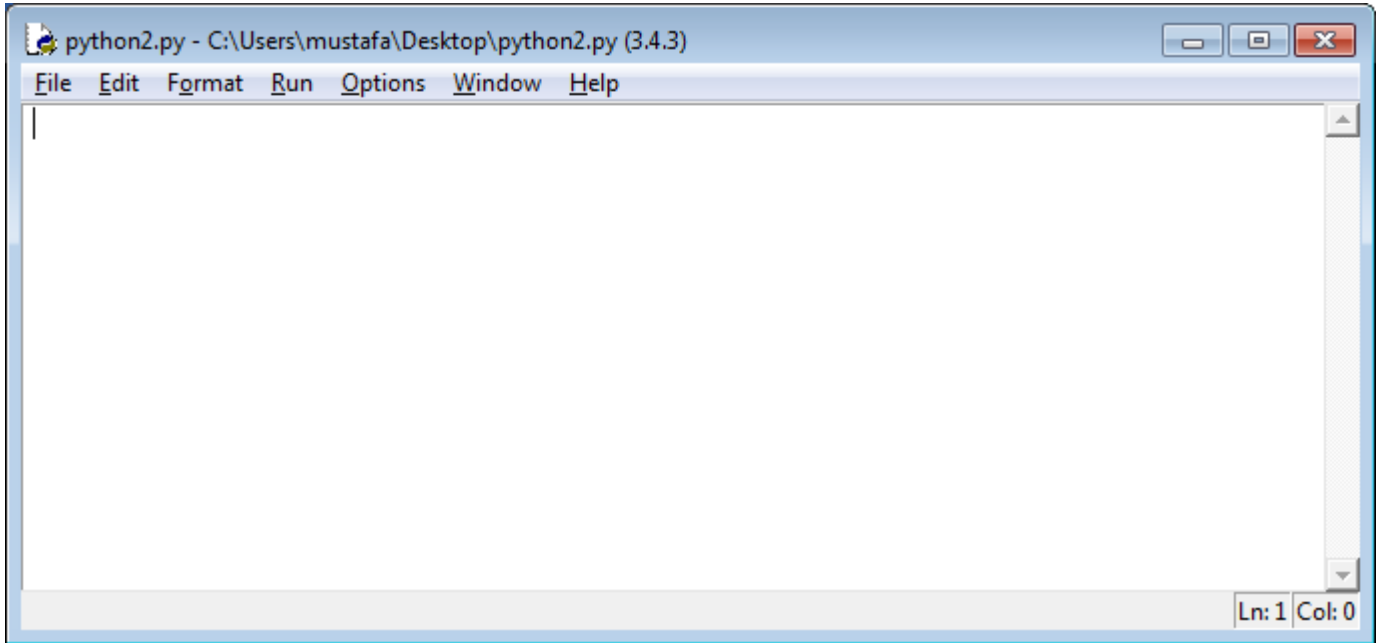


الان نقوم بإزالة علامة الصح (ان كانت موجودة) من المربع امام خيار (Hide extensions for known file types) ثم (ok) وبعدها سنلاحظ ان كل الملفات في الحاسوب ستظهر مع امتداداتها مما يسمح لنا بتغيير امتداد أي ملف بحسب

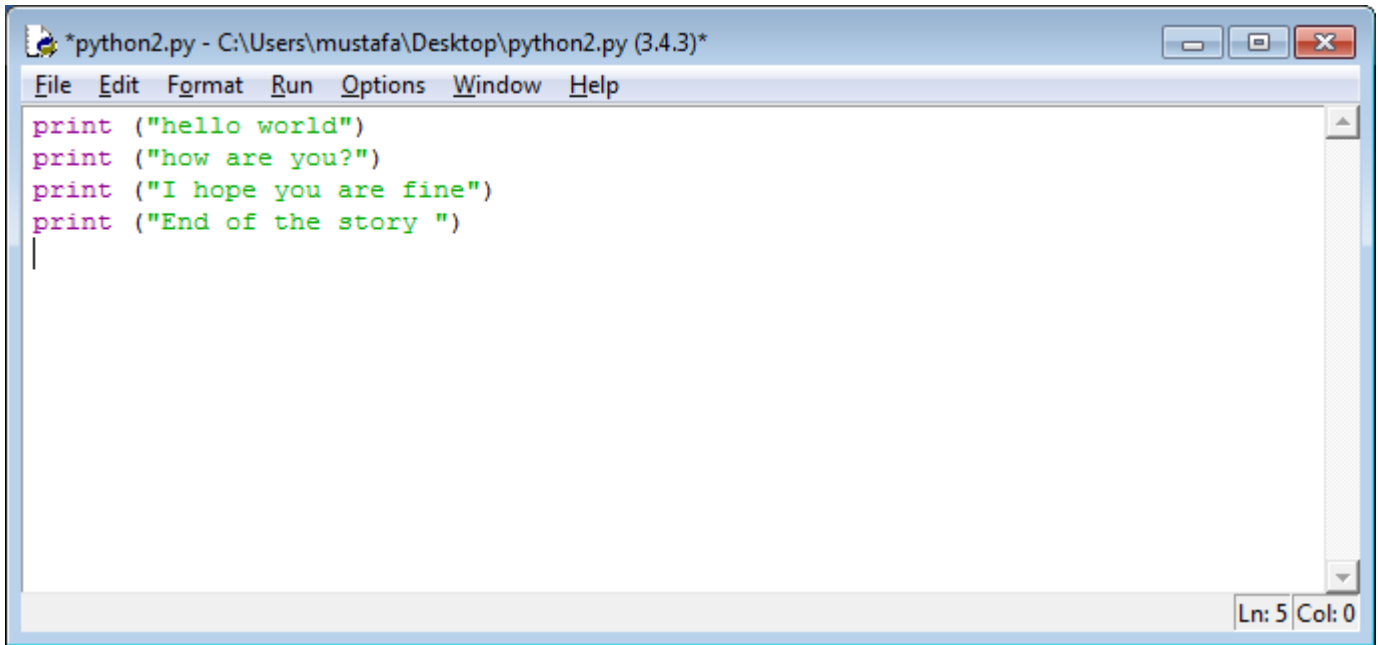
الحاجة وهو ما فعلناه هنا حيث غيرنا امتداد الملف النصي (.txt) الى ملف بايثون (.py) والذي عند فعله تظهر رسالة تطلب منا تأكيد ذلك فننقل ليكون شكل الملف النهائي كما في ادناه:



الآن نقوم بالنقر نقرة يمين على هذا الملف واختيار (edit with idle) لتفتح النافذة التالية:

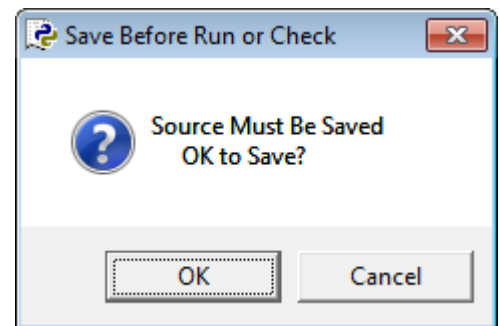


الآن نكتب أي عدد من أوامر البايثون هنا ولأننا لحد الآن لا نعرف سوى ايعاز الطباعة (print) فسنتكتب التالي:

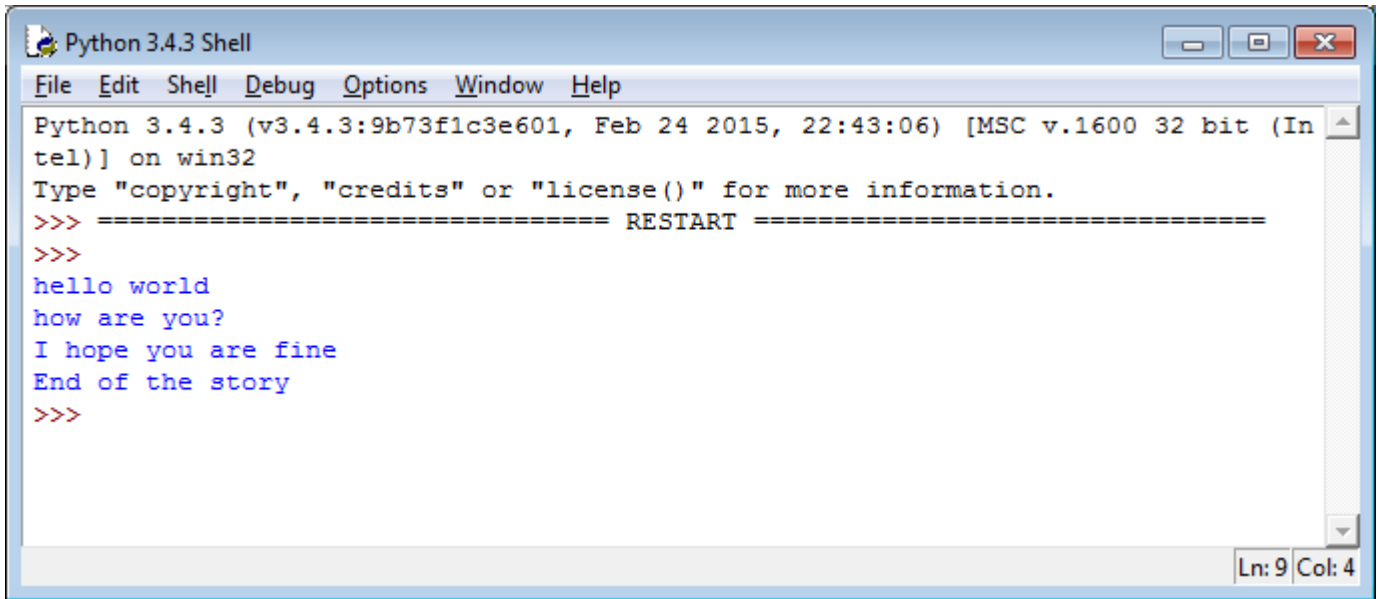


```
*python2.py - C:\Users\mustafa\Desktop\python2.py (3.4.3)*
File Edit Format Run Options Window Help
print ("hello world")
print ("how are you?")
print ("I hope you are fine")
print ("End of the story ")
|
Ln: 5 Col: 0
```

طبعاً يمكن كتابة المزيد ولكننا هنا نحاول إعطاء مثال فقط والان لتنفيذ كل هذه الايعازات دفعة واحدة نقوم بالذهاب الى قائمة (run) ثم اختيار (run module F5) لتظهر نتيجة التنفيذ التالية:



هنا يطلب منا حفظ الملف (save) حيث تتميز ملفات البايثون بعدم القابلية للتنفيذ بدون حفظ فننقر على (ok) ليتم حفظ تغييرات الملف وتنفيذه فيما بعد لتظهر النتائج التالية:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
hello world
how are you?
I hope you are fine
End of the story
>>>
Ln: 9 Col: 4
```

وهنا نرى ان النتائج جاءت بتنفيذ كل أسطر الأوامر دفعة واحدة كما هو الحال في برامجنا الاعتيادية في بقية لغات البرمجة.

الدرس الثالث (Input tools):

بعد ان شرحنا بيئة العمل في البايثون وبعض أدوات الإخراج (`print`) في الدروس السابقة نأتي اليوم لنشرح أحد أدوات الإدخال وهي أداة (`input`) والتي يمكن استخدامها بالطريقة التالية:

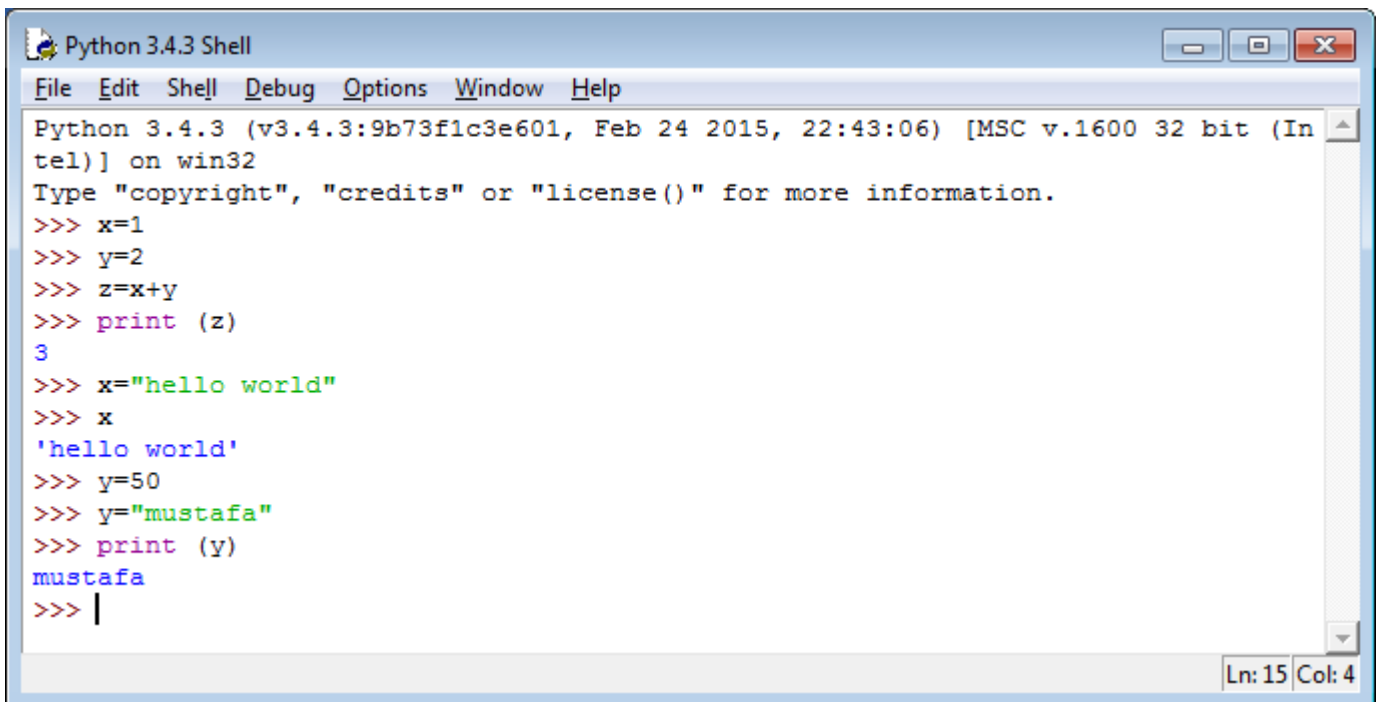
للبايثون ٣ وما بعدها `x= input ("enter your name")`

للبايثون ٢،٧ وما قبلها `x=raw_input("enter something ")`

وكما تلاحظون فإن الإدخال هنا يتم اسناده مباشرة الى متغير اسمه على سبيل المثال (`x`) ويمكن استخدام أي اسم اخر طبعاً وبنفس شروط التسمية في بقية لغات البرمجة الأخرى مثل ان يحتوي الاسم على أي تركيبية من الحروف والأرقام بشرط ان لا يبدأ برقم وان لا يحتوي علامة (`underscore`) وهي (`_`) وبقية الشروط المعروفة لكل المبرمجين.

كذلك من الأمور التي يجب ملاحظتها هنا وهي فرق رئيسي بين لغة بايثون وبقية اللغات انها لا تحتاج تعريف المتغير قبل استخدامه أي اننا نستطيع اسناد أي قيمة الى المتغير (`x`) كمثال واستخدامها واليكم المثال التالي:

نقوم بفتح ال (`idle python GUI`) ونكتب التالي:

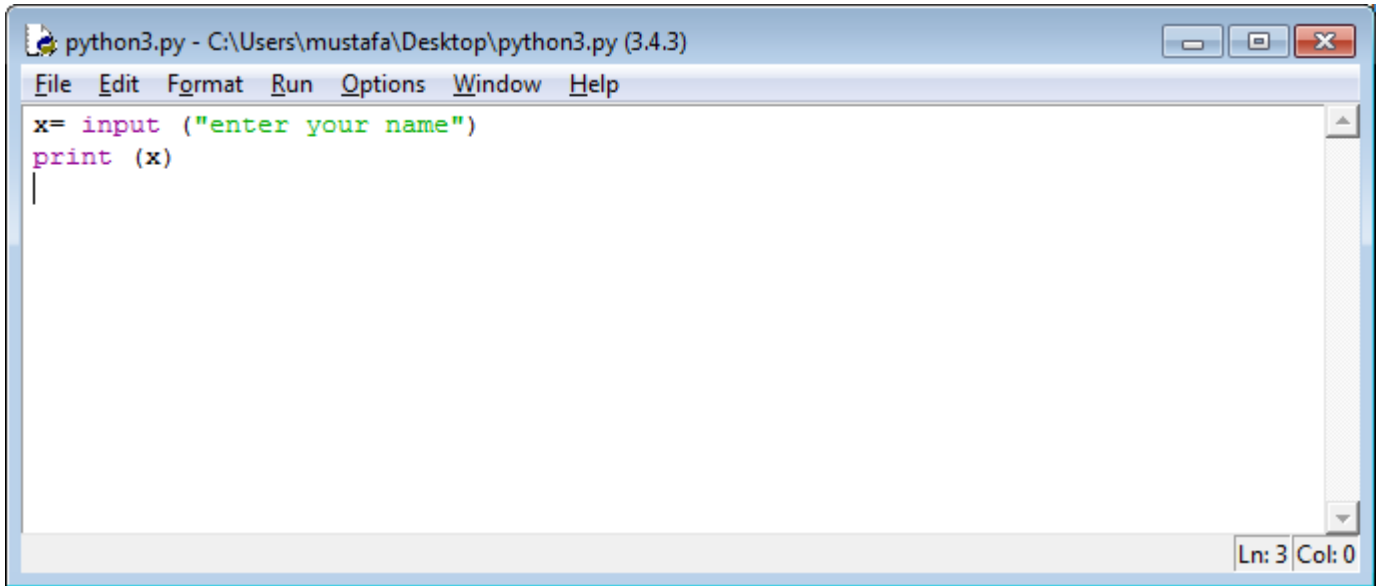


```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=1
>>> y=2
>>> z=x+y
>>> print (z)
3
>>> x="hello world"
>>> x
'hello world'
>>> y=50
>>> y="mustafa"
>>> print (y)
mustafa
>>> |
```

من هنا نلاحظ الأمور التالية:

١- إمكانية اسناد أي قيمة الى أي متغير مباشرة وبدون اعلان مسبق عن نوع المتغير.

- ٢- إمكانية إجراء العمليات الحسابية مباشرة وبدون مقدمات ولا استدعاء لمكتبات الرياضيات او غيرها كما في بقية لغات البرمجة الأخرى.
- ٣- طباعة قيمة أي متغير باستخدام ايعاز (print) في أي مكان من البرنامج.
- ٤- يمكن اسناد قيم جديدة للمتغير والذي يأخذ دائماً اخر قيمة أسندت له وينسى القيم القديمة فمثلاً ال (x) كانت قيمته الأولية ١ ثم وضعنا بداخله (أسندنا له) قيمة رمزية هي ("hello world") وحين طلبنا طباعة قيمته بالإيعاز print(x) قام المفسر بطباعة قيمة المتغير النهائية.
- ٥- يمكن طباعة قيمة أي متغير بذكر اسمه فقط ومثال ذلك حين كتبنا (x) قام المفسر بطباعة قيمته مباشرة وبدون الحاجة الى ايعاز (print) وهي خاصية مهمة جداً سنتعرف على أهميتها في الدروس القادمة ان شاء الله.
- والان نعود الى أداة الادخال موضوع الدرس ونكتب الكود التالي في ملف (.py) كما فعلنا في الدروس السابقة وكما في ادناه:

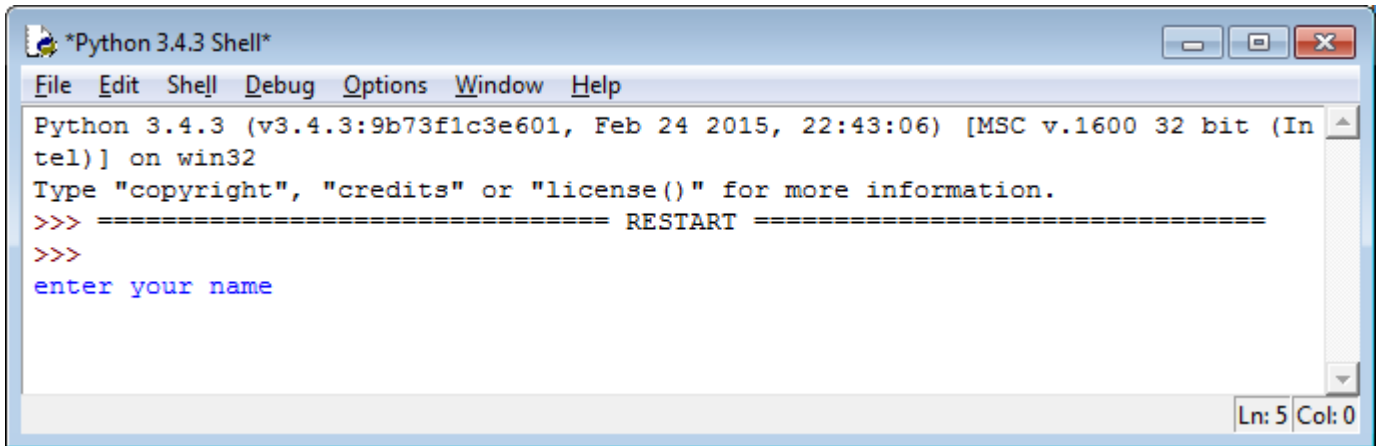


The screenshot shows a Python IDE window titled 'python3.py - C:\Users\mustafa\Desktop\python3.py (3.4.3)'. The window contains the following code:

```
x= input ("enter your name")
print (x)
```

The status bar at the bottom right indicates 'Ln: 3 Col: 0'.

الان نقوم بتنفيذ هذا الكود لنرى النتيجة:

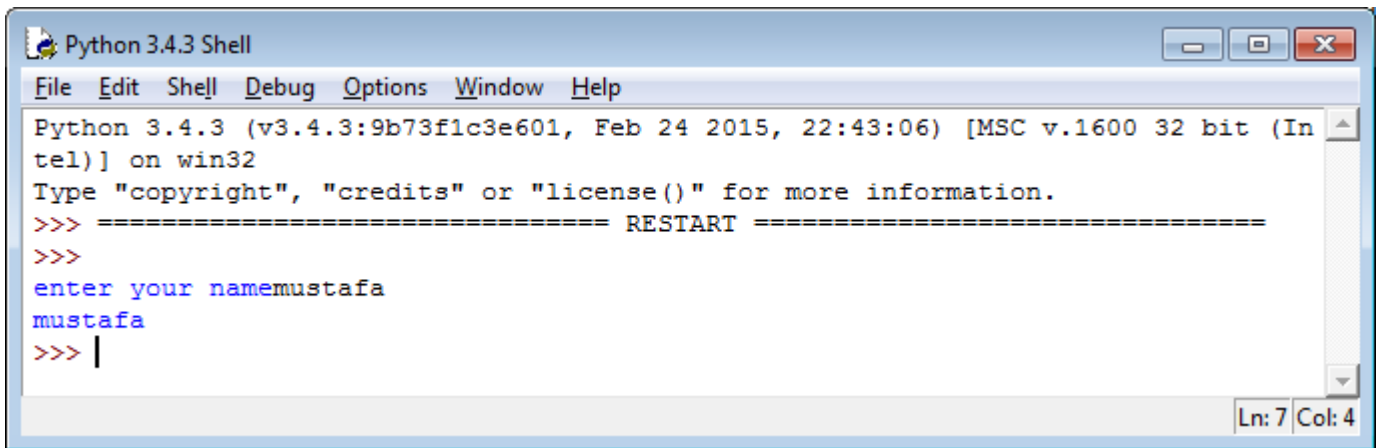


```

Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter your name
Ln: 5 Col: 0

```

هنا نرى ان المفسر قام بطباعة العبارة داخل ايعاز الادخال (input) منتظراً منا ادخال قيمة معينة ليقوم بأسنادها الى المتغير (x) فنقوم بكتابة أي شيء وننقر (enter) لنرى النتيجة:

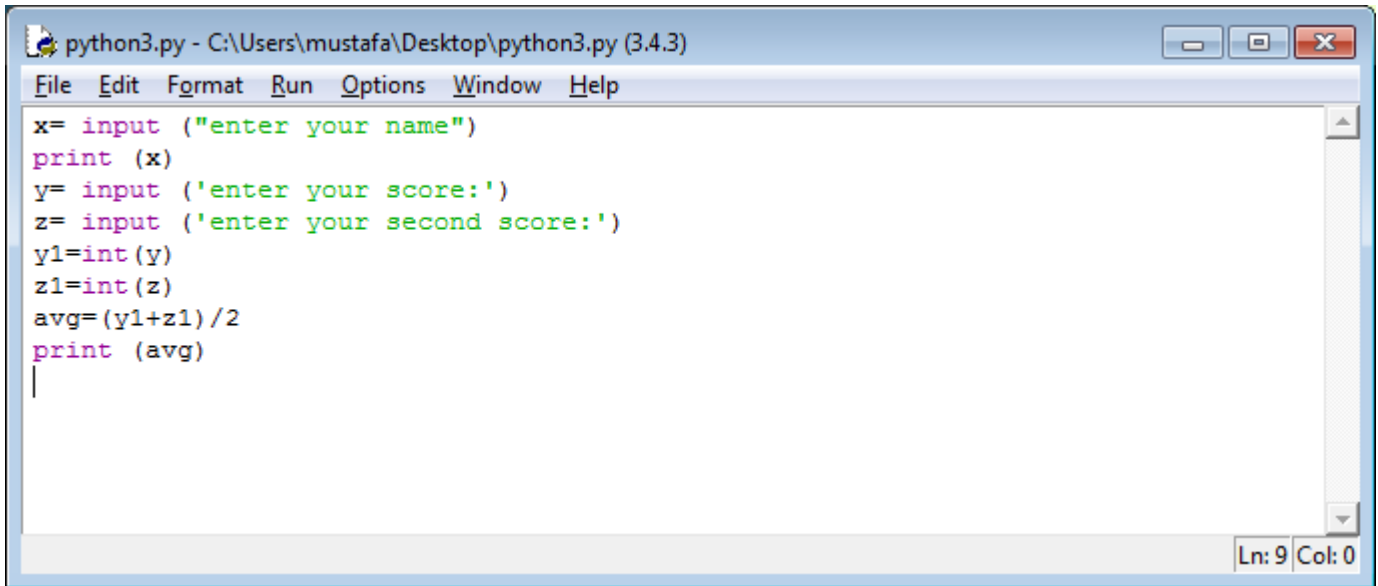


```

Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter your name
mustafa
>>> |
Ln: 7 Col: 4

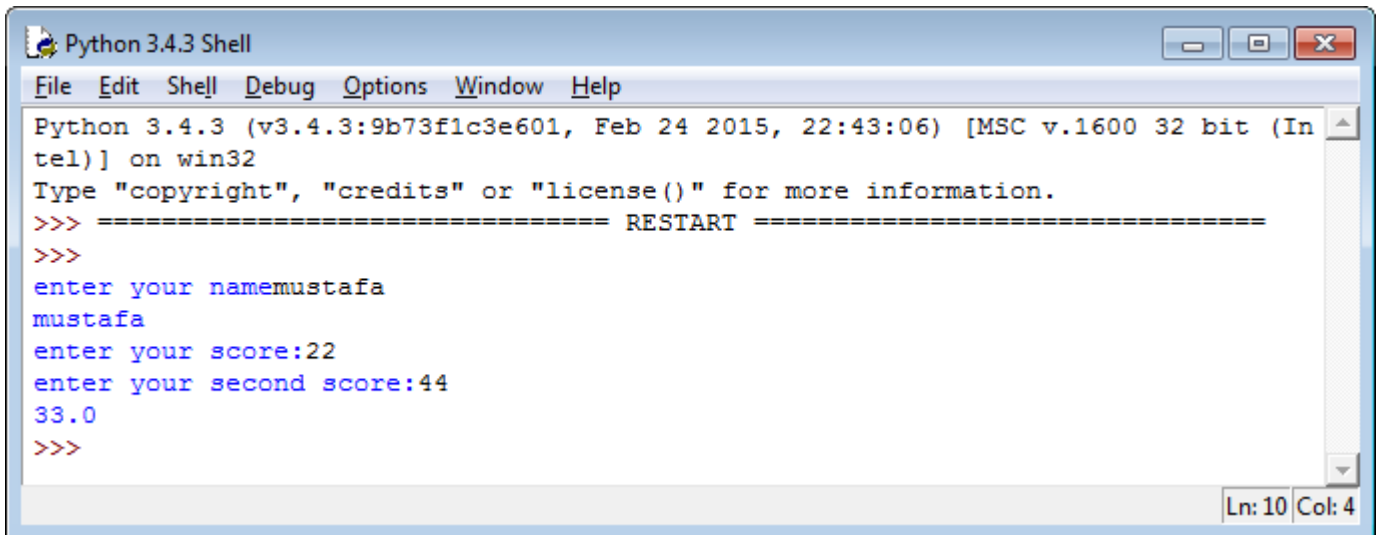
```

هنا قمنا بإدخال قيمة المتغير (x) وهي كلمة (mustafa) فقام المفسر بأسنادها الى المتغير x ثم طباعتها استناداً الى ايعاز الثاني print واليكم مجموعة ايعازات أكثر توضيحاً للفكرة:



```
python3.py - C:\Users\mustafa\Desktop\python3.py (3.4.3)
File Edit Format Run Options Window Help
x= input ("enter your name")
print (x)
y= input ('enter your score:')
z= input ('enter your second score:')
y1=int(y)
z1=int(z)
avg=(y1+z1)/2
print (avg)
|
Ln: 9 Col: 0
```

والان عند تنفيذ هذا البرنامج نجد النتائج التالية:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter your name: mustafa
mustafa
enter your score: 22
enter your second score: 44
33.0
>>>
```

ولتوضيح البرنامج أعلاه نقول:

الايغاز الأول لعرض رسالة للمستخدم تطلب منه ادخال قيمة معينة ليتم اسنادها الى المتغير x كسلسلة رمزية (String).

الايغاز الثاني لطباعة قيمة المتغير x

الايغاز الثالث لعرض رسالة للمستخدم تطلب منه ادخال قيمة معينة ليتم اسنادها الى المتغير y كسلسلة رمزية (String).

الايغاز الرابع لعرض رسالة للمستخدم تطلب منه ادخال قيمة معينة ليتم اسنادها الى المتغير z كسلسلة رمزية (String).

الايغاز الخامس هو أداة تحويل للسلسلة الرمزية المخزونة في (y) الى رقم صحيح (integer) ليخزن في المتغير y1.

الايغاز الخامس هو أداة تحويل للسلسلة الرمزية المخزونة في (z) الى رقم صحيح (integer) ليخزن في المتغير z1.

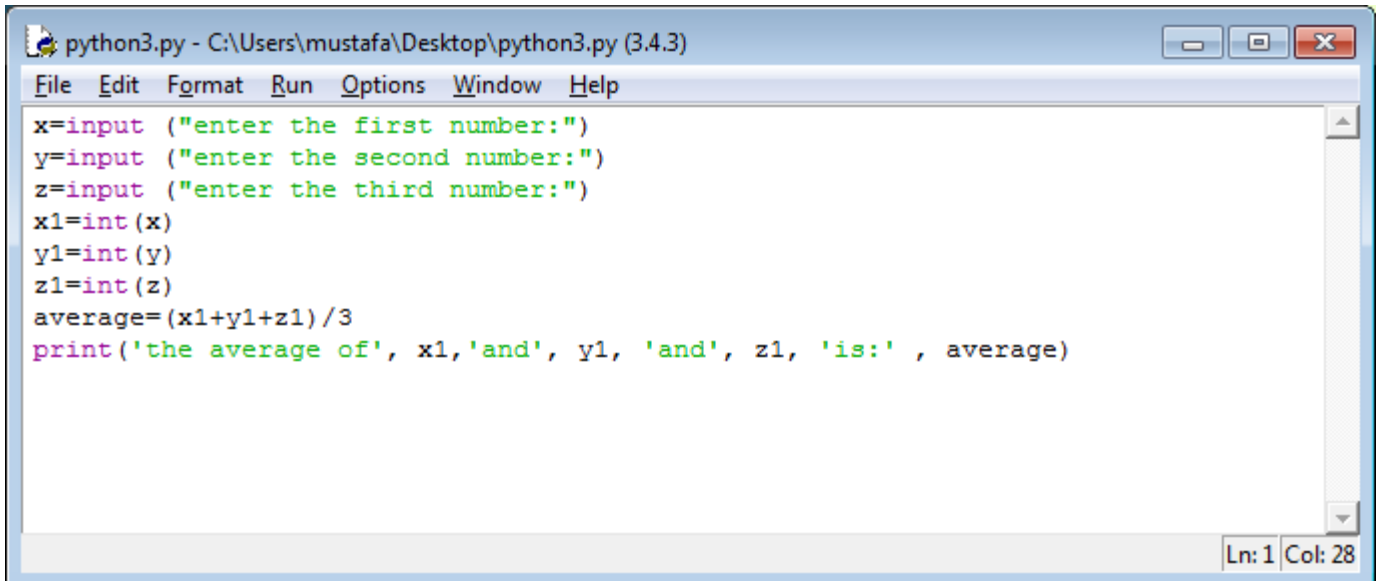
الايغاز السابع هو عملية رياضية لحساب معدل القيمتين التي تم ادخالهما ووضع الناتج في متغير يسمى avg

واخيراً الايغاز الثامن لطباعة قيمة الناتج (المعدل) المسمى (Avg) وهذا ما حصل في التنفيذ كما هو واضح أعلاه.

ملاحظة: للخروج من برنامج (IDLE Python GUI) يمكن فقط كتابة (exit ()) في سطر الأوامر او النقر (Ctrl + C) سوية من لوحة المفاتيح. اما محرر النصوص الذي قمنا بالكتابة بداخله من البداية فيجب الحرص على خزن التغييرات فيه قبل التنفيذ وقبل اغلاقه تجنباً لضياع الشفرات البرمجية التي قمنا بكتابتها.

ملاحظة أخرى: علامات الاقتباس المفردة (') وعلامات الاقتباس المزدوجة (") تستخدمان داخل ايعازات الادخال input والإخراج print بنفس الطريقة بدون أي فرق بينهما.

مثال اخر للتوضيح أكثر:



```
python3.py - C:\Users\mustafa\Desktop\python3.py (3.4.3)
File Edit Format Run Options Window Help
x=input ("enter the first number:")
y=input ("enter the second number:")
z=input ("enter the third number:")
x1=int(x)
y1=int(y)
z1=int(z)
average=(x1+y1+z1)/3
print('the average of', x1,'and', y1, 'and', z1, 'is:' , average)
Ln: 1 Col: 28
```

وعند التنفيذ تظهر النتائج التالية:

```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter the first number:12
enter the second number:33
enter the third number:44
the average of 12 and 33 and 44 is: 29.666666666666668
>>>
Ln: 9 Col: 4

```

وايضاً لتوضيح البرنامج نذكر التالي:

الاسطر الثلاثة الأولى هي مجرد ادخال لثلاث قيم وخرزنها في متغيرات تسمى x,y,z كما سبق شرحه.

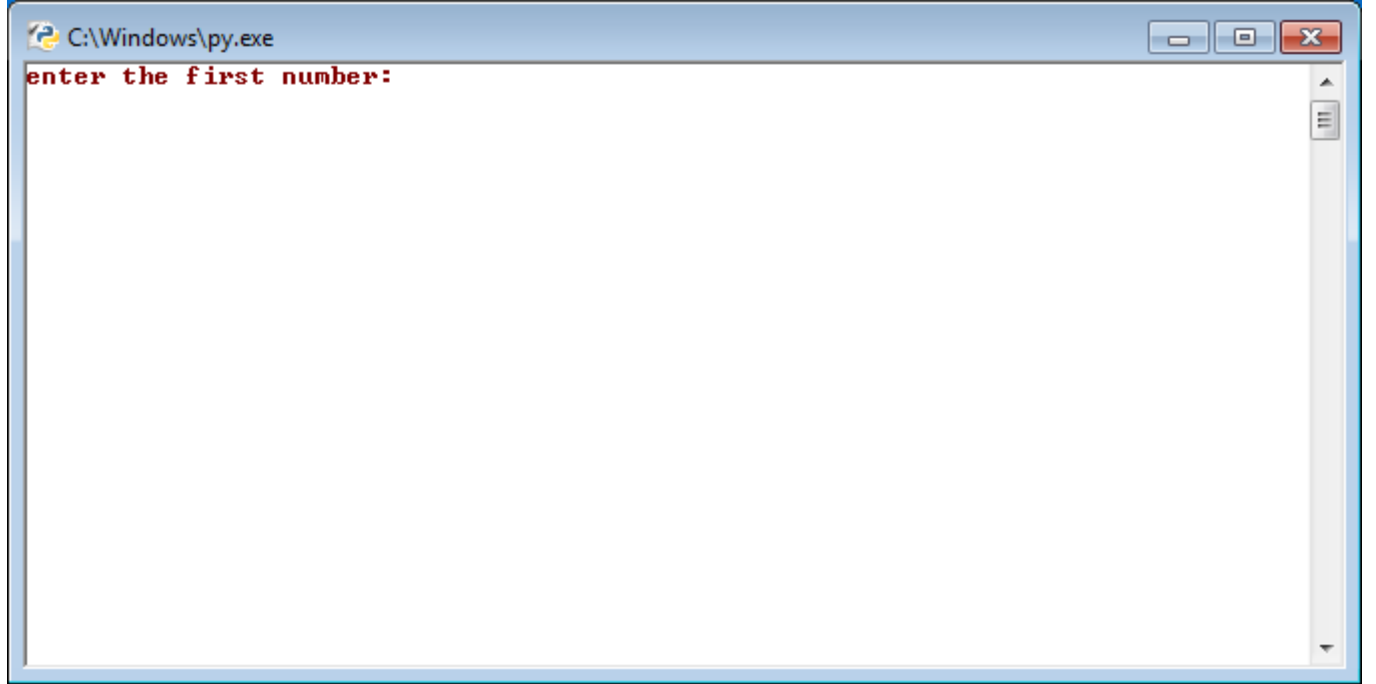
الاسطر الرابع والخامس والسادس هي لتحويل قيم تلك المتغيرات من سلاسل رمزية (string) الى قيم صحيحة (int).

السطر السابع لحساب المعدل فقط كما تم توضيحه سابقاً

السطر الأخير لطباعة عبارة الإخراج للمستخدمين والتي تحتوي الكثير من الأمور منها رسائل للمستخدم لا يتم تفسيرها من قبل المفسر (interpreter) وقد وضعت بين علامتي اقتباس (مفردة او مزدوجة فلا فرق بينهما) واما بقية الأمور التي فصلت بينها وبين الرسائل الغير مفسرة فوارز متعددة فهي قيم رياضية او منطقية او رمزية ليتم طباعة قيمها على شاشة الإخراج.

ملاحظة أخيرة:

الان وبعد ان اكملنا العمل على هذا البرنامج، نستطيع اغلاقه واغلاق مفسر البايثون (idle python gui) ثم النقر نقرتين على ملف البرنامج الذي قمنا بخرزنته في البداية تحت اسم (*.py) لنرى النتيجة التالية:



مما يعني ان برنامجنا الذي قمنا بكتابته وحفظه قد أصبح ملف قابل للتنفيذ بعيداً عن اللغة ومفسرها أي اننا نستطيع وضع أي شفرة برمجية بداخله وحفظه ليتم تنفيذه بشكل مباشر فيما بعد وهو أحد المميزات الممتازة للغة البايثون مقارنة ببقية لغات البرمجة الأخرى التي يتطلب انشاء ملف تنفيذي فيها خطوات كثيرة.

الدرس الرابع: التعليقات والتعامل مع الأرقام (comments and numbers)

بداية وككل لغات البرمجة تحتوي لغة البايثون على أدوات لإضافة تعليقات (comments) في داخل البرنامج والتي لا يقوم المفسر بتفسيرها ولا حتى عرضها عند التنفيذ بل هي تستخدم فقط لترك ملاحظات وتعليقات للمبرمج نفسه او لبقية المبرمجين المسؤولين عن تدقيق او تطوير البرنامج في المستقبل وهذه التعليقات في لغة البايثون هي ببساطة كل ما يأتي بعد علامة (#) وكما في المثال ادناه:

نفتح ال (IDLE python GUI) كما تعلمنا سابقاً ونكتب التالي ونرى نتيجة التنفيذ

```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> # this is just a comment
>>> # this is also just a comment
>>> x=1 # the value of the variable x is 1
>>> y=2 # the value of variable y is 2
>>> z=x+y # z is the sum of x and y
>>> print (z) # this instruction is used to print the value of z
3
>>> |
Ln: 10 Col: 4

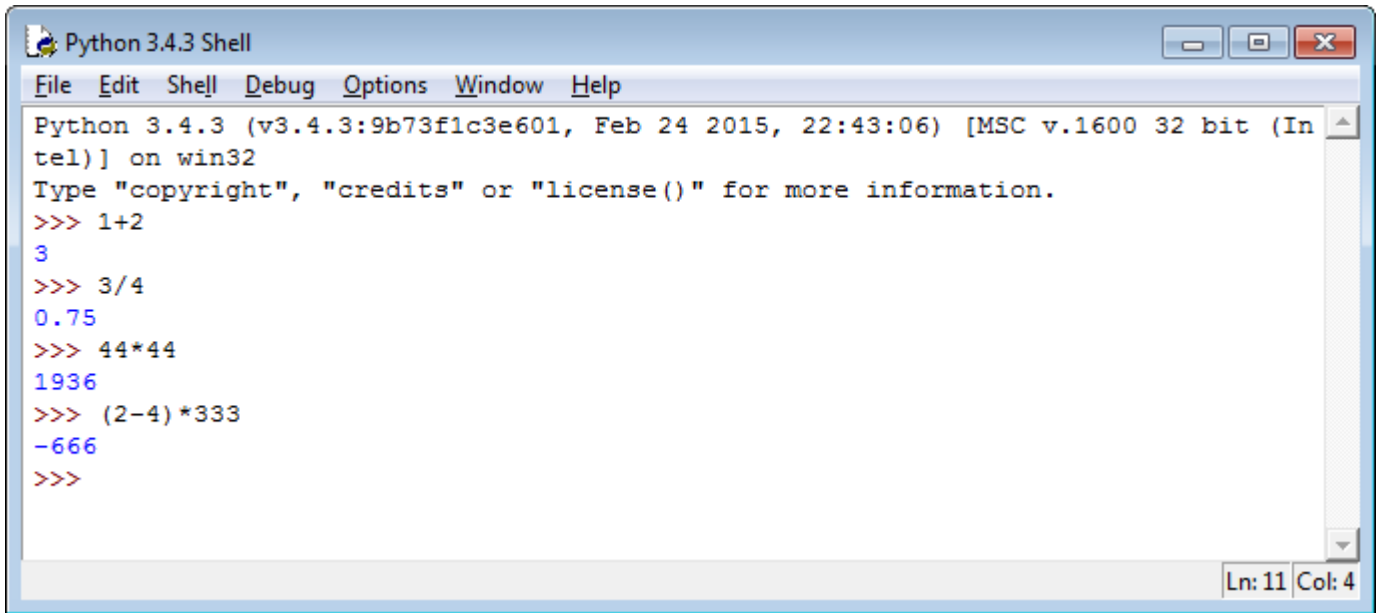
```

هنا رأينا كيف ان الكلمات التي بعد العلامة (#) لم يتم اخذها بعين الاعتبار عند التنفيذ فالمفسر لا ينظر اليها اصلاً وهي للمبرمج فقط وليس للمستخدمين وهي ذات أهمية كبيرة للمبرمجين خصوصاً للبرامج الكبيرة والمتشعبة.

الان ننتقل الى الجزء الثاني من درسنا اليوم وهو كيفية التعامل مع الأرقام في البايثون:

- استخدام البايثون كألة حاسبة:

نستطيع استخدام محرك الأوامر في مفسر البايثون كألة حاسبة مباشرة وكما يلي:



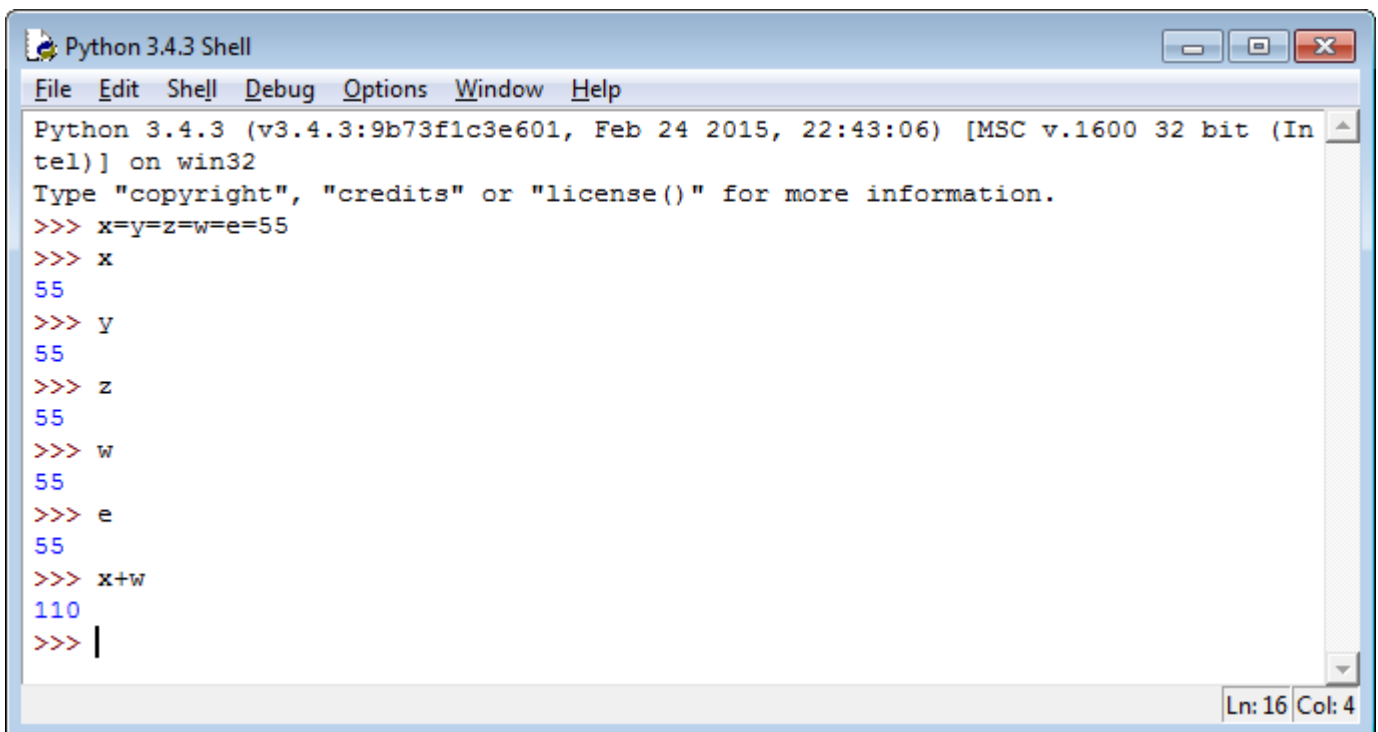
```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 1+2
3
>>> 3/4
0.75
>>> 44*44
1936
>>> (2-4)*333
-666
>>>
Ln: 11 Col: 4

```

كما نرى فقد تم تنفيذ كل الايعازات بمجرد النقر على (enter) بعرض نتائج العمليات الرياضية مباشرة مع عدم اهمال أي كسور عشرية.

كذلك من مميزات هذه اللغة عن بقية لغات البرمجة الأخرى قابلية الاسناد المتعدد وكما في ادناه:

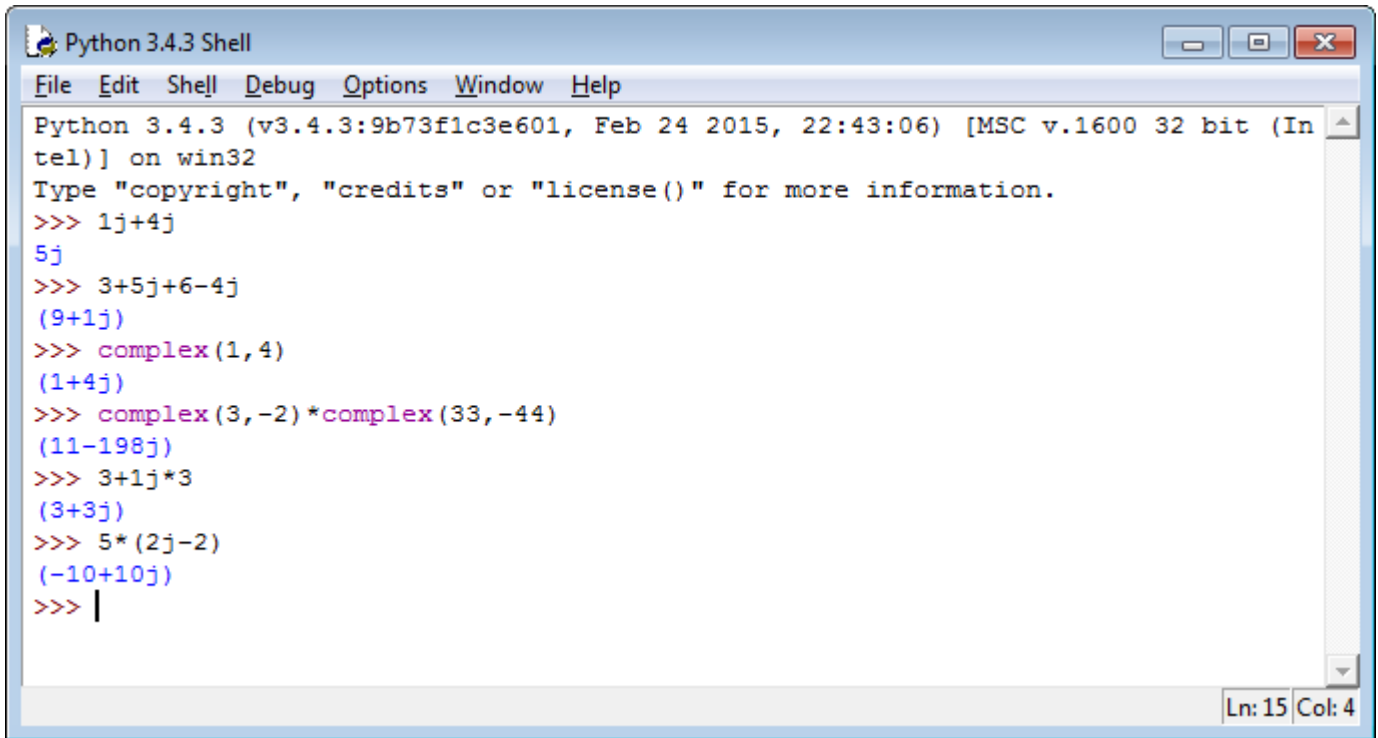


```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=y=z=w=e=55
>>> x
55
>>> y
55
>>> z
55
>>> w
55
>>> e
55
>>> x+w
110
>>> |
Ln: 16 Col: 4

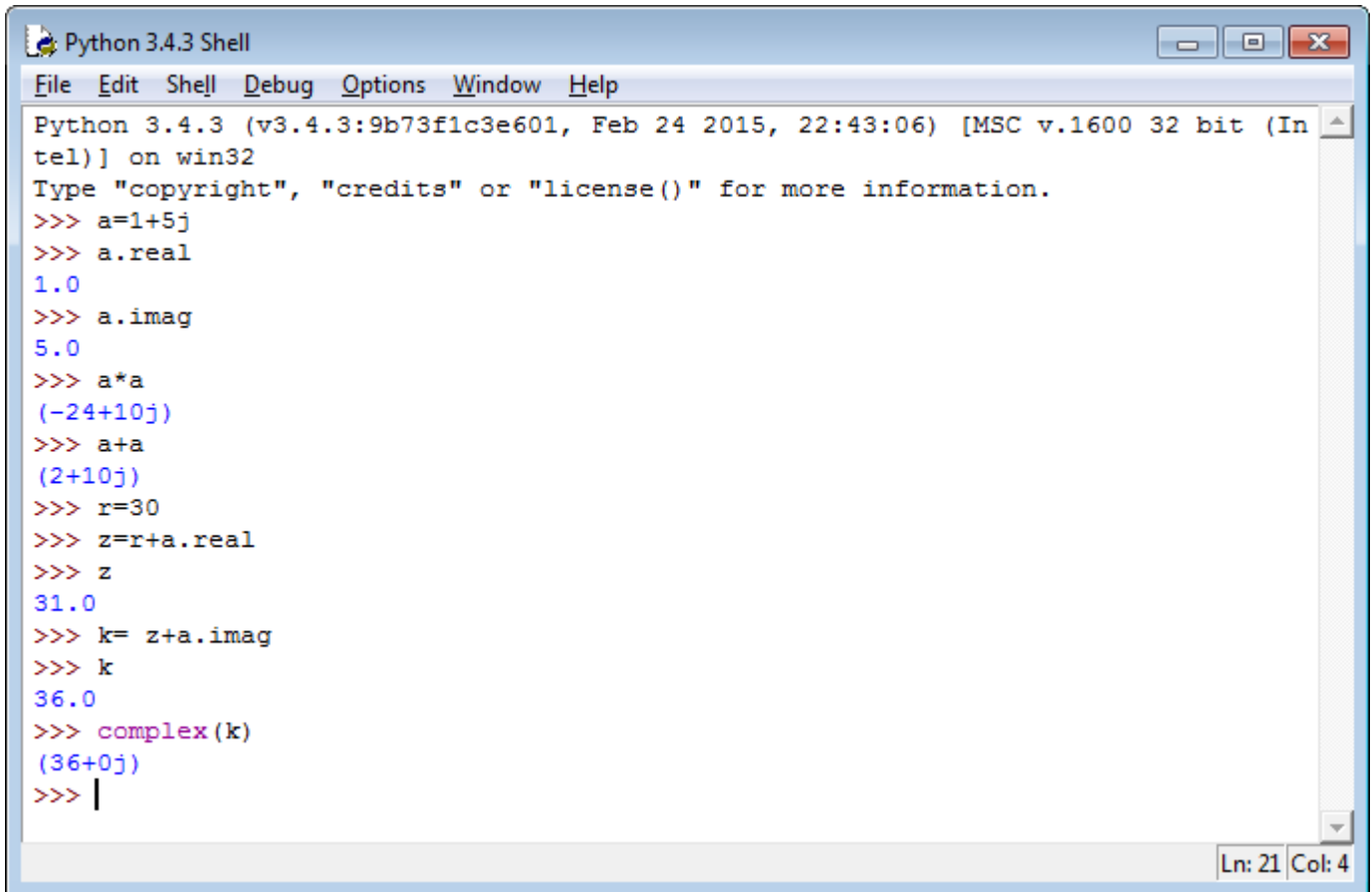
```

ويعلم مبرمجي لغات الجافا والسي بلس بلس ان هذا غير مقبول في تلك اللغات مما يميز لغة بايثون عنها في هذا المجال. كذلك يمكن التعامل مع الاعداد المركبة وهي الاعداد التي تتكون من جزئين حقيقي وتخيلي ($x+jy$) حيث يمثل ال (j) قيمة غير حقيقية تمثل جذر السالب واحد وكما في ادناه:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 1j+4j
5j
>>> 3+5j+6-4j
(9+1j)
>>> complex(1,4)
(1+4j)
>>> complex(3,-2)*complex(33,-44)
(11-198j)
>>> 3+1j*3
(3+3j)
>>> 5*(2j-2)
(-10+10j)
>>> |
```

ومن الأمثلة أعلاه نلاحظ اننا نستطيع التعبير عن الاعداد المركبة بعدة طرق واجراء كل العمليات الرياضية عليها بسهولة. كذلك يمكن التعامل مع أجزاء الاعداد المركبة الحقيقي (Real) والتخيلي (imaginary) وكما في ادناه:

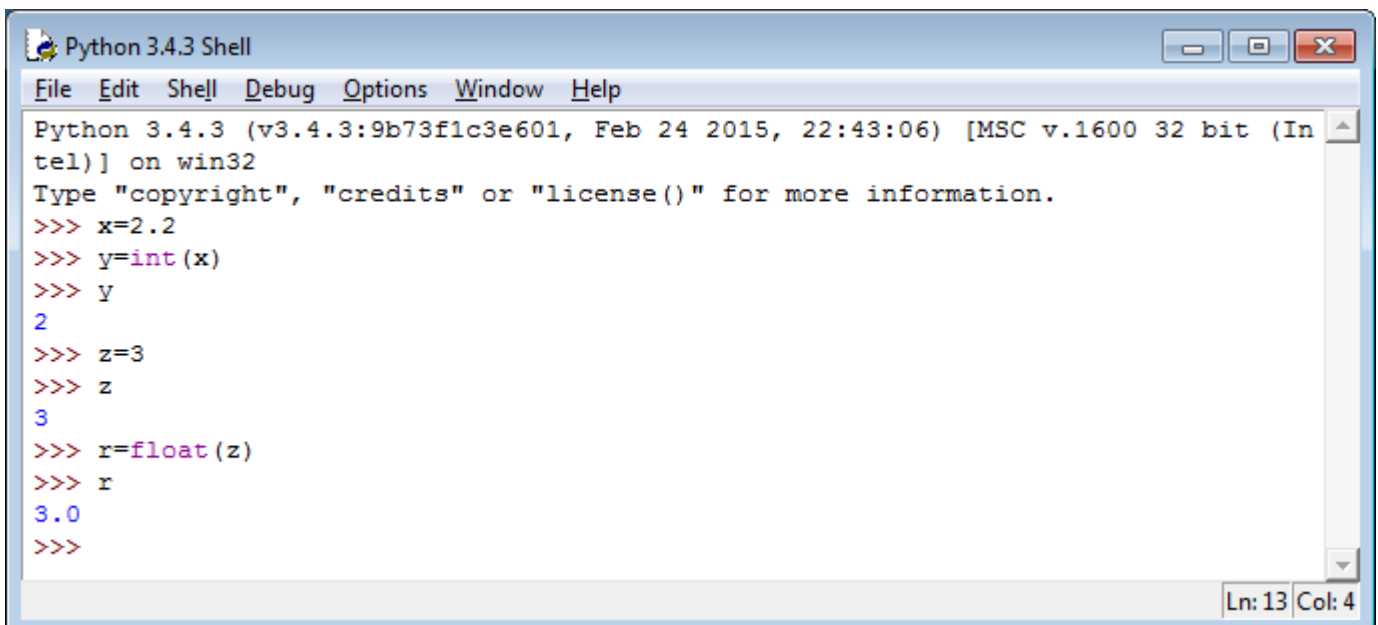


```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=1+5j
>>> a.real
1.0
>>> a.imag
5.0
>>> a*a
(-24+10j)
>>> a+a
(2+10j)
>>> r=30
>>> z=r+a.real
>>> z
31.0
>>> k= z+a.imag
>>> k
36.0
>>> complex(k)
(36+0j)
>>> |
Ln: 21 Col: 4

```

ومن هنا نرى اننا نستطيع تسميه العدد التخيلي باسم معين ثم التعامل مع جزئه الحقيقي والتخيلي كل على حدة. وكما عرفنا من الدروس السابقة فهناك عدة دوال للتحويل بين الأنواع المختلفة للأعداد ومنها (`int()`, `float()`) وكما في ادناه:

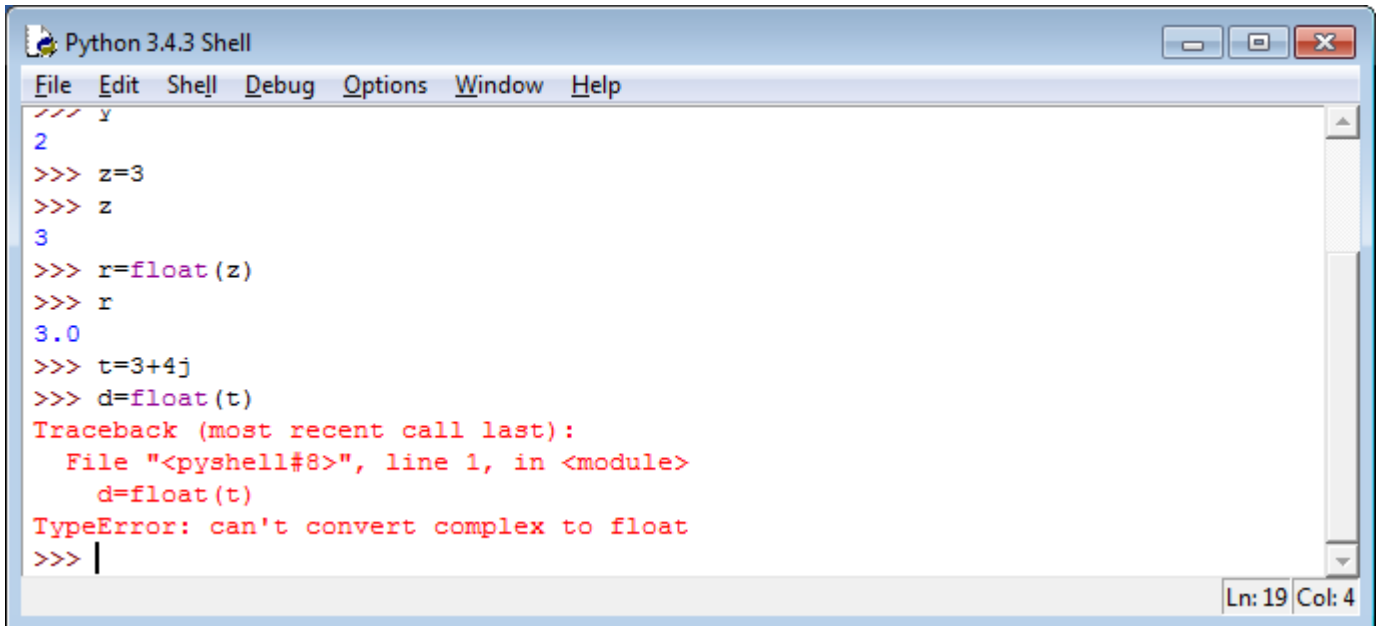


```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=2.2
>>> y=int(x)
>>> y
2
>>> z=3
>>> z
3
>>> r=float(z)
>>> r
3.0
>>>
Ln: 13 Col: 4

```

من هنا نرى اننا نستطيع تحويل الاعداد الصحيحة الى عشرية وبالعكس الا ان هذا لا يمكن مع الاعداد المركبة وكما في ادناه:

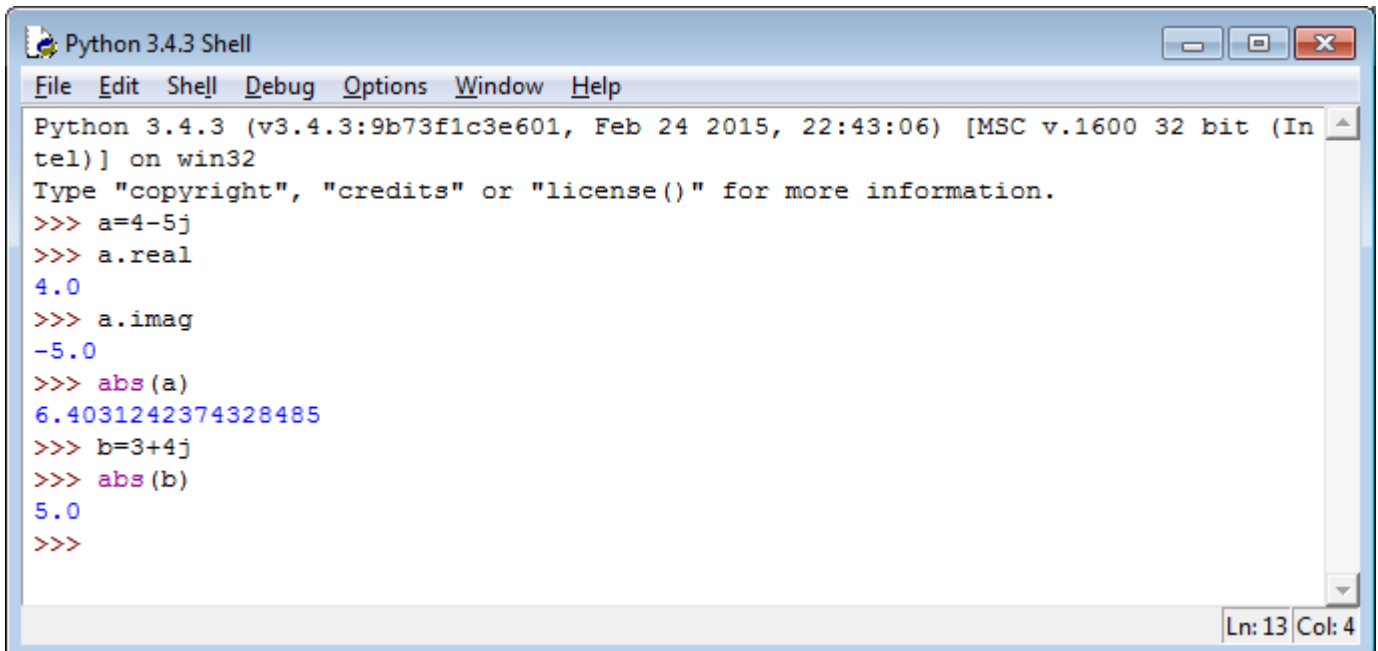


```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
>>> y
2
>>> z=3
>>> z
3
>>> r=float(z)
>>> r
3.0
>>> t=3+4j
>>> d=float(t)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    d=float(t)
TypeError: can't convert complex to float
>>> |
```

حيث نرى رسالة الخطأ واضحة بعدم قابلية تحويل الاعداد المركبة الى اعداد عشرية ولكننا نستطيع إيجاد القيمة المطلقة للعدد المركب وهي الجذر التربيعي لنتاج جمع الجزء الحقيقي تربيع مع الجزء التخيلي تربيع وكما في ادناه:

$$\text{Abs}(a)=\text{sqrt}((a.\text{real}*a.\text{real})+(a.\text{imag}*a.\text{imag}))$$

وكما في ادناه:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=4-5j
>>> a.real
4.0
>>> a.imag
-5.0
>>> abs(a)
6.4031242374328485
>>> b=3+4j
>>> abs(b)
5.0
>>>
```


ملاحظة أخرى: عند التعامل مع المفسر كحاسبة لأجراء العمليات الحسابية فإن اخر نتيجة او رقم يتم طباعته يتم حفظه في متغير خاص داخل المفسر اسمه (_) بحيث يمكن التعامل معه على انه متغير يدخل في عمليات حسابية أخرى وكما في ادناه:

```

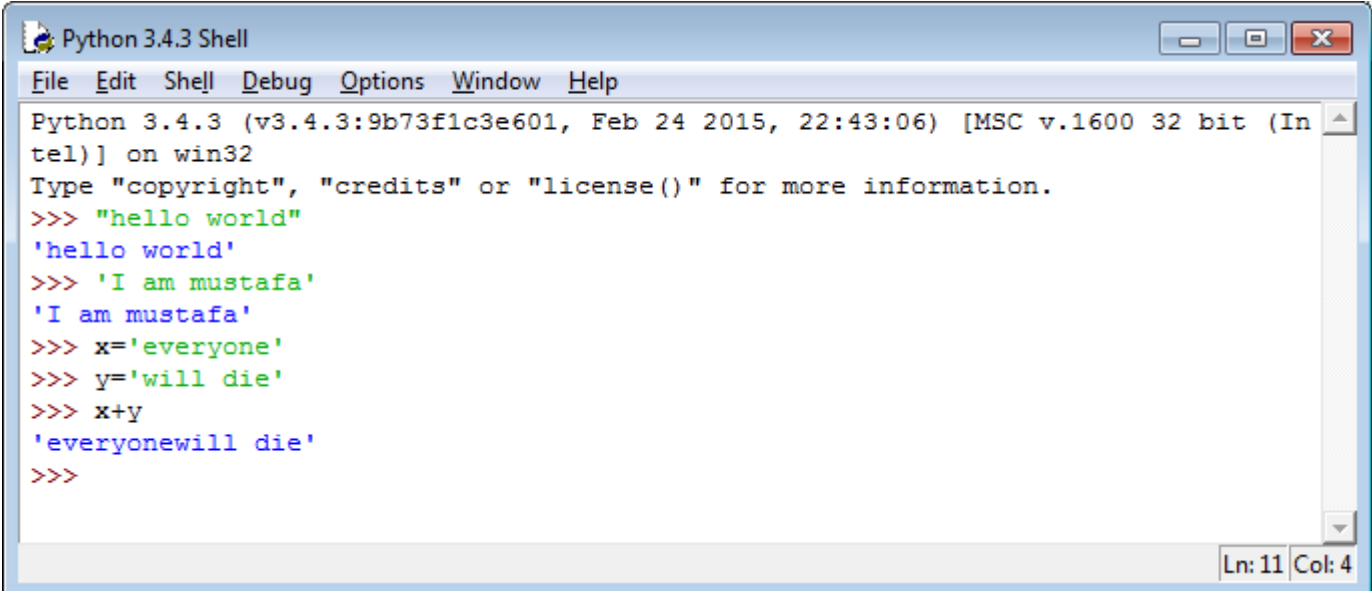
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=44
>>> b=45
>>> c=345
>>> a+b+c
434
>>> _
434
>>> a+_
478
>>> _+_
0
>>> a/_
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    a/_
ZeroDivisionError: division by zero
>>> |
Ln: 19 Col: 4

```

وكما نرى هنا فقد تم التعامل مع المتغير (_) على انه قيمة عددية تدخل في الحسابات وحين قمنا بطرحه من نفسه وكان الناتج صفر، تم خزن الصفر في نفس المتغير وحين حاولنا القسمة عليه ظهر خطأ عدم قابلية القسمة على صفر.

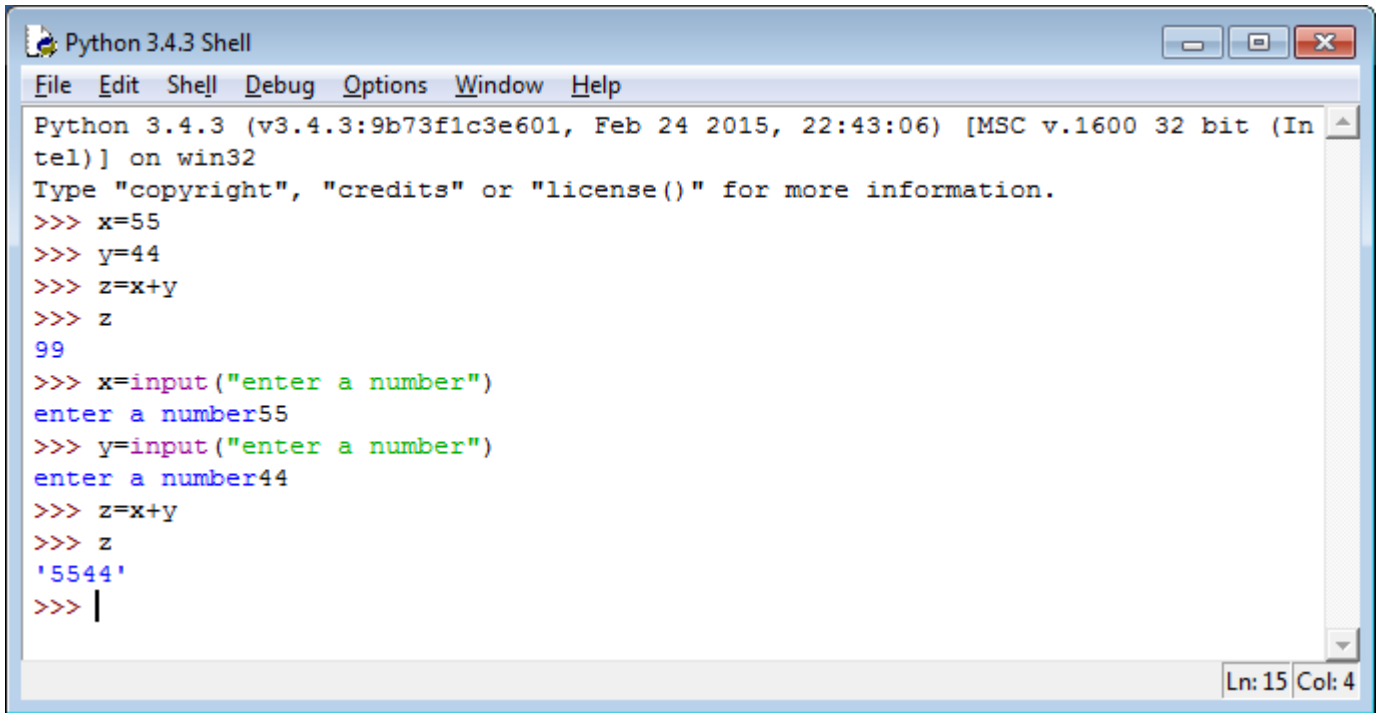
الدرس الخامس: التعامل مع السلاسل الرمزية (Strings)

بعد ان شرحنا أدوات الادخال والإخراج والتعليقات والأرقام نأتي اليوم الى كيفية التعامل مع السلاسل الرمزية وهي ببساطة أي تركيبة من الحروف والأرقام والرموز الخاصة ويتم ببساطة إدخالها الى المفسر محصورة بين علامات اقتباس مفردة (' ') او علامات اقتباس مزدوجة (" ") وكما في المثال ادناه:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> "hello world"
'hello world'
>>> 'I am mustafa'
'I am mustafa'
>>> x='everyone'
>>> y='will die'
>>> x+y
'everyonewill die'
>>>
```

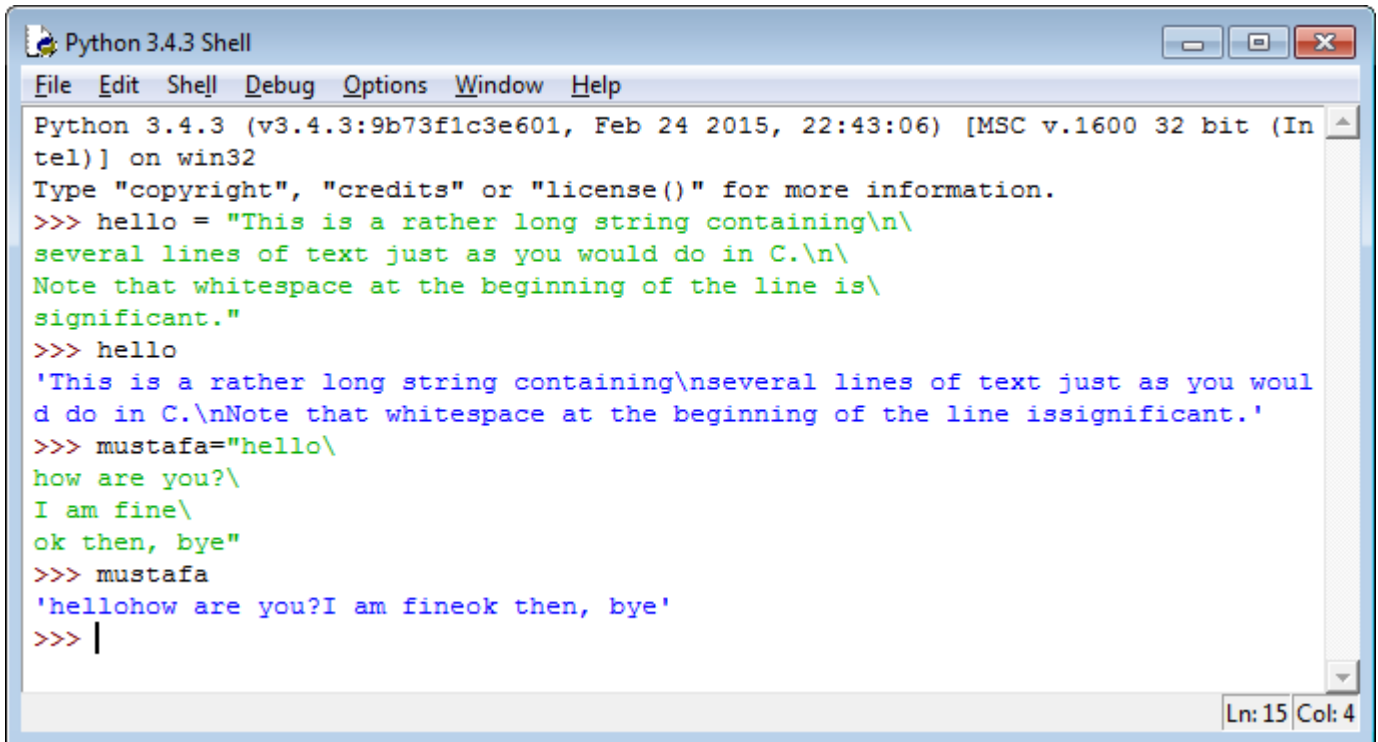
وكما نرى هنا فإن أي شيء محصور بعلامات اقتباس يتم طباعته بدون تصرف من المفسر وكذلك فإن اسناد هذه السلاسل الرمزية الى متغيرات وطباعتها مرفقة مع بعضها البعض يتم باستخدام علامة (+) والتي تمثل هنا علامة ارفاق (appending) وليست جمع حيث يتم طباعة المتغير الأول ثم الثاني وهاكم مثلاً آخر لتوضيح الفكرة:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=55
>>> y=44
>>> z=x+y
>>> z
99
>>> x=input("enter a number")
enter a number55
>>> y=input("enter a number")
enter a number44
>>> z=x+y
>>> z
'5544'
>>> |
```

هنا نلاحظ الفرق بين الادخال المباشر وبين الادخال عن طريق أداة (input) ففي المرة الأولى تعامل المفسر مع ال(x) وال (y) على انها متغيرات رقمية وفي المرة الثانية عاملها على انها سلاسل رمزية والتي يمكن تحويلها الى ارقام باستخدام الدالة التي تحدثنا عنها سابقاً (int(x)) ويمكن مراجعة الدروس السابقة للاطلاع عليها.

كذلك فإن السلاسل الرمزية يمكن ان تمتد لأكثر من سطر واحد كما هو الحال في بقية لغات البرمجة الأخرى وباستخدام (n) كما في المثال ادناه:

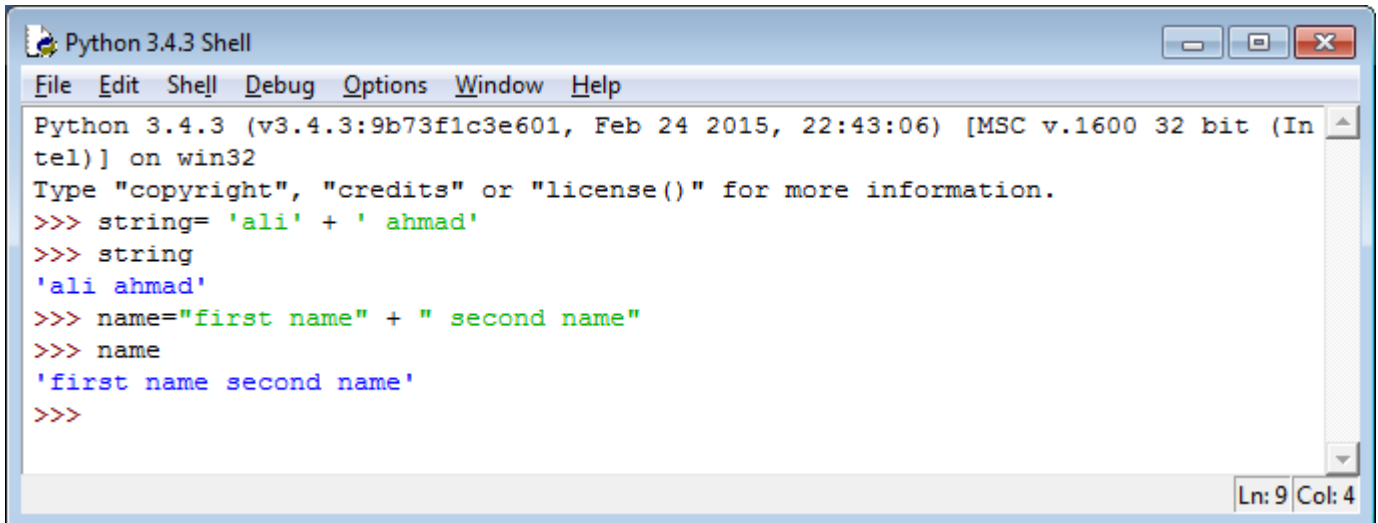


```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> hello = "This is a rather long string containing\n\
several lines of text just as you would do in C.\n\
Note that whitespace at the beginning of the line is\
significant."
>>> hello
'This is a rather long string containing\nseveral lines of text just as you would do in C.\nNote that whitespace at the beginning of the line issignificant.'
>>> mustafa="hello\
how are you?\
I am fine\
ok then, bye"
>>> mustafa
'hellohow are you?I am fineok then, bye'
>>> |
Ln: 15 Col: 4

```

كذلك يمكن ان تتكون السلسلة الرمزية من مجموعة سلاسل رمزية جزئية وكما في ادناه:



```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> string= 'ali' + ' ahmad'
>>> string
'ali ahmad'
>>> name="first name" + " second name"
>>> name
'first name second name'
>>>
Ln: 9 Col: 4

```

لاحظ ايضاً الأمثلة ادناه:

```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> string= 'ali' + ' ahmad'
>>> string
'ali ahmad'
>>> name="first name" + " second name"
>>> name
'first name second name'
>>> name*4
'first name second namefirst name second namefirst name second name'
>>> name1="hello" "world"
>>> name1
'helloworld'
>>> |
Ln: 14 Col: 4

```

الآن وكما في لغة (C) من المهم التعامل مع مكونات السلسلة الرمزية (الرموز) كل على حدة لتغيير أو تعديل أي شيء فيها ويتم ذلك بفهرسة (index) السلسلة الرمزية وهو شيء يحصل تلقائياً من قبل المفسر بحيث يكون الرمز الأول ذو فهرس صفر والعنصر الثاني ذو فهرس ١ وهكذا وكما موضح في الأمثلة أدناه:

```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> string="hello"
>>> string[0]
'h'
>>> string[1]
'e'
>>> string[5]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    string[5]
IndexError: string index out of range
>>> string[2:4]
'll'
>>> string[2:5]
'llo'
>>> string[:3]
'hel'
>>> string[3:]
'lo'
>>>
Ln: 21 Col: 4

```

والان دعونا نشرح كل سطر على حدة:

السطر الأول كان لإدخال قيمة سلسلة رمزية اسمها (string) ومحتوياتها (hello)

السطر الثاني string[0] يستخدم لطلب طباعة الرمز الأول من السلسلة (string)

السطر الثالث يستخدم لطلب طباعة الرمز الثاني من السلسلة (string)

السطر الرابع string[5] يطلب طباعة الرمز السادس من السلسلة ولأن السلسلة تتكون من ٥ رموز فقط فقد حصلنا على رسالة خطأ.

السطر الخامس يطلب طباعة الرموز من ٢ الى ٤ مع عدم شمول الرمز الرابع فقام بطباعة الرمز الثاني والثالث فقط

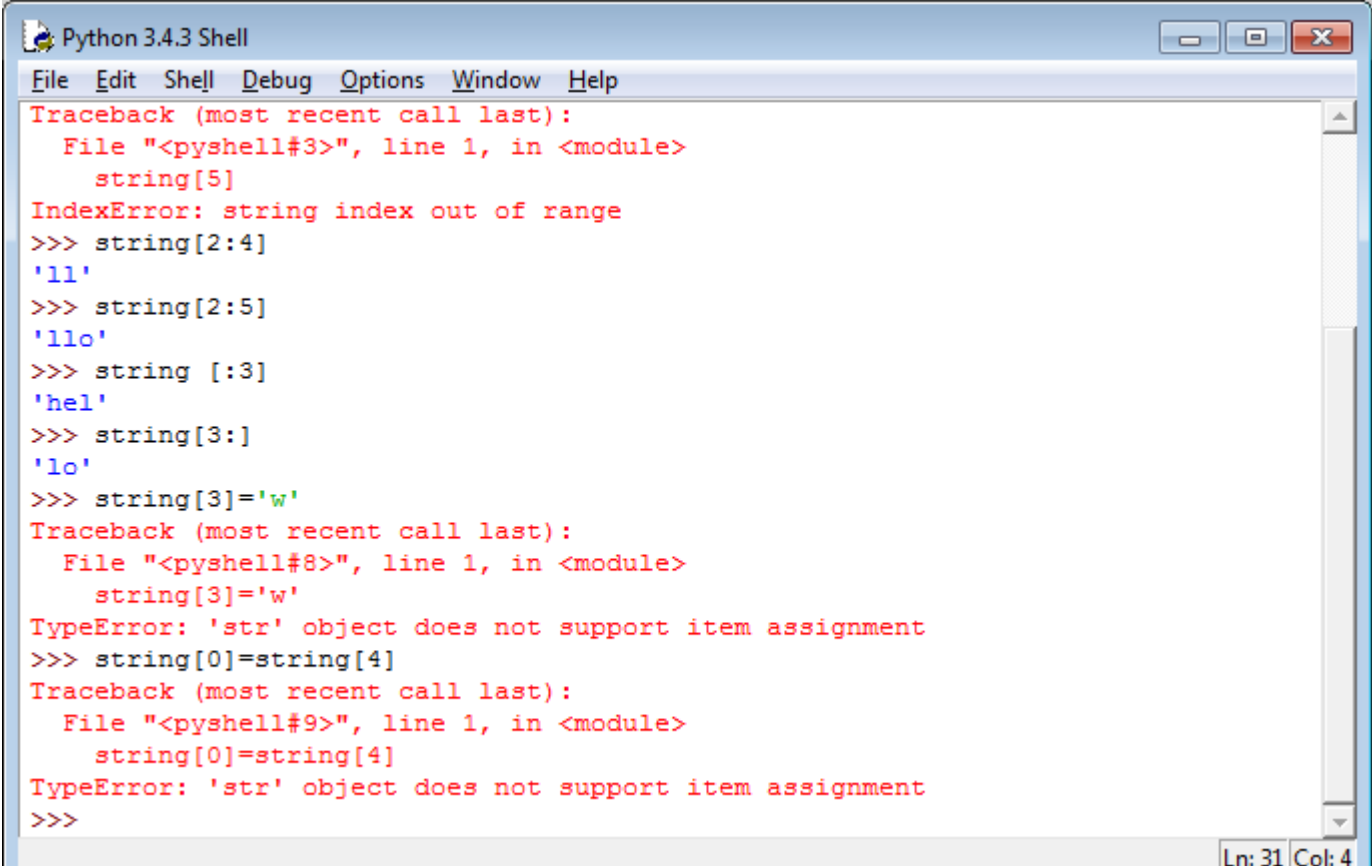
السطر السادس يطلب طباعة الرموز من ٢ الى ٥ مع عدم شمول الرمز الخامس (الغير موجود اصلاً) فقام بطباعة الرموز الثاني والثالث والرابع.

السطر السابع string[:3] يطلب طباعة كل شيء قبل الرمز الرابع (الذي فهرسه ٣)

السطر الأخير string[3:] يطلب طباعة الرمز الرابع (ذو الفهرس ٣) وما بعده.

وهكذا نستطيع تعميم كل هذه المعلومات على بقية السلاسل الرمزية الأخرى.

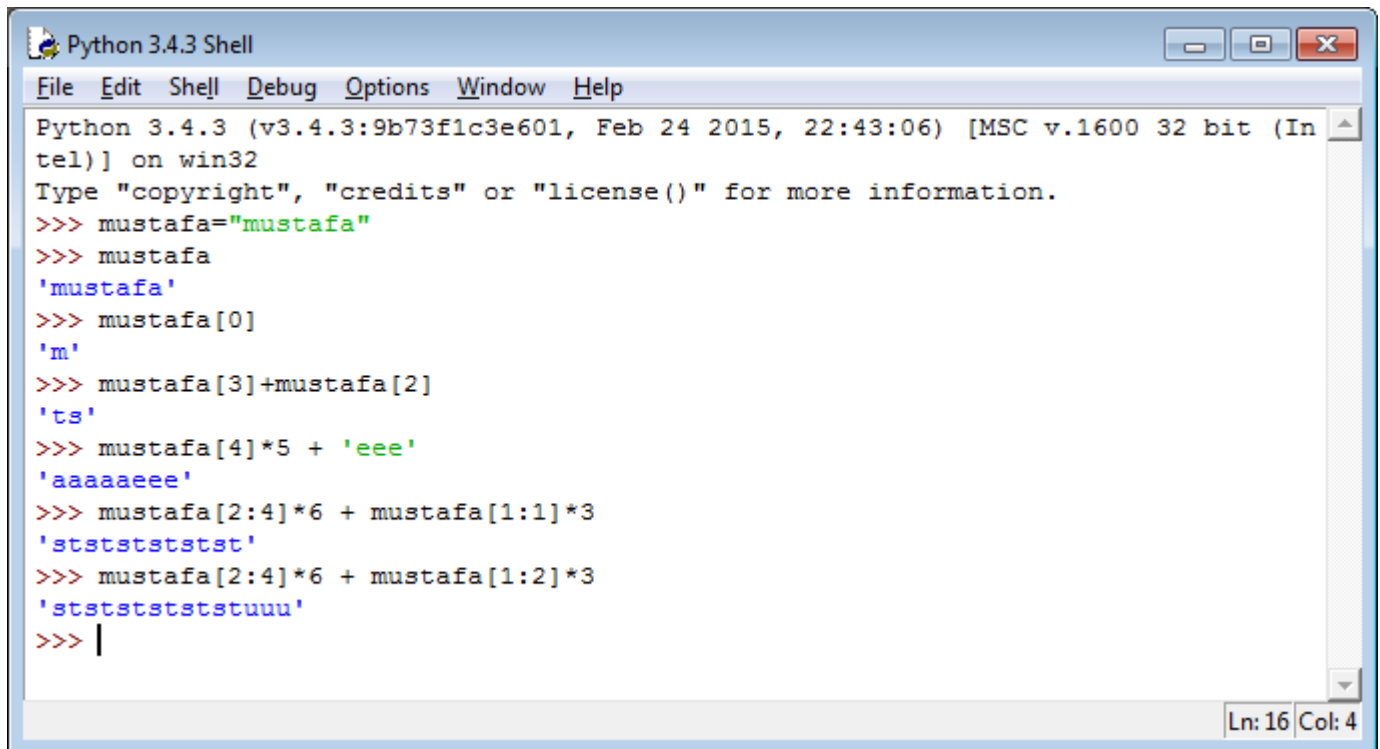
ملاحظة: على خلاف بقية لغات البرمجة فإن لغة بايثون لا تتقبل تغيير قيم رموز داخل السلاسل الرمزية وكما موضح في المثال ادناه:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    string[5]
IndexError: string index out of range
>>> string[2:4]
'll'
>>> string[2:5]
'llo'
>>> string[:3]
'hel'
>>> string[3:]
'lo'
>>> string[3]='w'
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    string[3]='w'
TypeError: 'str' object does not support item assignment
>>> string[0]=string[4]
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    string[0]=string[4]
TypeError: 'str' object does not support item assignment
>>>
```

Ln: 31 Col: 4

كذلك يمكن انتاج سلاسل رمزية جديدة باستخدام الخصائص أعلاه وكما في المثال التالي:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> mustafa="mustafa"
>>> mustafa
'mustafa'
>>> mustafa[0]
'm'
>>> mustafa[3]+mustafa[2]
'ts'
>>> mustafa[4]*5 + 'eee'
'aaaaaeeee'
>>> mustafa[2:4]*6 + mustafa[1:1]*3
'stststststst'
>>> mustafa[2:4]*6 + mustafa[1:2]*3
'ststststststuuu'
>>> |
```

Ln: 16 Col: 4

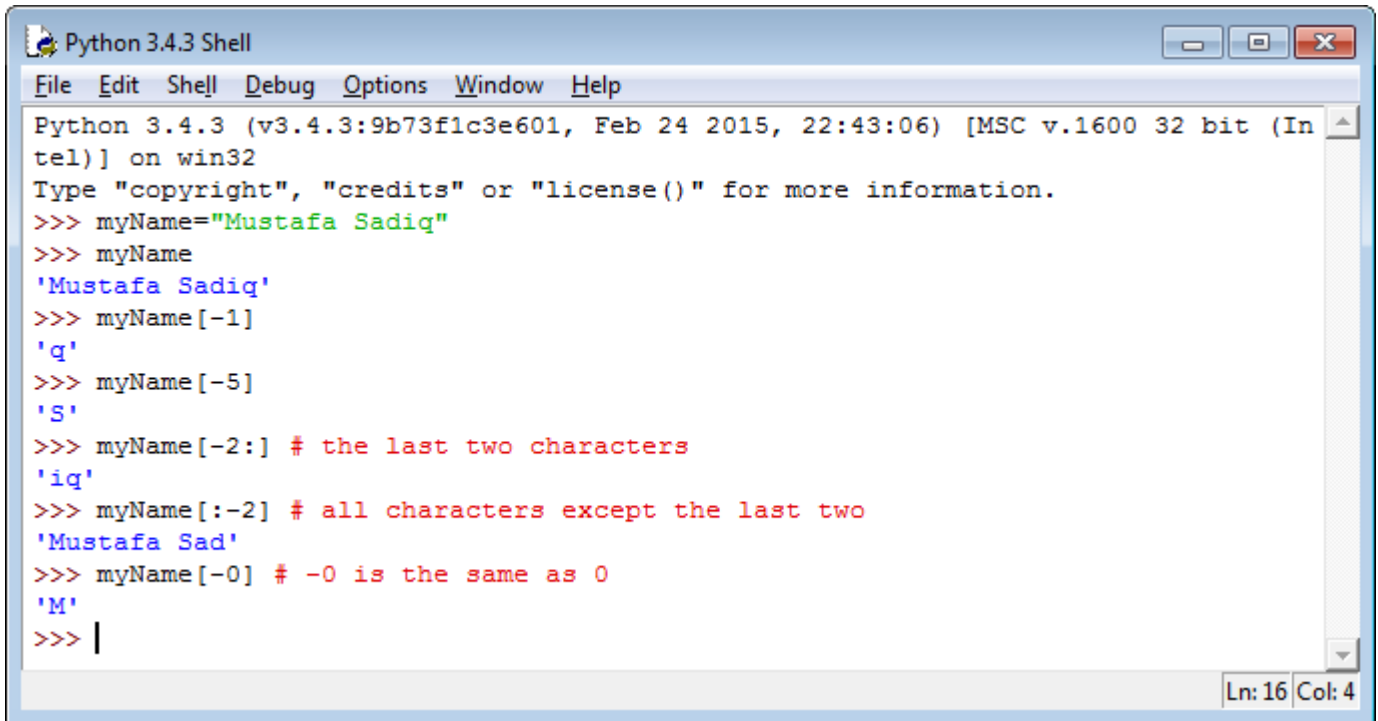
مزيد من الأمثلة على التعامل مع أجزاء السلاسل الرمزية:


```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x="123456789"
>>> x
'123456789'
>>> x[0]
'1'
>>> x[9]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    x[9]
IndexError: string index out of range
>>> x[:5] + x[5:]
'123456789'
>>> x[5:] + x[:5]
'678912345'
>>> x[1:100]
'23456789'
>>> x[:10]
'123456789'
>>> x[10:]
''
>>> x[5:4]
''
>>> |
Ln: 25 Col: 4

```

واخيراً نذكر اننا حين نكتب قيم فهارس سالبة فإننا بذلك نخبر المفسر بأن يبدأ الحساب من اليمين الى اليسار أي ان الفهرس سالب واحد يعني اخر رمز في السلسلة والفهرس سالب ٢ هو الرمز قبل الأخير وهكذا وكما في ادناه:

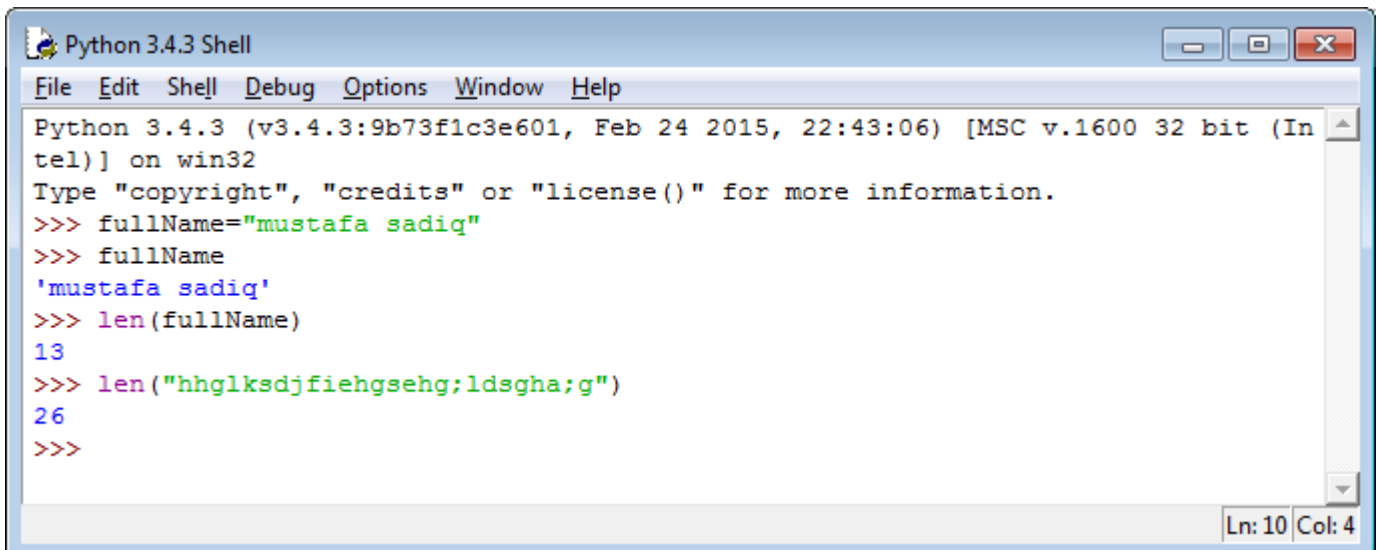


```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> myName="Mustafa Sadiq"
>>> myName
'Mustafa Sadiq'
>>> myName[-1]
'q'
>>> myName[-5]
'S'
>>> myName[-2:] # the last two characters
'iq'
>>> myName[:-2] # all characters except the last two
'Mustafa Sad'
>>> myName[-0] # -0 is the same as 0
'M'
>>> |
Ln: 16 Col: 4

```

وأخيراً يجب الحذر عند التعامل مع فهارس السلاسل الرمزية من اليمين واليسار وملاحظة الترقيم الصحيح لكل اتجاه. كما في كل لغات البرمجة الأخرى، فإن لغة البايثون توفر دالة حساب طول السلسلة الرمزية وهي `len(string)` وكما في ادناه:



```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> fullName="mustafa sadiq"
>>> fullName
'mustafa sadiq'
>>> len(fullName)
13
>>> len("hhglksdjfiehgsehg;ldsgha;g")
26
>>>
Ln: 10 Col: 4

```

الدرس السادس: الثوابت والمتغيرات (Constants and variables)

قبل البدء بمحتوى درس اليوم هناك مجموعة من الملاحظات التي يجب الانتباه لها:

١- اننا في هذه الدروس نعمل بشكل متبادل على لغات البايثون ٢ والبايثون ٣ ورغم التشابه الكبير بينهما الا ان هناك اختلافات جوهرية بين تراكيب الجمل البرمجية الخاصة بكل منهما لذا يجب الانتباه الى أي منهما قتم بتنصيبها وهو السبب في ظهور بعض رسائل الخطأ للبرامج التي نكتبها هنا.

٢- مما لا شك فيه ان كل هذه الدروس ستكون بدون فائدة من دون تطبيق لكل خطوة نشرحها والسؤال عن كل شيء غير مفهوم بل والتفكير الإبداعي فيما وراء المشروح ومحاولة تجربة كل ما يخطر ببالكم وبأبسط الأدوات بالتدرج فهكذا يتكون العقل البرمجي لديكم بالتجربة والخطأ الى حد الوصول الى مرحلة التمرس فالبرمجة هي لغة الممكن كالسياسة تماماً 😊.

٣- سيتم تطبيق بعض البرامج في بيئة الدوز في الويندوز (command prompt) في حين سيتم تطبيق البرامج الأكبر والأكثر تعقيداً كما شرحنا باستخدام برنامج (IDLE Python GUI) لذا يرجى ملاحظة الفرق بينهما عند التنفيذ.

٤- اعود وأكد على أهمية المتابعة والتطبيق والسؤال عن كل ما هو غير مفهوم ونحن هنا ننتظر اسئلتكم لتوضيح ما نحتاج اضافته الى المادة العلمية التي ننشرها لتعميم الفائدة وعدم الوقوع في نفس الأخطاء مرة بعد أخرى.

والان نبدأ درس اليوم وبعد ان شرحنا أدوات الادخال والإخراج والتعليقات وبعض الأمور الأخرى في الدروس السابقة وقبل البدء بشرح أدوات الشروط والتكرار، نود ان نذكر المعلومات البديهية (للمبرمجين القدامى والمحترفين) التالية:

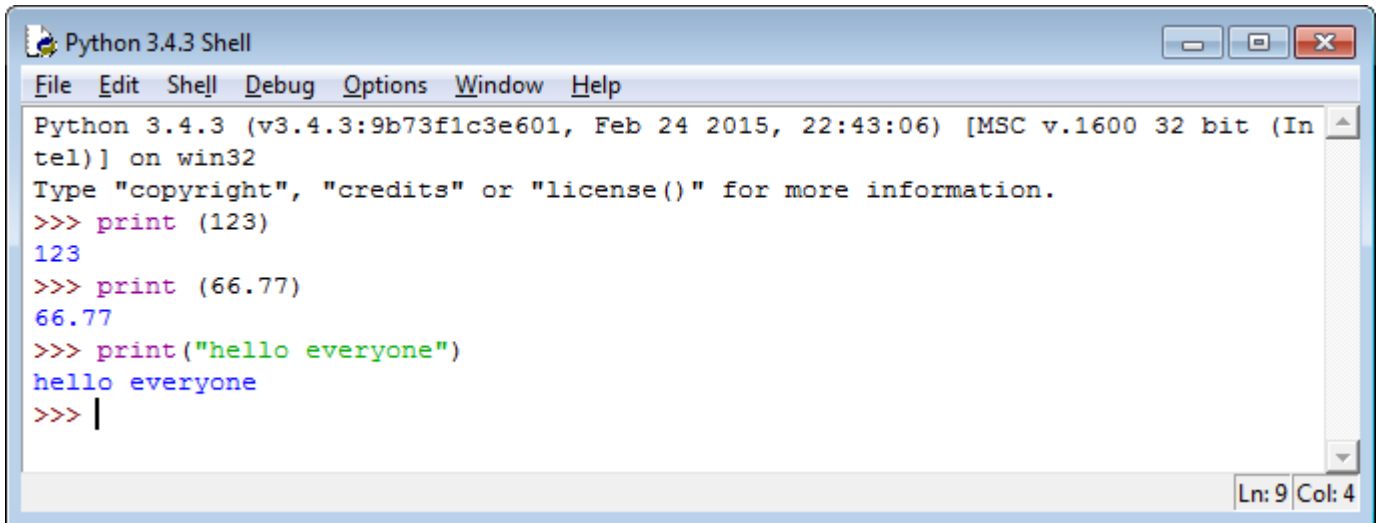
الثوابت في لغة بايثون:

وهي على نوعين:

ثوابت رقمية مثل (11, 22.4, -45) وهي الأرقام الصحيحة والكسرية الموجبة والسالبة التي يمكن إدخالها مباشرة بأسنادها الى متغيرات او عن طريق لوحة المفاتيح باستخدام أدوات الادخال (input) وغيرها.

الثوابت النصية او السلاسل الرمزية مثل ("hello world") والتي يتم إدخالها محصورة بين علامتي اقتباس مفردة او مزدوجة وكما رأينا في الدروس السابقة.

بعض الأمثلة على ادخال الثوابت في بايثون:



```

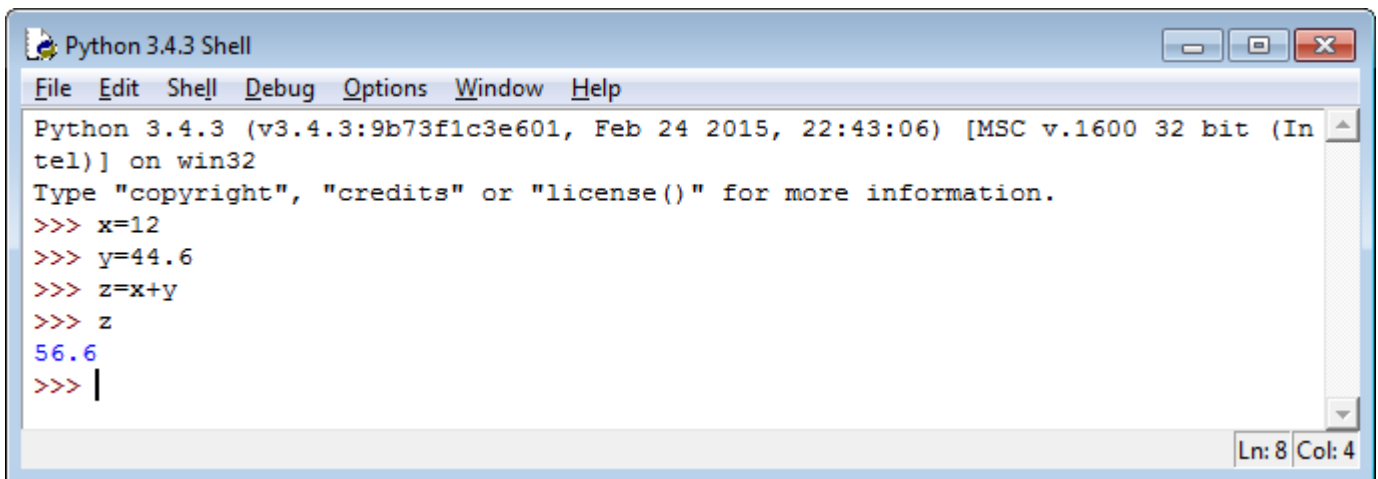
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print (123)
123
>>> print (66.77)
66.77
>>> print("hello everyone")
hello everyone
>>> |
Ln: 9 Col: 4

```

المتغيرات في لغة البايثون:

وهي الأسماء التي تطلق على مواقع تخزين البيانات في الذاكرة حيث يستطيع المبرمج تخزين واسترجاع البيانات بداخلها ويستطيع المبرمج تحديد اسم المتغير وكذلك تحديد نوعية محتوياته (بدون تصريح مسبق) على خلاف بقية لغات البرمجة الأخرى.

امثلة استخدام المتغيرات في لغة البايثون:



```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=12
>>> y=44.6
>>> z=x+y
>>> z
56.6
>>> |
Ln: 8 Col: 4

```

شروط تسمية المتغيرات: كل لغات البرمجة فيها شروط معينة لتسمية المتغيرات وهي تقريباً متشابهة في كل لغات البرمجة وتتمثل التالي:

- يجب ان يبدأ اسم المتغير بحرف او علامة (underscore _).
- يجب ان يحتوي حروف وأرقام ورمز (underscore _) فقط ولا يحتوي أي رموز خاصة أخرى

- يجب الانتباه الى ان لغة بايثون حساسة لحالة الحروف (case sensitive) أي ان المتغير الذي اسمه (Ali) يختلف عن المتغير الذي اسمه (ALI) وكذلك عن المتغير (ali) فيجب الانتباه.
- يفضل ان تكون أسماء المتغيرات قريبة من معانيها الحقيقية لتسهيل قراءة البرنامج من قبل مبرمجه وغيره من المبرمجين وفي حالة كثرة المتغيرات وتشابهها يفضل استخدام التعليقات (comments) لتوضيح معانيها.

امثلة على أسماء المتغيرات المقبولة وغير المقبولة:

مقبول: _saad Qasim22 ali o1k2j3

سيء: Alaa ALAA alaa لأنها متشابهة وتسبب مشاكل للمبرمجين

غير مقبول: 123all لأنه يبدأ برقم. *Ali*: لأنه يحتوي رمز خاص (*) وهكذا.

- الشرط الأخير لأسماء المتغيرات ان لا تكون أحد الكلمات المحجوزة للغة وهي التالية:

Reserved Words

- You can not use reserved words as variable names / identifiers

and del for is raise
assert elif from lambda return
break else global not try
class except if or while
continue exec import pass yield
def finally in print

تتكون برامج البايثون بصورة عامة من أسطر برمجية وتختلف أنواع هذه الاسطر البرمجية باختلاف مكوناتها وهي كما يلي:

Sentences or Lines

`x = 2` ← Assignment Statement

`x = x + 2` ← Assignment with expression

`print x` ← Print statement

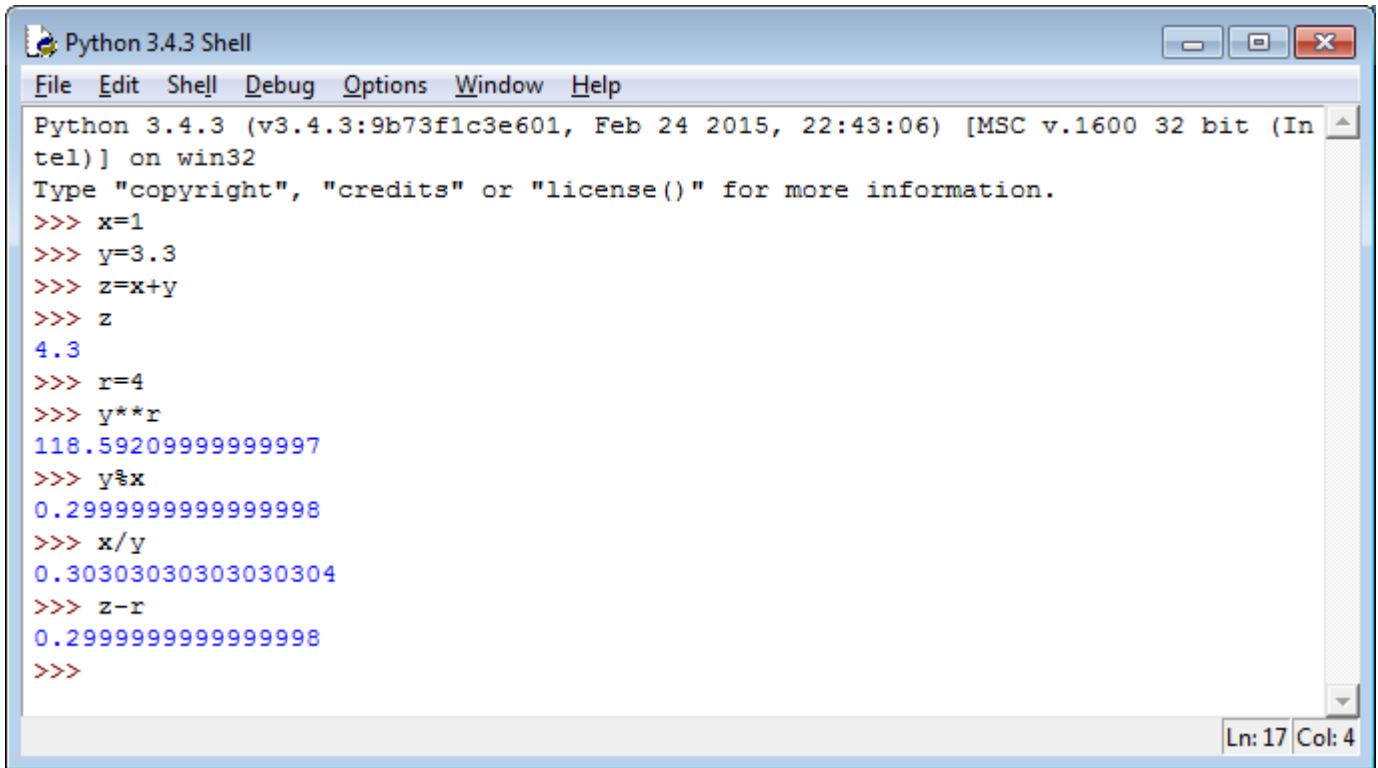
Variable
Operator
Constant
Reserved Word

حيث تكون بعض الجمل البرمجية هي عبارات اسناد فقط مثل ($x=2$) وبعضها تكون جمل اسناد وتعابير رياضية مثل ($y=x+2$) وبعضها عبارات ادخال او طباعة او شروط او تكرار وكما سنرى في الدروس القادمة ان شاء الله.

العمليات الرياضية الأساسية للغة البايثون:

الرموز	العمليات
+	الجمع
-	الطرح
*	الظرب
/	القسمة
**	الاس
%	باقي القسمة

امثلة على تطبيق العمليات الرياضية في بايثون:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=1
>>> y=3.3
>>> z=x+y
>>> z
4.3
>>> r=4
>>> y**r
118.59209999999997
>>> y%x
0.29999999999999998
>>> x/y
0.30303030303030304
>>> z-r
0.29999999999999998
>>>
```

تسلسل تنفيذ العمليات في لغة بايثون: كما في كل لغات البرمجة الأخرى:

- الأقواس
- الأسس
- الضرب والقسمة
- الجمع والطرح
- من اليسار الى اليمين

مثال:

```

>>> x = 1 + 2 ** 3 / 4 * 5
>>> print x
11
>>>

```

Parenthesis
 Power
 Multiplication
 Addition
 Left to Right

ملاحظة: لتبسيط الأمور على المفسر والقاري فيما بعد يفضل فصل كل العمليات ذات الأولوية المهمة بأقواس عن بقية أجزاء العمليات الرياضية.

ملاحظة أخرى: القسمة في البايثون ٢ معقدة قليلاً بحيث ان ناتج قسمة عدد صحيح على عدد صحيح اخر هو عدد صحيح مع اهمال الكسور العشرية ولكن هذا الشيء اختلف الان مع البايثون ٣ حيث أصبح المفسر لا يهمل أي جزء من الناتج ويعرضه كاملاً حتى لو كانت القيم الاصلية صحيحة وكما في ادناه:

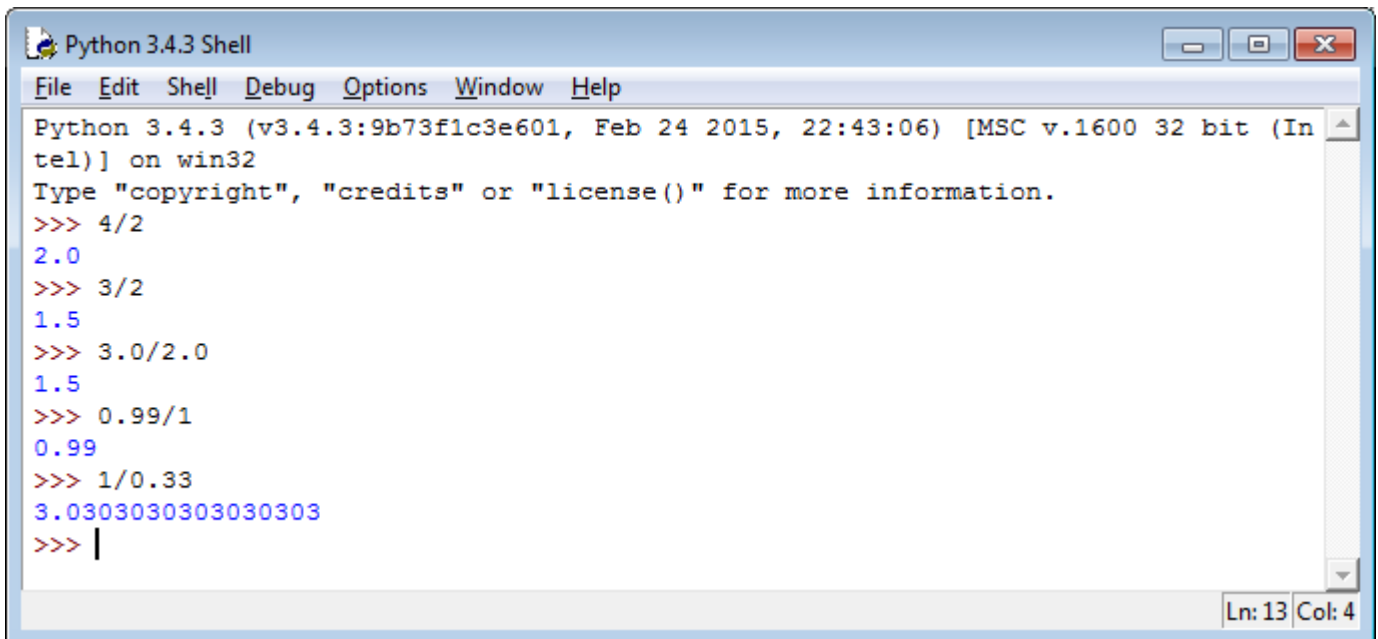
في البايثون ٢


```

>>> print 10 / 2
5
>>> print 9 / 2
4
>>> print 99 / 100
0
>>> print 10.0 / 2.0
5.0
>>> print 99.0 / 100.0
0.99

```

اما في البايثون ٣



The screenshot shows a Python 3.4.3 Shell window with the following content:

```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 4/2
2.0
>>> 3/2
1.5
>>> 3.0/2.0
1.5
>>> 0.99/1
0.99
>>> 1/0.33
3.0303030303030303
>>> |

```

في لغة البايثون ٢ كانت عملية قسمة عدد صحيح على عدد عشري او بالعكس تعطي ناتج عشري دائماً وهو امر مماثل لما يحصل الان في لغة بايثون ٣ التي عالجت المشكلة من الأساس.

أنواع المتغيرات والثوابت (Types of constants and variables):

بعض الأحيان نواجه مشاكل في التعامل مع المتغيرات والثوابت وتظهر لنا رسائل خطأ كثيرة بسبب محاولة إجراء بعض العمليات الحسابية البسيطة بين الثوابت والمتغيرات من (أنواع) مختلفة وكما في ادناه:

```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x="hello world"
>>> y=5
>>> x+y
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x+y
TypeError: Can't convert 'int' object to str implicitly
>>> x=input("enter a number")
enter a number55
>>> y+x
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    y+x
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
Ln: 17 Col: 4

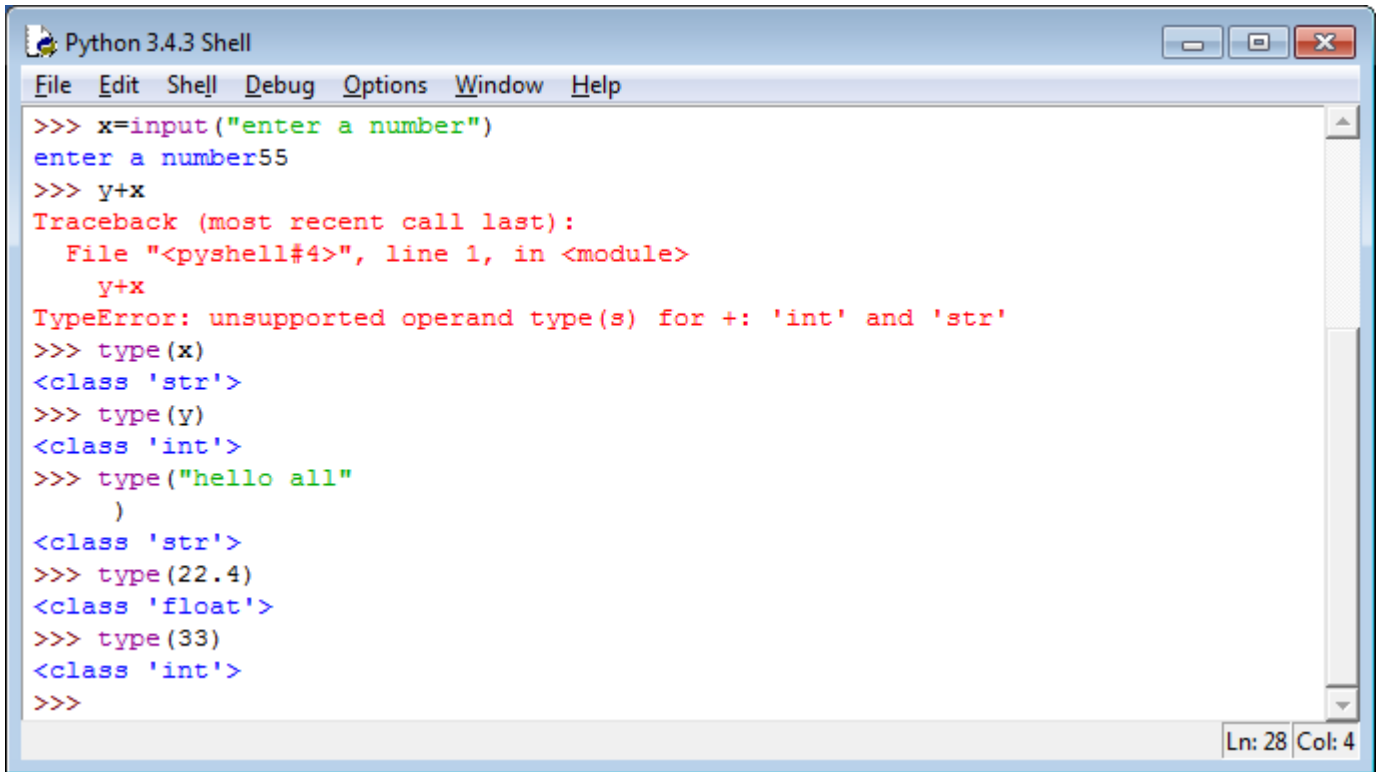
```

والسبب في الخطأ الأول هو محاولتنا جمع عدد صحيح مع سلسلة رمزية وهو خطأ بديهي طبعاً.

اما السبب للخطأ الثاني فعلى الرغم من اننا قمنا بإدخال رقم (x=55) ومحاولة جمعه مع رقم اخر وهو (y=5) الا ان رسالة خطأ قد ظهرت لأن ادخالات أداة (input) كلها تعتبر سلاسل رمزية ولمعرفة نوع كل متغير نقوم بكتابة:

Type (variable name)

وكما في ادناه:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
>>> x=input("enter a number")
enter a number55
>>> y+x
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    y+x
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> type(x)
<class 'str'>
>>> type(y)
<class 'int'>
>>> type("hello all")
<class 'str'>
>>> type(22.4)
<class 'float'>
>>> type(33)
<class 'int'>
>>>
```

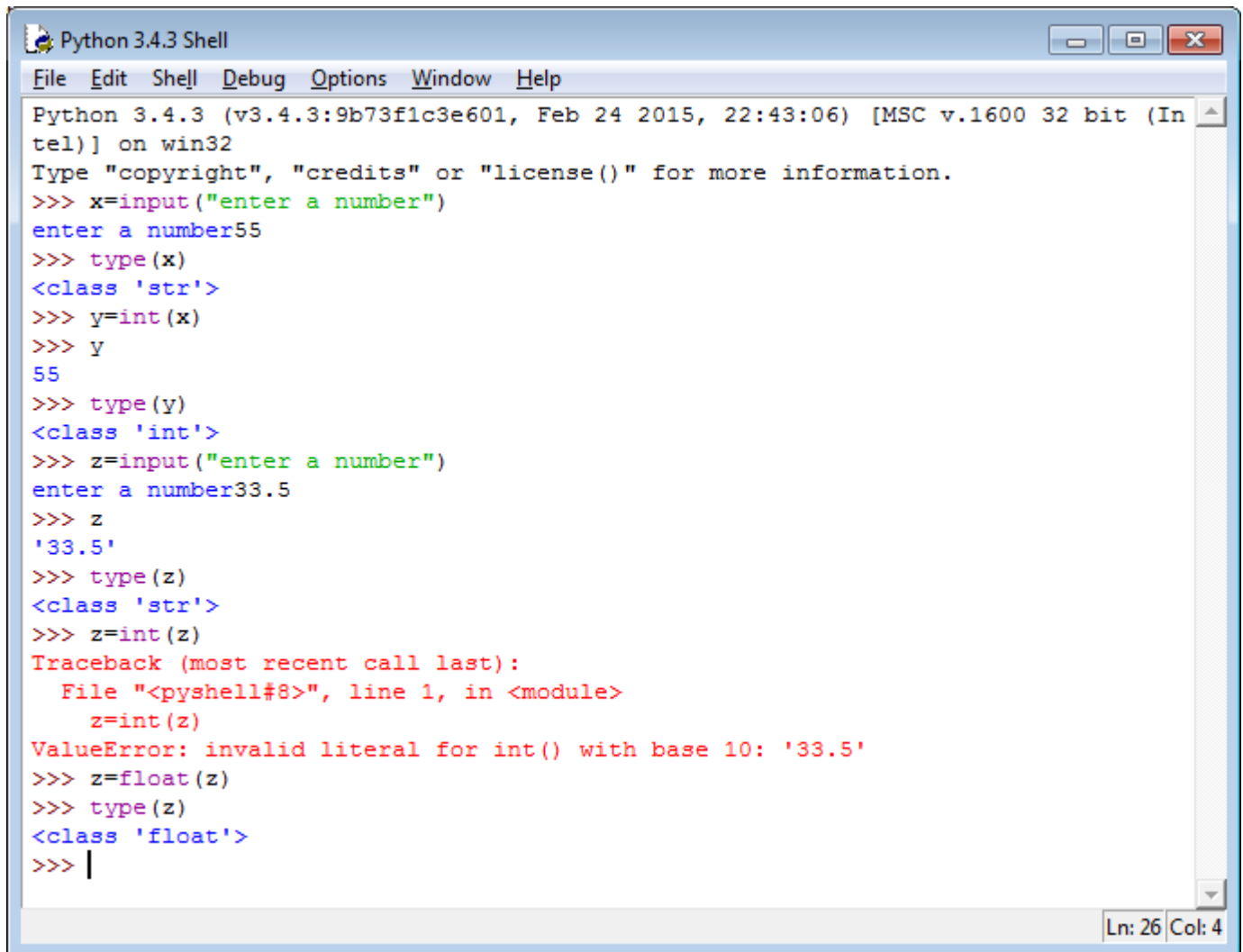
ومن هنا عرفنا ما هو نوع كل متغير او ثابت بوضعه بين قوسين وسبق ذلك بكلمة `type` والتي تساعد في حل مشاكل الأنواع واما التحويل بين الأنواع فقد تحدثنا عن بعضه سابقاً وذلك بتحويل السلسلة الرمزية الى عدد صحيح باستخدام دالة:

`int(variable name or value)`

ويمكن استخدام دالة التحويل الى عدد عشري باستخدام:

`float(variable name or value)`

وكما في ادناه:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=input("enter a number")
enter a number55
>>> type(x)
<class 'str'>
>>> y=int(x)
>>> y
55
>>> type(y)
<class 'int'>
>>> z=input("enter a number")
enter a number33.5
>>> z
'33.5'
>>> type(z)
<class 'str'>
>>> z=int(z)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    z=int(z)
ValueError: invalid literal for int() with base 10: '33.5'
>>> z=float(z)
>>> type(z)
<class 'float'>
>>> |
```

Ln: 26 Col: 4

الدرس السابع: الجمل الشرطية (conditional statements)

كما في كل لغات البرمجة، فإن لغة بايثون تتعامل مع الشروط بشكل ممتاز وباستخدام عبارة (if) وبالصيغة التالية:

If condition:

Statements

Or:

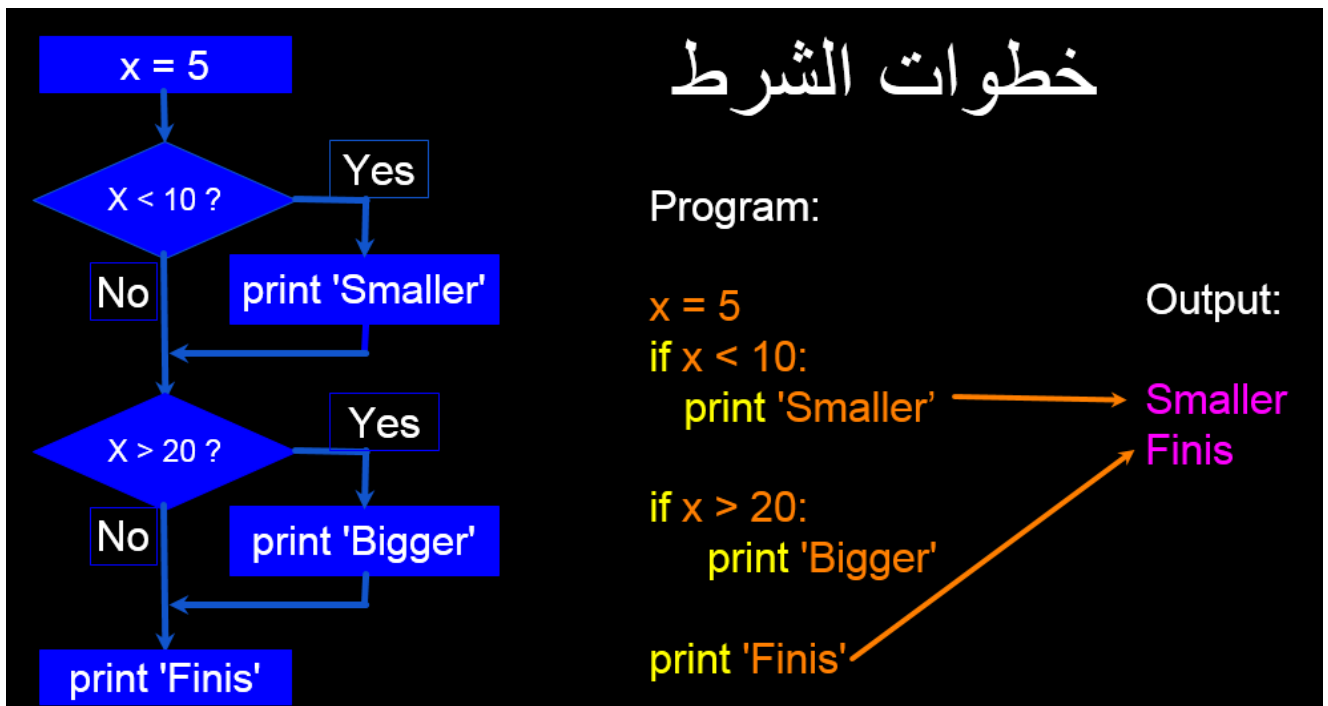
If condition:

Statements

Else:

Statements

وكما يعرف الجميع فإن العبارات (statements) لا تنفذ الا إذا كان الشرط (condition) صحيحاً واما إذا كان الشرط خطأ فيتم القفز مباشرة الى العبارات بعد بلوك ال (if) وكما في التوضيح التالي:



وكمثال على ذلك لاحظوا البرنامج التالي:

The screenshot shows two windows from a Python IDE. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
x=5
if x>3:
    print ("greater")
    print (" hello")
    print ("also will be printed")
else:
    print ("smaller")
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
greater
 hello
also will be printed
>>> |
```

ونلاحظ هنا ان الشرط بجانب ال (if) وهو (x>3) صحيح فتم تنفيذ كل العبارات داخل بلوك ال (if) واما لتمييز العبارات التابعة لل (if) فبخلاف لغات البرمجة الأخرى، لا تستخدم بايثون الاقواس وانما المسافات (indents) لتمييز العبارات التابعة لل (if) ولغيرها من عبارات الشروط والتكرار التي سنتحدث عنها لاحقاً. واما لإنهاء بلوك ال (if) او ال (else) او أي شيء غيرها فنقوم فقط بإرجاع مؤشر الطباعة الى بداية السطر او بمحاذاة الدالة السابقة لنعطي للبايثون رسالة تفيد بأن البلوك الحالي انتهى. وقبل الخوض في بقية تفاصيل استخدام عبارة (if) لا بد من التذكير بعبارات المقارنة المنطقية التي يمكن استخدامها مع عبارة (if) وهي كما يلي:

أدوات المقارنة المنطقية	معانيها
<	اقل من
<=	اقل من او يساوي
==	هل يساوي؟
>=	اكبر من او يساوي
>	اكبر من
!=	لا يساوي

وكمثال لدور المسافات (indents) في التفريق بين بلوك واخر لاحظ الصورة التالية:

```
x = 5
if x == 5 :
    print 'Equals 5'
if x > 4 :
    print 'Greater than 4'
if x >= 5 :
    print 'Greater than or Equals 5'
if x < 6 : print 'Less than 6'
if x <= 5 :
    print 'Less than or Equals 5'
if x != 6 :
    print 'Not equal 6'
```

Equals 5
Greater than 4
Greater than or Equals 5
5
Less than 6
Less than or Equals 5
Not equal 6

حيث نلاحظ ان كل لون مختلف هو بلوك يتميز عن بقية البلوكات وبعضها يجتمع بسطر واحد وبعضها بعدة أسطر.

كما يعرف المبرمجون ببقية لغات البرمجة، فإن عبارة (if) تحتوي الكثير من الخيارات والمميزات ومنها انها يمكن ان تستخدم في التنفيذ بمسار واحد او بعدة مسارات وبحسب نوعية استخدامها وكما في التفصيل ادناه:

التنفيذ بمسار واحد

```

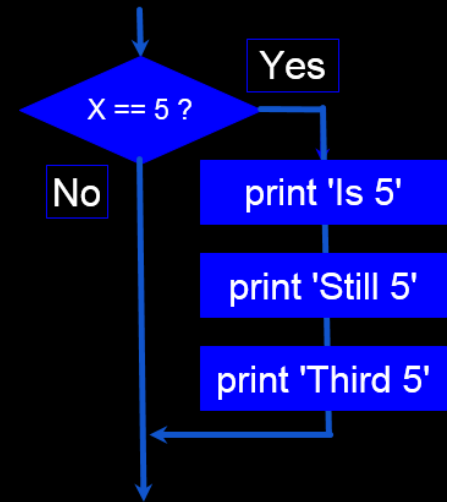
x = 5
print 'Before 5'
if x == 5 :
    print 'Is 5'
    print 'Is Still 5'
    print 'Third 5'
print 'Afterwards 5'
print 'Before 6'
if x == 6 :
    print 'Is 6'
    print 'Is Still 6'
    print 'Third 6'
print 'Afterwards 6'

```

```

Before 5
Is 5
Is Still 5
Third 5
Afterwards
5
Before 6
Afterwards
6

```



حيث يتضح من المثال أعلاه ان هناك شرط واحد للتنفيذ اما ان يكون صحيح فيتم التنفيذ واما ان يكون خطأ فيقفز المفسر الى ما بعد بلوك (if) لتنفيذه.

ملاحظة: كما ذكرنا سابقاً فإن المسافات (indents) تعمل بدل الاقواس لفصل بلوك (if) عن بقية مكونات البرنامج وهنا يجب الانتباه الى استخدام زر (space) من لوحة المفاتيح أربع مرات والابتعاد كلياً عن استخدام (tab) لأنه يعطي رسالة خطأ دوماً حيث يعتبر ال (tab) رمزاً غير مفهوم للغة بايثون فيجب الحذر.

ولمزيد من التوضيح حول المسافات ودورها في تحديد مسار التنفيذ لاحظ الأمثلة التالية:

اللون الأخضر مسافة تلقائية من المفسر بعد if
اللون البنفسجي هو الغاء المسافة من قبل المبرمج
للدلالة على ان بلوك if انتهى

```
→ x = 5
→ if x > 2 :
→     print 'Bigger than 2'
→     print 'Still bigger'
← print 'Done with 2'

→ for i in range(5) :
→     print i
→     if i > 2 :
→         print 'Bigger than 2'
←     print 'Done with i', i
← print 'All Done'
```

ولتوضيح البلوكات المتداخلة نضعها بألوان مختلفة هنا:

```

x = 5
if x > 2 :
    print 'Bigger than 2'
    print 'Still bigger'
print 'Done with 2'

for i in range(5) :
    print i
    if i > 2 :
        print 'Bigger than 2'
    print 'Done with i', i
print 'All Done'

```

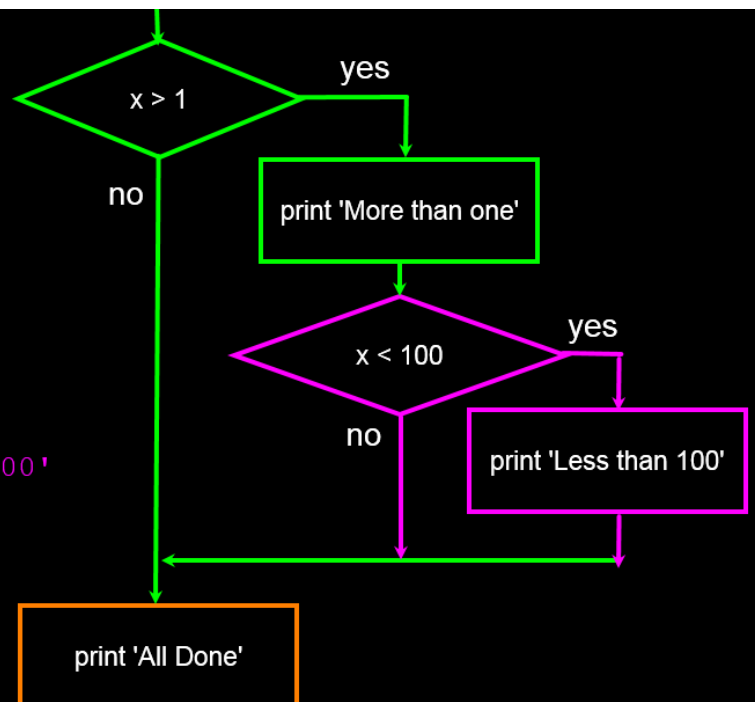
لاحظ عدم وجود اقواس وانما مسافات فقط.

القرارات المتشعبة

```

x = 42
if x > 1 :
    print 'More than one'
    if x < 100 :
        print 'Less than 100'
print 'All done'

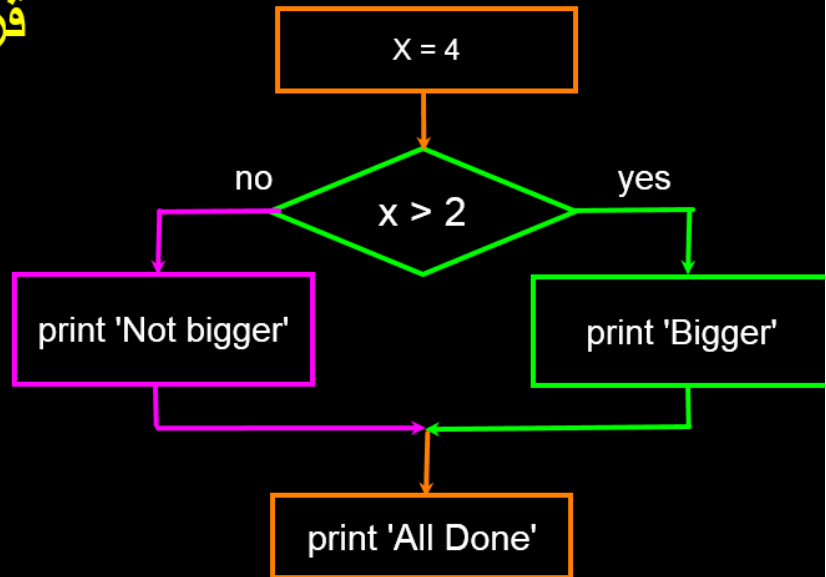
```



كما في الصورة السابقة نرى ان عبارة (if) تستخدم بشكل متشعب (if بداخل if) وتسمى باصطلاح البرمجة (nested if) وتعني انه ان كان الشرط الأول صحيح فقم بالدخول الى البلوك الخاص به لتجد شرطاً اخر، فأن كان هذا الشرط صحيح ايضاً يقوم المفسر بالدخول الى داخل البلوك الخاص به والا فلا. وهكذا؟

قرارات التنفيذ بمسارين

- في اغلب الأحيان نريد التنفيذ بمسار ان كان الشرط صحيحاً او الانتقال الى مسار اخر ان كان الشرط خاطئاً وهنا تفيد عبارة (elif = else if)



يمكن في هذه الحالات استخدام (else) او (elif = else if) وبحسب الحاجة وكما في ادناه:

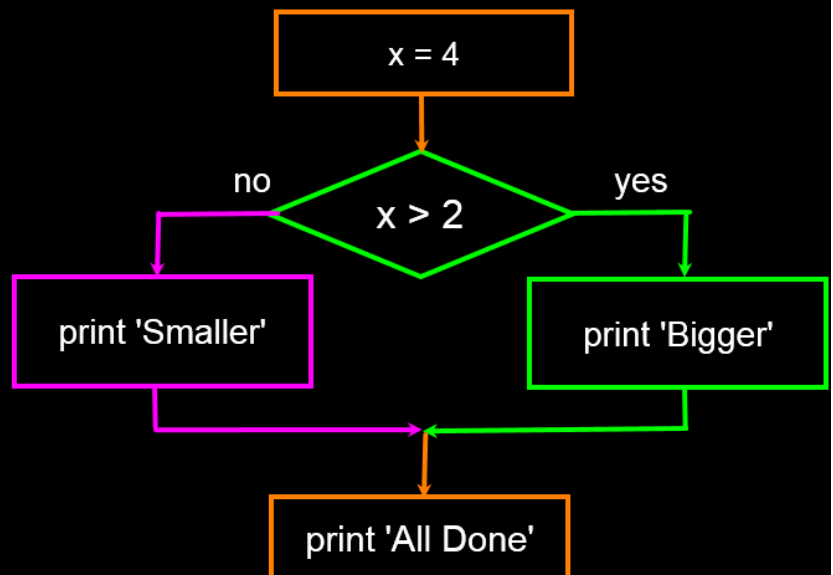
التنفيذ بمسارين باستخدام else

```

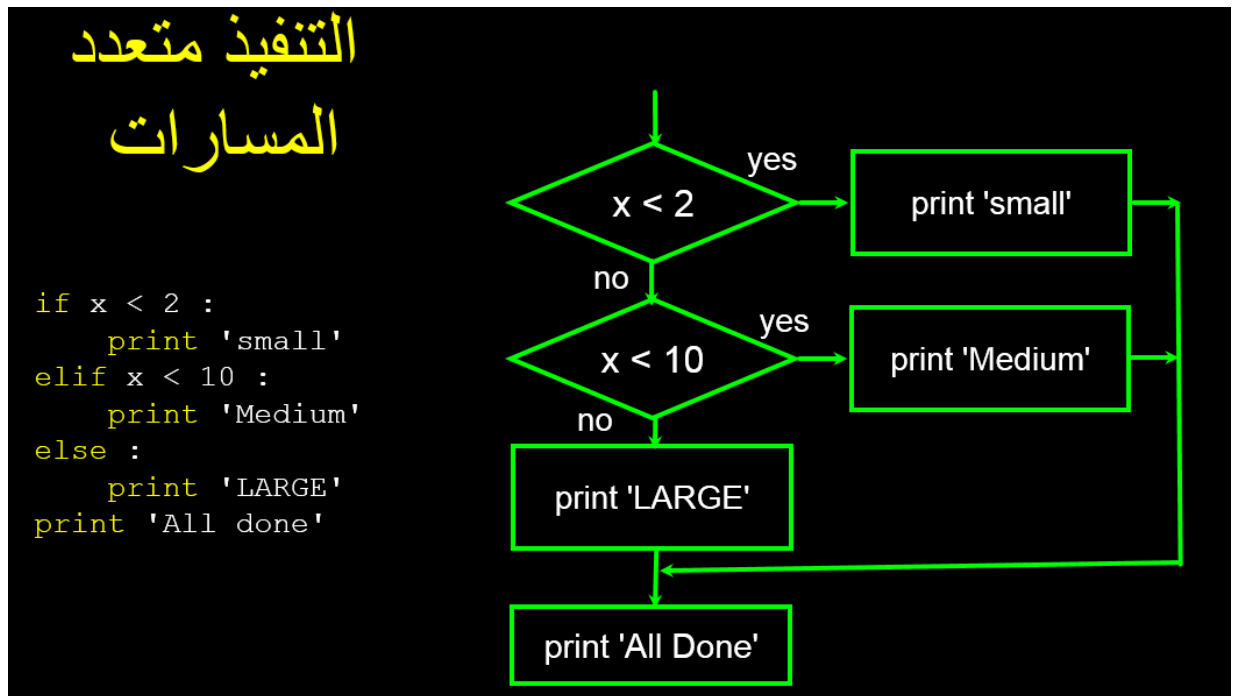
x = 4

if x > 2 :
    print 'Bigger'
else :
    print 'Smaller'

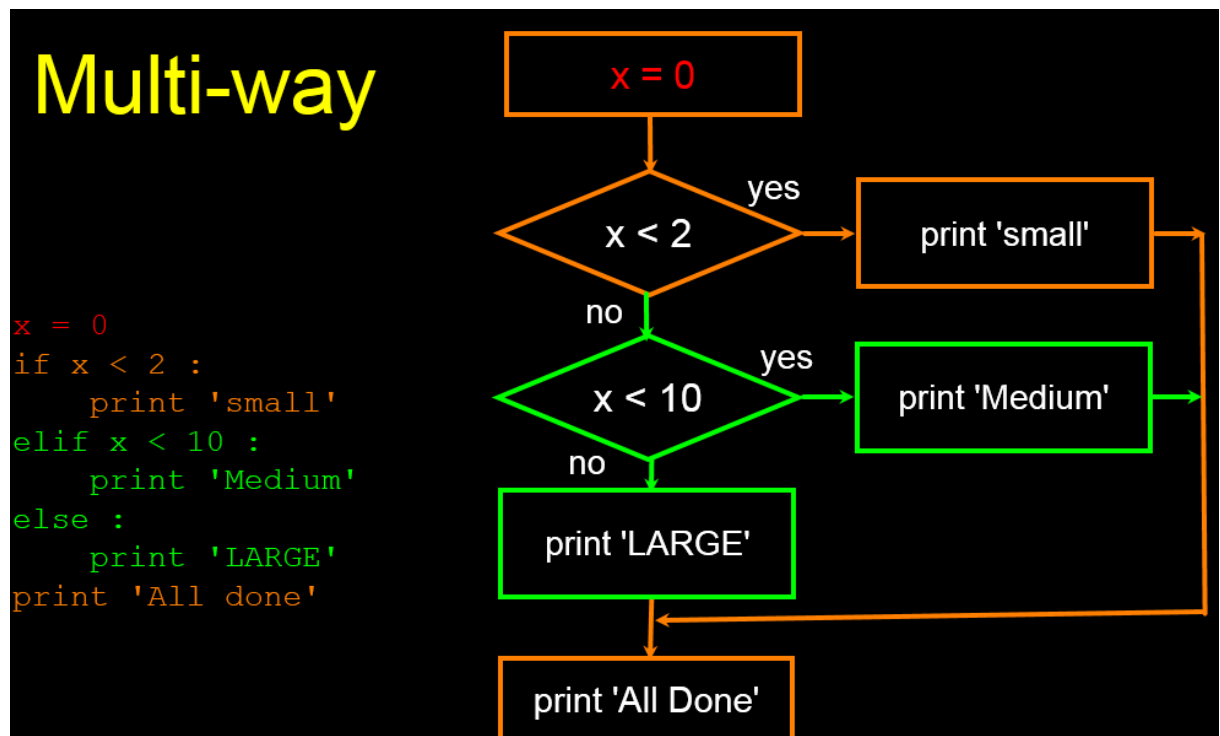
print 'All done'
  
```



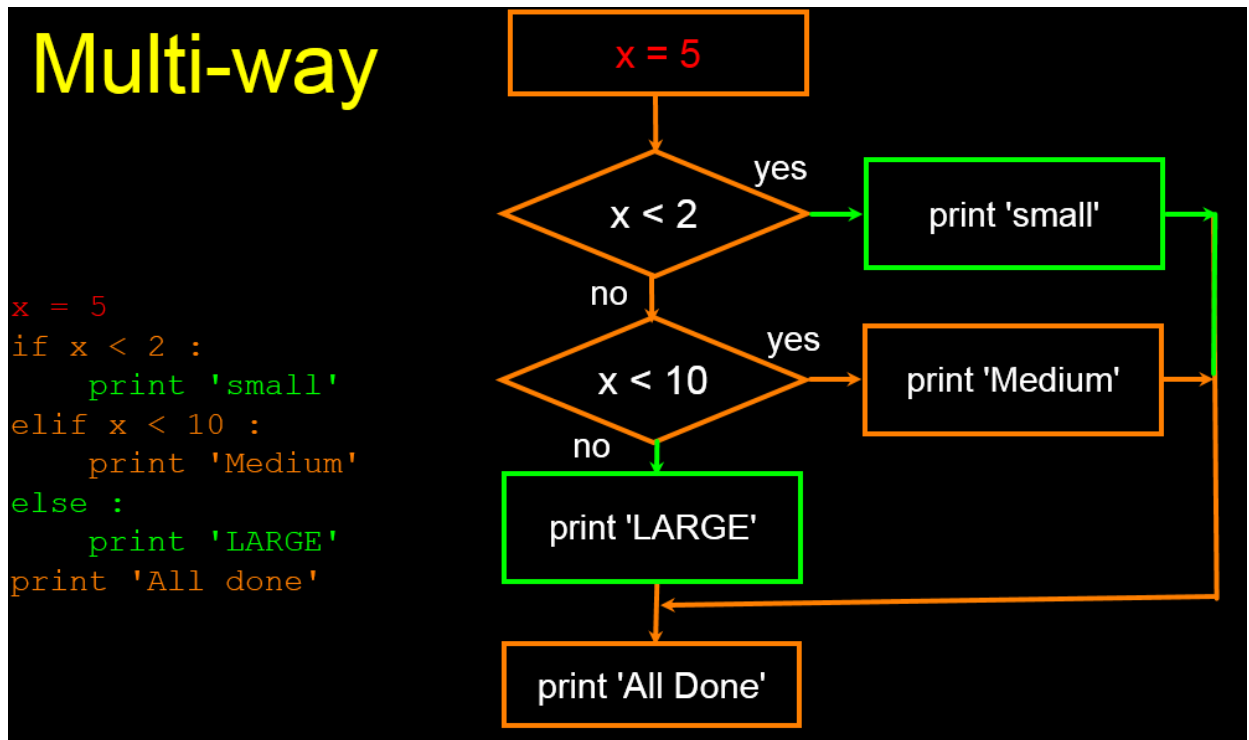
ويمكن ان يكون التنفيذ متعدد المسارات وكما في ادناه:



ويمكن تنفيذ الكود أعلاه بعدة طرق اعتماداً على قيمة (x) وكما في ادناه:

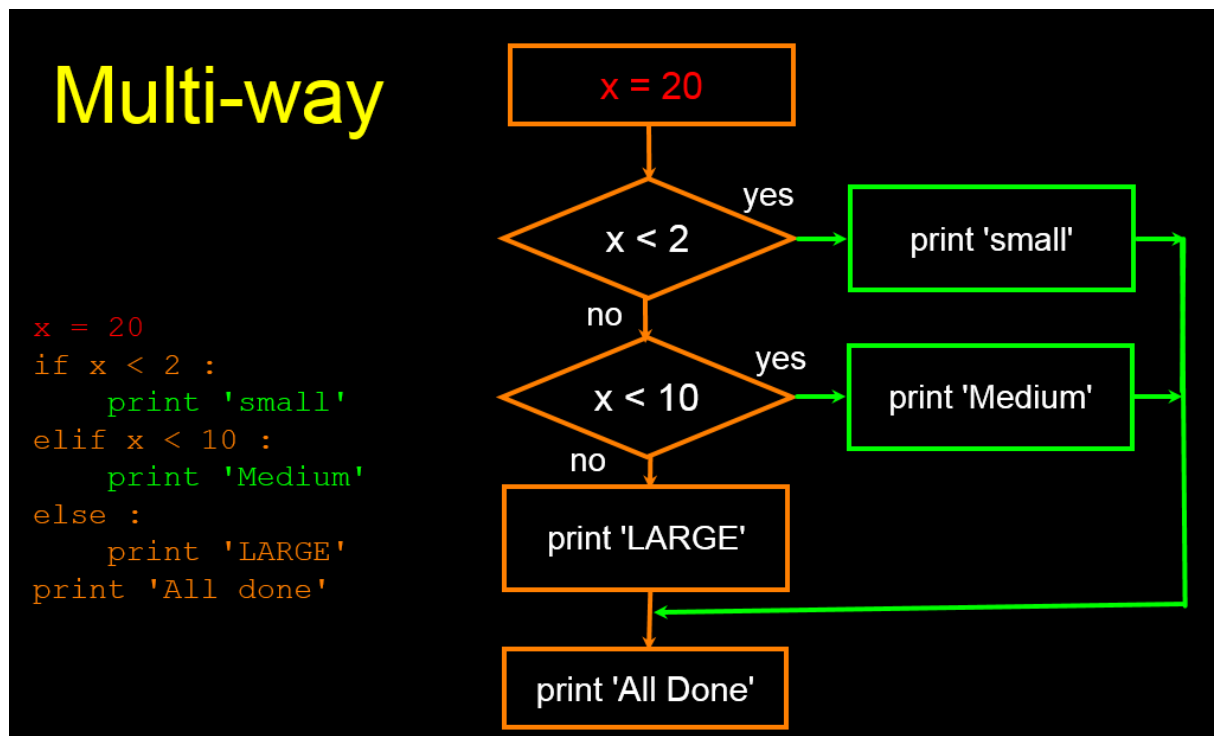


حيث ينفذ الجزء البرتقالي فقط لأن قيمة (x) تساوي صفر.



وهنا تنفذ الأجزاء البرتقالية لأن قيمة (x) تساوي ٥

واخيراً لقيمة (x) تساوي ٢٠ نرى التنفيذ التالي:



لاحظ ايضاً:

في التنفيذ متعدد المسارات يمكن استخدام `else` ويمكن عدم استخدامها

```
# No Else
x = 5
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'

print 'All done'
```

```
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'
elif x < 20 :
    print 'Big'
elif x < 40 :
    print 'Large'
elif x < 100:
    print 'Huge'
else :
    print 'Ginormous'
```

واخيراً يمكن كتابة كودات برامج تحتوي على أسطر لا تنفذ ابداً وكما في الصورة التالية:

بعض الاسطر هنا لن تنفذ ابداً
فهل يمكنك معرفتها؟

```
if x < 2 :
    print 'Below 2'
elif x >= 2 :
    print 'Two or more'
else :
    print 'Something else'
```

```
if x < 2 :
    print 'Below 2'
elif x < 20 :
    print 'Below 20'
elif x < 10 :
    print 'Below 10'
else :
    print 'Something else'
```

الدرس الثامن: أدوات الشرط والاستثناء (Try-except statement)

بعد ان درسنا في الدرس السابق هيكل واستخدام الأنواع المختلفة لعبارة (if) الشرطية، نأتي اليوم لمناقشة مشكلة تحصل بكثرة اثناء استخدام أدوات الادخال للمتغيرات والثوابت وهي احتمالية ادخال المستخدم لقيمة لا تطابق شروط البرنامج او ما يتوقعه المبرمج فيسبب ذلك خطأ في التنفيذ عادة ولكن وجود أداة (try-except) يسمح بتلافي هذه المشاكل وكما في التوضيح التالي:

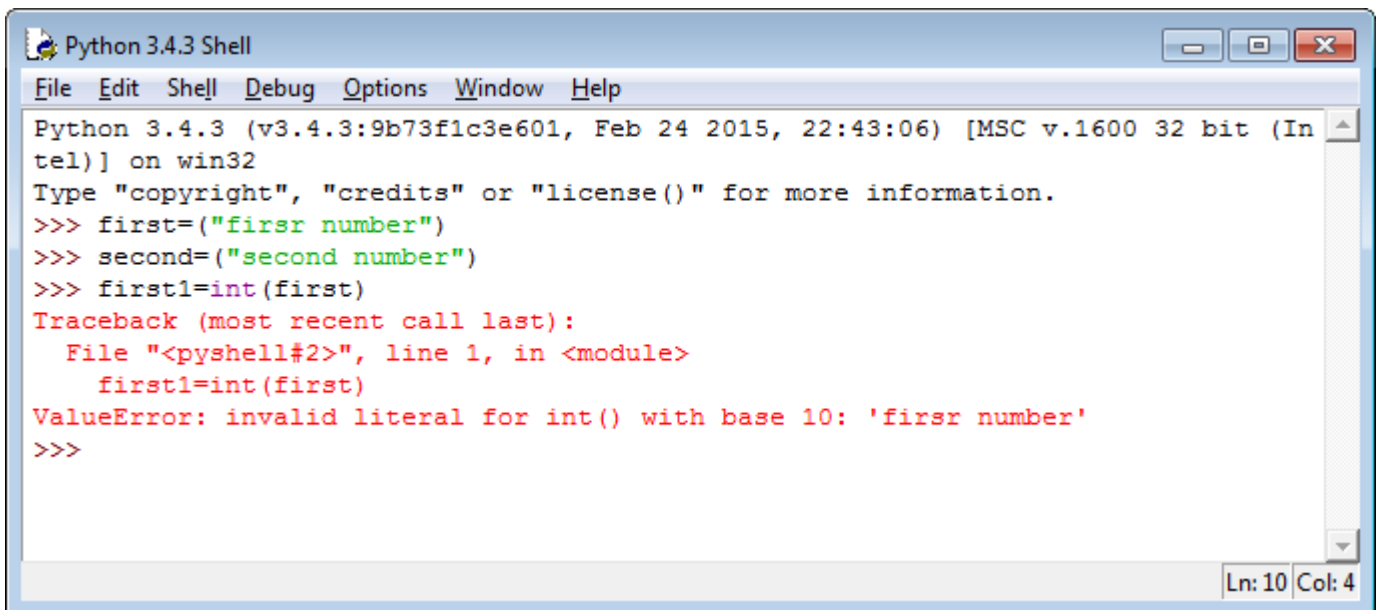
Try

هنا نضع الكود الخطر او المحتمل ان يسبب فشل تنفيذ البرنامج

Except

هنا نضع حل الاشكال المحتمل وكما سنرى في المثال التالي:

نفترض اننا نريد المستخدم ان يدخل قيمة رقمية (integer or float) لأدخالها في معادلات رياضية ولكنه يصدف ان يقوم المستخدم بأدخال قيمة متغير رمزي (string) مما يسبب توقف التنفيذ وظهور رسالة خطأ وكما في ادناه:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> first=("firsr number")
>>> second=("second number")
>>> first1=int(first)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    first1=int(first)
ValueError: invalid literal for int() with base 10: 'firsr number'
>>>
```

وهنا نرى انه من الطبيعي ان تظهر هذه الرسالة لأننا حاولنا تحويل قيمة رمزية الى عدد صحيح وهو شيء غير مقبول برمجياً. ولتجنب هذه المشكلة نقوم بالتالي:

The image shows a screenshot of a Python IDE window titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)'. The code in the editor is as follows:

```
x="first"
y="second"
try:
    x1=int(x)
    y1=int(y)
except:
    x1=-1
    y1=-1
print(x1)
print(y1)
```

Below the editor is a 'Python 3.4.3 Shell' window showing the execution output:

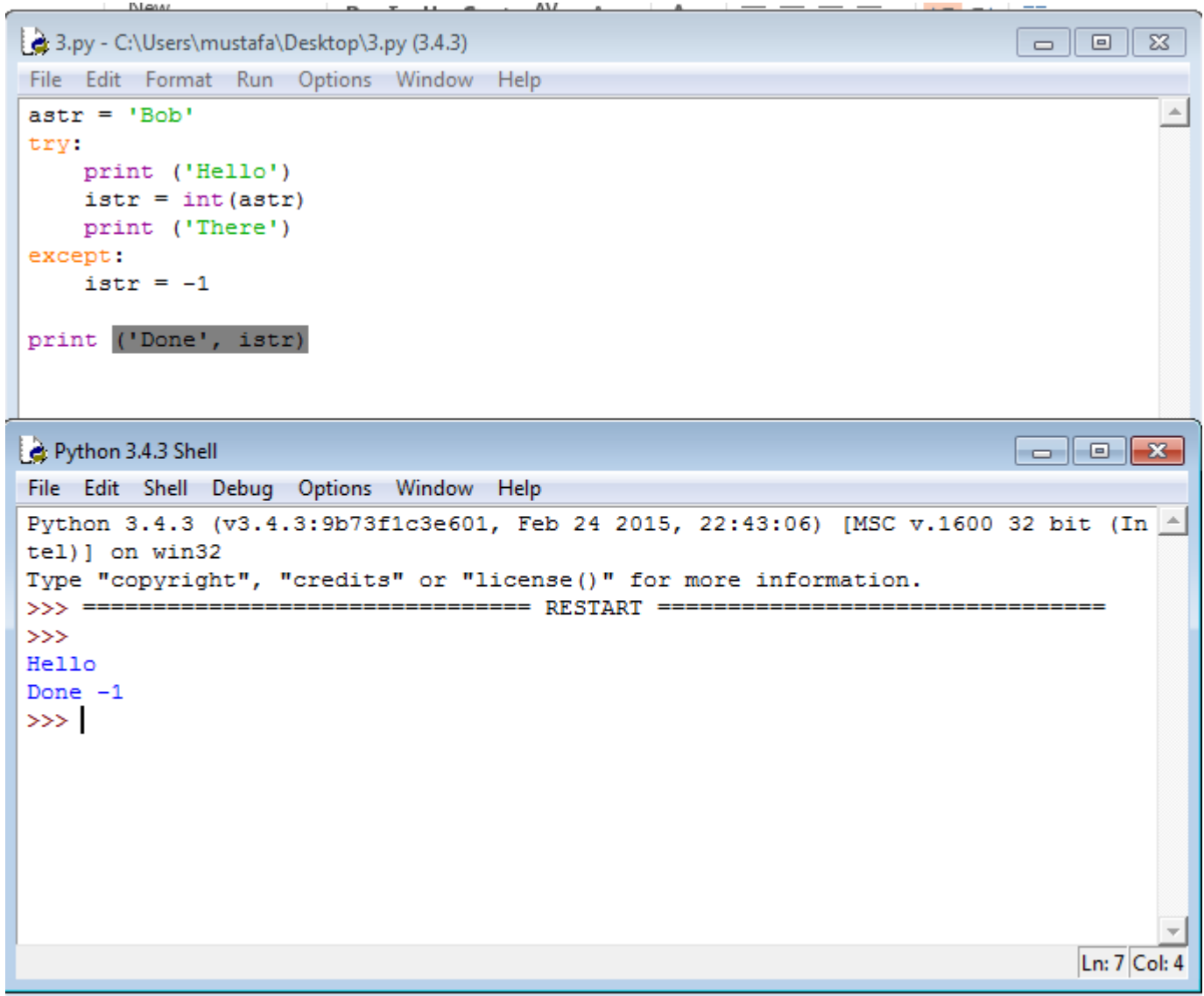
```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
-1
-1
>>>
```

The shell window also shows the status 'Ln: 7 Col: 4' at the bottom right.

وهنا قمنا بأخبار المفسر بالقيام بالتالي:

ان يقوم بأخذ قيمة المتغير (x) على انها "first" والمتغير (y) على انها "second" ثم قلنا للمفسر حاول (try) تحويل المتغيرين (x,y) الى قيم صحيحة، فأن كان ذلك ممكناً فسيقوم بتنفيذ ما بداخل عبارة (try) والا فسيقفز مباشرة الى ما بداخل عبارة الاستثناء (except) لينفذ ما بداخلها وهو ما حصل أعلاه.

مثال اخر على التعامل مع عبارة المحاولة والاستثناء:



The image shows a screenshot of a Python IDE window titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)'. The code in the editor is as follows:

```
astr = 'Bob'
try:
    print ('Hello')
    istr = int(astr)
    print ('There')
except:
    istr = -1

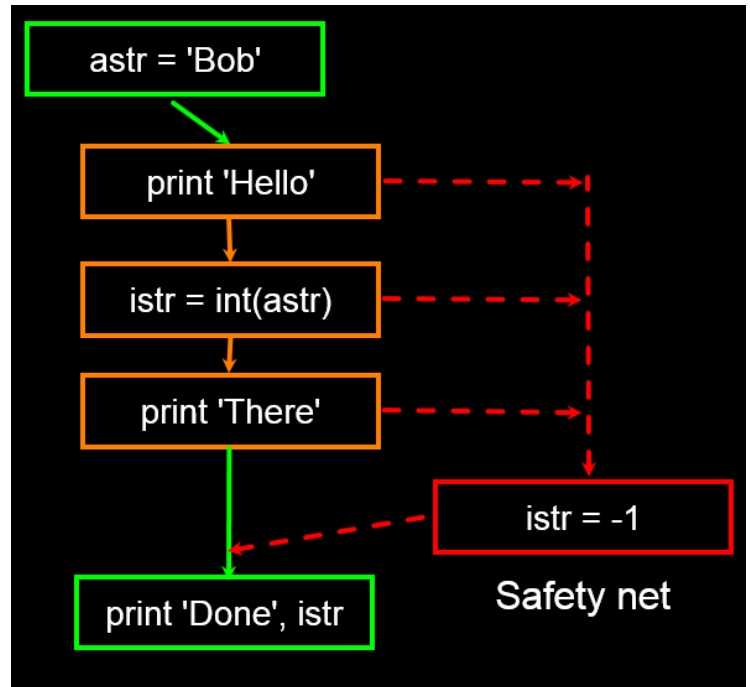
print ('Done', istr)
```

Below the editor is a 'Python 3.4.3 Shell' window showing the execution output:

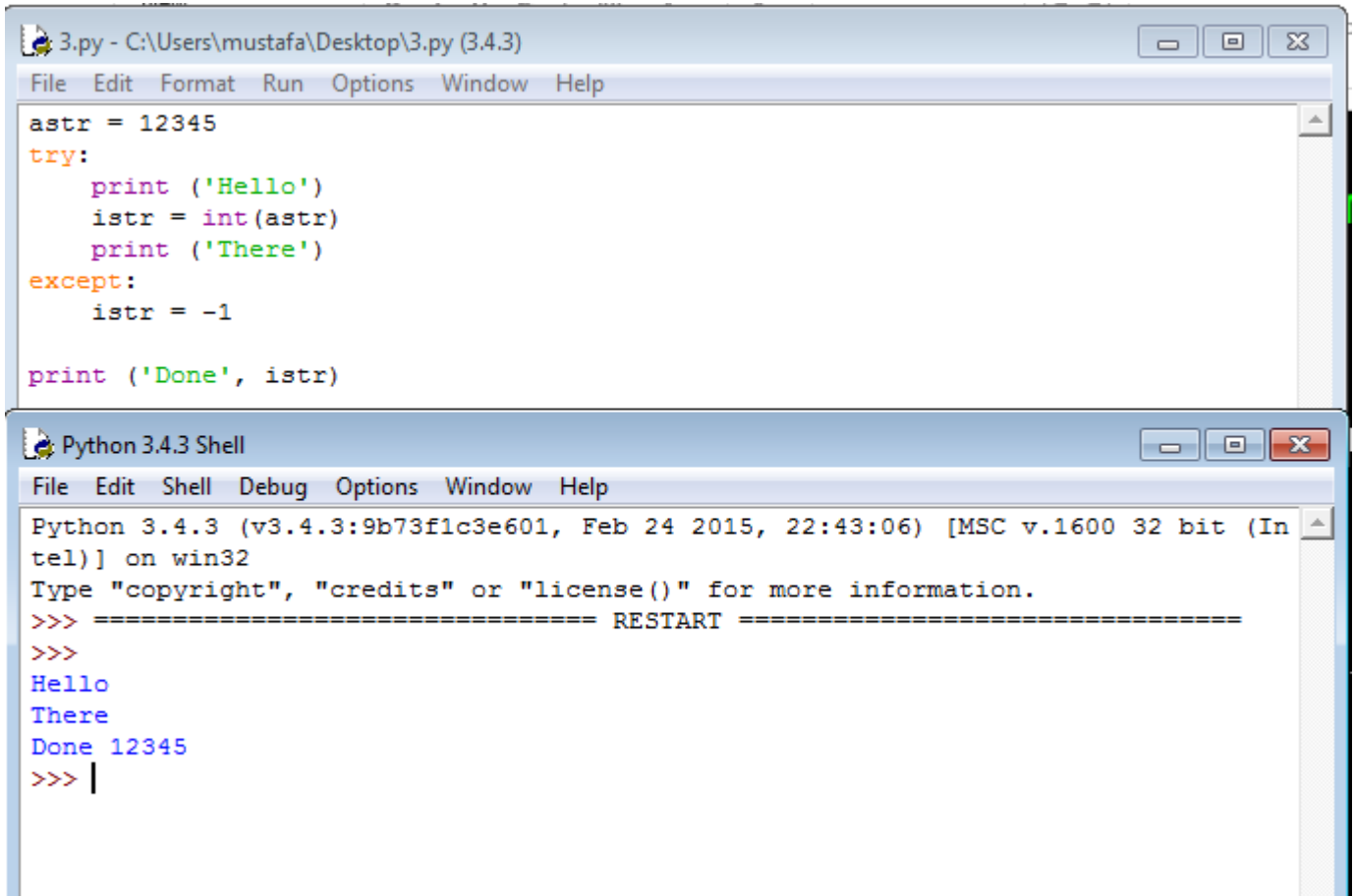
```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Hello
Done -1
>>> |
```

The status bar at the bottom right of the shell window indicates 'Ln: 7 Col: 4'.

والتوضيح لتسلسل التنفيذ كما في ادناه:



وهذه نتيجة التنفيذ في حالة تبديل قيمة المتغير (astr) وجعلها قيمة عددية حيث يتم تنفيذ ما بداخل عبارة (try) وكما في ادناه:

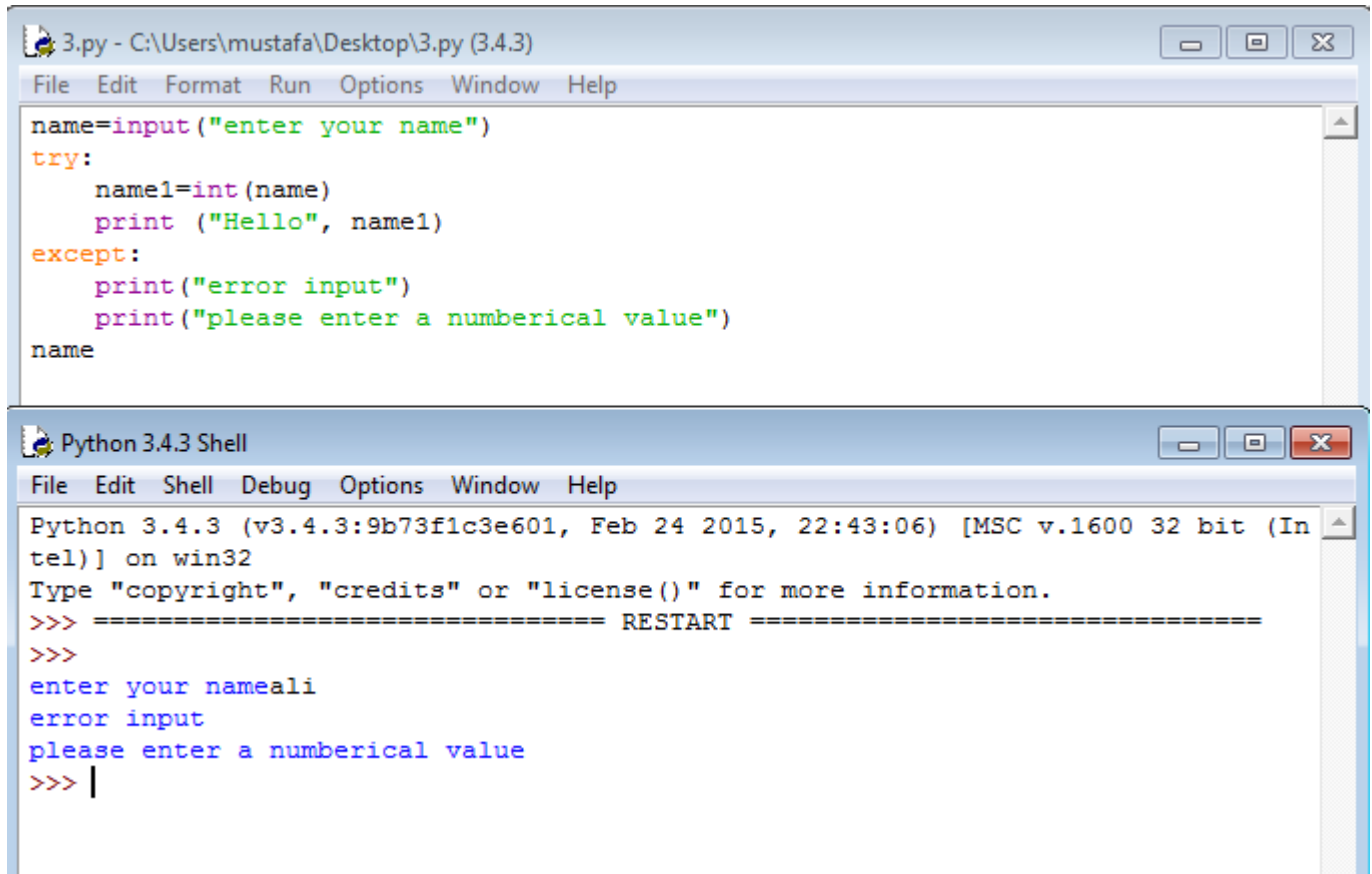


```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
astr = 12345
try:
    print ('Hello')
    istr = int(astr)
    print ('There')
except:
    istr = -1

print ('Done', istr)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Hello
There
Done 12345
>>> |
```

واخيراً لابد من الإشارة الى ان عبارة المحاولة والاستثناء يفضل ان تستخدم في كل البرامج التي تتضمن نوع من الادخال للمستخدم وعادة تستخدم لوضع رسالة تظهر للمستخدم في حالة ادخال خاطيء وغير مناسب لمتطلبات البرنامج وكما في ادناه:



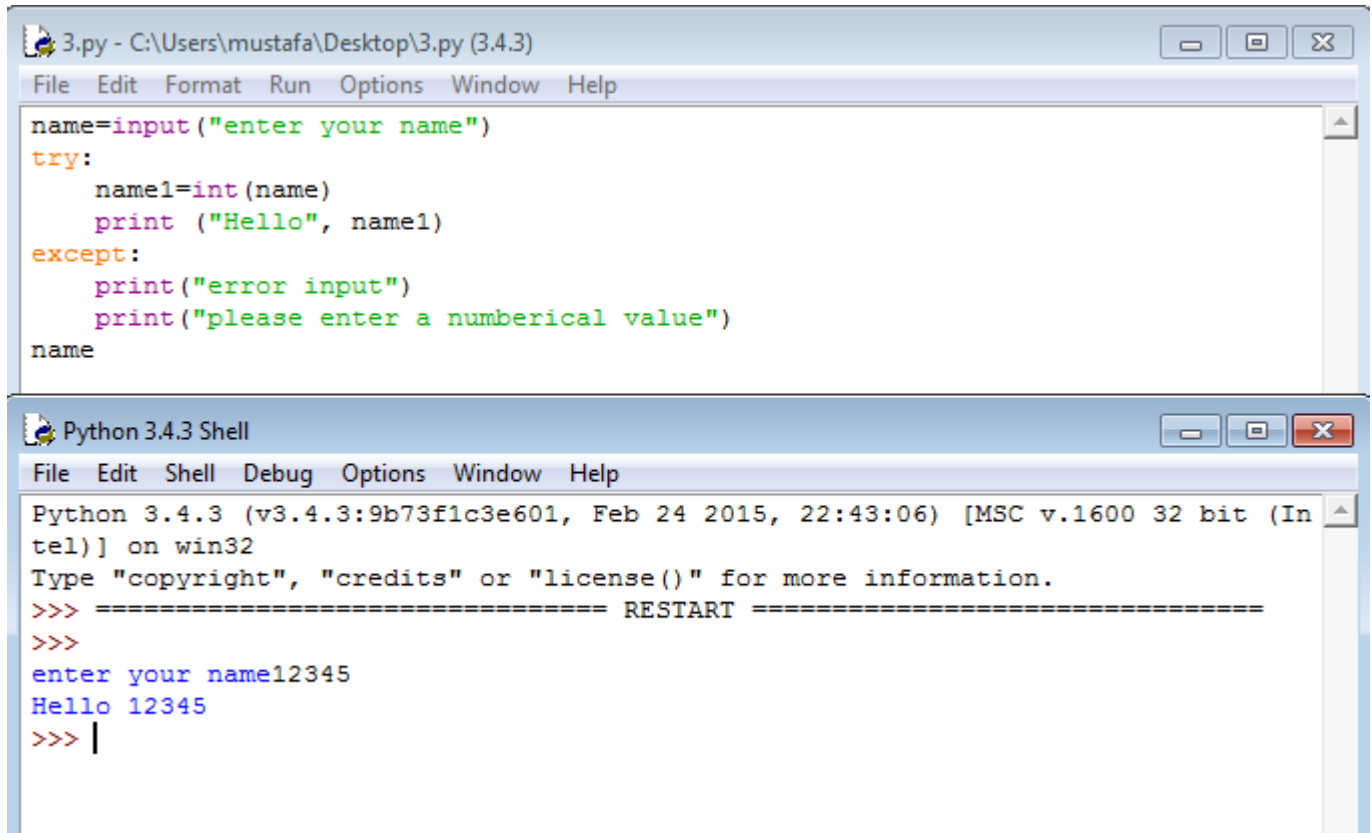
The image shows two windows from a Python IDE. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
name=input("enter your name")
try:
    name1=int(name)
    print("Hello", name1)
except:
    print("error input")
    print("please enter a numerical value")
name
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter your nameali
error input
please enter a numerical value
>>> |
```

واما في حالة ادخال قيمة صحيحة فيكون التنفيذ كما في ادناه:



The image shows two windows from a Python IDE. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
name=input("enter your name")
try:
    name1=int(name)
    print("Hello", name1)
except:
    print("error input")
    print("please enter a numerical value")
name
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution of the script. The output is as follows:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter your name12345
Hello 12345
>>> |
```

وفي نهاية درس اليوم ارفق لكم احبتي بعض الامثلة المحلولة لبرامج ذات متطلبات معينة أتمنى ان تكون مفيدة لكم ومن الله التوفيق:

3.1 Write a **PROGRAM** to prompt the user for hours and rate per hour using `raw_input` to compute gross pay. Pay the hourly rate for the hours up to 40 and 1.5 times the hourly rate for all hours worked above 40 hours. Use 45 hours and a rate of 10.50 per hour to test the program (the pay should be 498.75). You should use `raw_input` to read a string and `float()` to convert the string to a number. Do not worry about **ERROR** checking the user input - assume the user types numbers properly.

Check Code

Reset Code



Exit

Grade updated on server.

```

1 hrs = raw_input("Enter Hours:")
2 h = float(hrs)
3 rate=raw_input("enter the rate")
4 r=float(rate)
5 if h > 40:
6     pay=(40*r) + (h-40)*(r*1.5)
7     print pay
8 else:
9     pay=h*r
10    print pay
11

```

3.3 Write a **PROGRAM** to prompt for a score between 0.0 and 1.0. If the score is out of range, print an error. If the score is between 0.0 and 1.0, print a grade using the following table:

Score Grade

>= 0.9 A

>= 0.8 B

>= 0.7 C

>= 0.6 D

< 0.6 F

If the user **ENTERS** a value out of range, print a suitable error message and exit. For the test, enter a score of 0.85.

Check Code



Exit

Grade updated on server.

```

1 score=raw_input("enter a number between 0.0 and 1.0")
2 try:
3     s=float(score)
4 except:
5     print "not a number"
6 if s>1.0:
7     print "error"
8     quit()
9 if s>=0.9:
10    print "A"
11 elif s>=0.8:
12    print "B"
13 elif s>=0.7:
14    print "C"
15 elif s>=0.6:
16    print "D"
17 else:
18    print "F"
19

```

الدرس التاسع: الدوال (functions)

الدوال او البرامج الفرعية كما تسمى في بعض لغات البرمجة هي قطع برمجية يتم انشائها من قبل المبرمج (user defined functions) او تكون موجودة في لغة البرمجة مبنية بداخلها (built in functions) ويتم استدعائها بعد انشائها بأي عدد من المرات وتعتبر بديلاً ممتازاً لكتابة هذه الاكواد والقطع البرمجية مئات المرات داخل البرنامج الواحد. فمثلاً لو احتجنا الى استدعاء دالة المفكوك (factorial) في برنامج لحل المعادلات المعقدة عدداً كبيراً من المرات فبدل ان نقوم بكتابة كود المفكوك كل مرة نحتاجه فيها، نقوم بوضع هذا الكود داخل دالة نستدعيها كل مرة ولمتغيرات مختلفة حيث نقوم بتمرير الأرقام او المتغيرات التي نريد حساب مفكوكها كل مرة بدل كتابة كود جديد.

ولتوضيح فائدة الدوال نقوم بتنفيذ المثال التالي:

The screenshot shows a Python IDE window titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)'. The code in the editor is as follows:

```
def hello():
    print("hello world")
    print("python is cool")
    print("are you ready for the next lessons?")
x=input("enter your name")
print(x)
hello()
print(x*5)
hello()
```

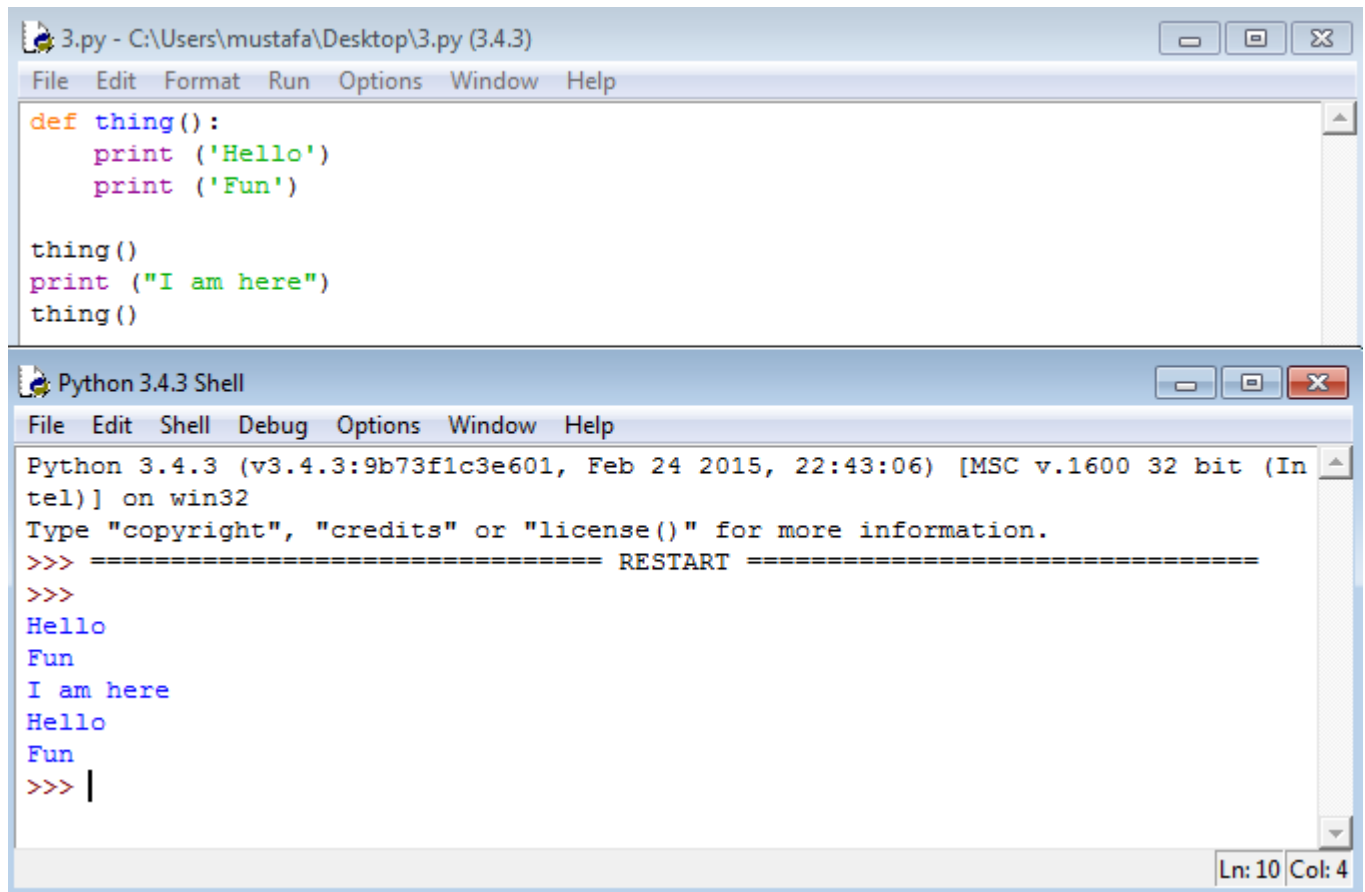
Below the code editor is the 'Python 3.4.3 Shell' window. It shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter your namemustafa
mustafa
hello world
python is cool
are you ready for the next lessons?
mustafamustafamustafamustafamustafa
hello world
python is cool
are you ready for the next lessons?
>>> |
```

The status bar at the bottom right of the shell window shows 'Ln: 14 Col: 4'.

هنا نلاحظ الأمور التالية:

- ١- لتعريف دالة معينة نسبق اسمها بالعبارة المفتاحية (def) وهي مختصر كلمة (Define) بمعنى تعريف.
 - ٢- نكتب اسم الدالة بعد ذلك متبوعة بأقواس فارغة (او تحتوي شيئاً ما سنشرحه لاحقاً) ثم (:)
 - ٣- نلاحظ ان المفسر يقوم مباشرة بتزحيف (shift) العبارات داخل جسم الدالة بمقدار ٤ فراغات (4 spaces) ليبدل على الان نكتب بداخل جسم الدالة.
 - ٤- يمكن ان تحتوي الدالة على أي عدد من العبارات البرمجية الحسابية او المنطقية او التكرارية او الشرطية.
 - ٥- لتعريف المفسر ان جسم الدالة قط انتهى نقوم فقط بالنزول الى سطر جديد والرجوع الى بدايته أي الغاء المسافات التلقائية ونحن بذلك نقوم بأخبار المفسر ان جسم الدالة قد انتهى.
 - ٦- كل ما سبق يسمى تعريف الدالة (function definition) وهو يقول للمفسر فقط خذ بنظر الاعتبار وجود هذه الدالة ولا يتم تنفيذ أي جزء منه حتى يتم استدعاء الدالة (function call or revoke) والذي يتم بكتابة اسم الدالة متبوعة بالأقواس الفارغة او المملوءة بشيء ما (سنعرفه بعد قليل).
 - ٧- يتم تنفيذ ما بداخل الدالة عند استدعائها وبأي عدد من المرات داخل البرنامج وهذه هي الميزة الرئيسية والفائدة الكبرى للدوال: تجنب تكرار كتابة نفس الكود أكثر من مرة.
 - ٨- يمكن كتابة أي عدد من الدوال في أي مكان من البرنامج واستدعائها عند الحاجة لأي عدد من المرات.
- والان لمزيد من الأمثلة لتوضيح الفكرة:



The image shows two windows from a Python IDE. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
def thing():  
    print ('Hello')  
    print ('Fun')  
  
thing()  
print ("I am here")  
thing()
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>> ===== RESTART =====  
>>>  
Hello  
Fun  
I am here  
Hello  
Fun  
>>> |
```

The status bar at the bottom right of the shell window indicates 'Ln: 10 Col: 4'.

وهذا المثال ايضاً:

The screenshot shows a Python IDE window titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)'. The code in the editor is as follows:

```
def repeat (name) :
    print (name)
n=input("enter your name: ")
repeat (n)
print ("something else")
repeat (n)
repeat (n)
```

Below the editor is the 'Python 3.4.3 Shell' window. It shows the execution of the script:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter your name: Mustafa
Mustafa
something else
Mustafa
Mustafa
>>> |
```

The status bar at the bottom right of the shell window indicates 'Ln: 10 Col: 4'.

هنا نرى ان الاقواس لم تعد فارغة والسبب في ذلك اننا في هذه الدالة قمنا باستدعاء الدالة لتنفيذ ما بداخلها لقيمة معينة يتم استيرادها من البرنامج الرئيسي وهي قيمة المتغير (n) والتي تأخذ محتواها من ادخال من المستخدم وهنا يكون تسلسل التنفيذ كالآتي:

- ١- يتم تعريف دالة للمفسر اسمها (repeat) وتحتوي متغير استدعاء (argument) اسمه (name).
- ٢- تحتوي الدالة على عبارة واحدة تتمثل في طباعة قيمة المتغير (name).
- ٣- بعد ذلك نقوم بتعريف متغير اسمه (n) وهو عبارة عن قيمة يتم إدخالها من قبل المستخدم بدالة (input).
- ٤- والان نقوم باستدعاء دالة (repeat) للمتغير (n) أي اننا نقول للمفسر: خذ قيمة المتغير (n) وضعها بدل متغير الدالة المسمى (name) ثم طبق ما بداخل الدالة لهذا المتغير (n).
- ٥- عندها يقوم المفسر بأخذ قيمة المتغير (n) وهي شيء يقوم المستخدم بإدخاله ويضعها بدل المتغير (name) في الدالة ويطبق الدالة (التي تحتوي عبارة واحدة هي طباعة قيمة المتغير name).
- ٦- بعدها يكمل البرنامج عمله بشكل طبيعي بتكرار استدعاء الدالة للمتغير (n) نفسه.

ولكن ماذا يحصل لو أردنا تكرار تنفيذ الدالة لمتغيرات مختلفة؟

دعونا نرى الآن:

The screenshot shows a Python IDE window titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)'. The code in the editor is as follows:

```
def repeat(name):
    print (name)
n=input("enter your name: ")
x=input("enter your father's name")
y=input("enter your grandfather's name")

repeat(n)
print ("something else")
repeat(x)
print ("something else")
repeat(y)
```

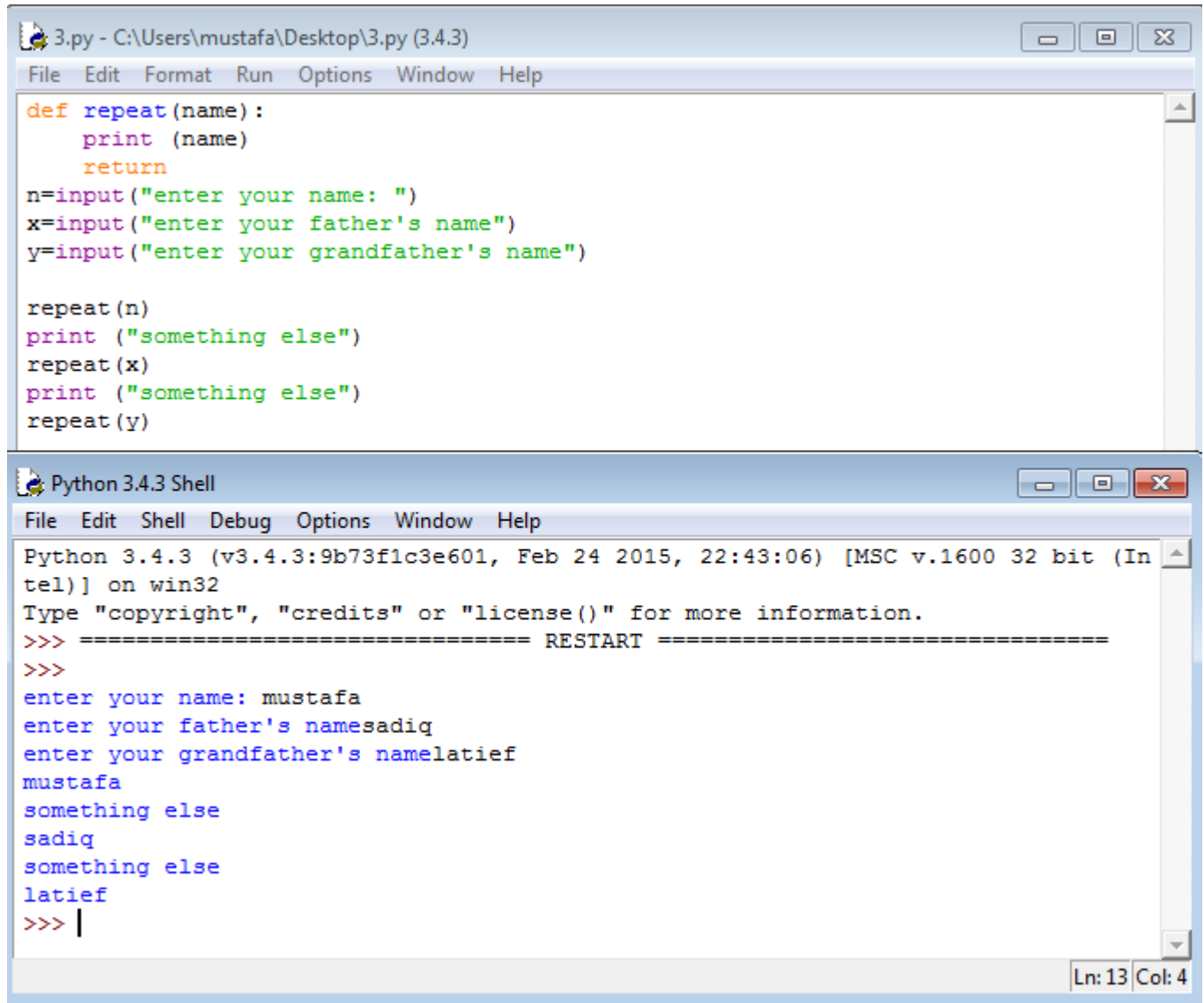
Below the editor is a 'Python 3.4.3 Shell' window showing the execution of the code. The output is:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter your name: mustafa
enter your father's namesadiq
enter your grandfather's namelatief
mustafa
something else
sadiq
something else
latief
>>> |
```

The shell window also shows the status 'Ln: 13 Col: 4' at the bottom right.

هنا نرى ميزة أخرى وفائدة أخرى من فوائد الدوال وهي أننا قمنا باستدعاء الدالة لأكثر من متغير وبعدها أسماء وقيم وهي (n,x,y) وفي كل مرة يتم استدعاء الدالة فيها يتم طباعة او تنفيذ الدالة بشكل مختلف.

والآن نأتي الى توضيح أحد مميزات الدوال الأخرى وهي استخدام دالة (Return) لإرجاع قيمة من الدالة الى البرنامج الرئيسي وكما في المثال ادناه:



The image shows a screenshot of a Python IDE window titled "3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)". The code in the editor is as follows:

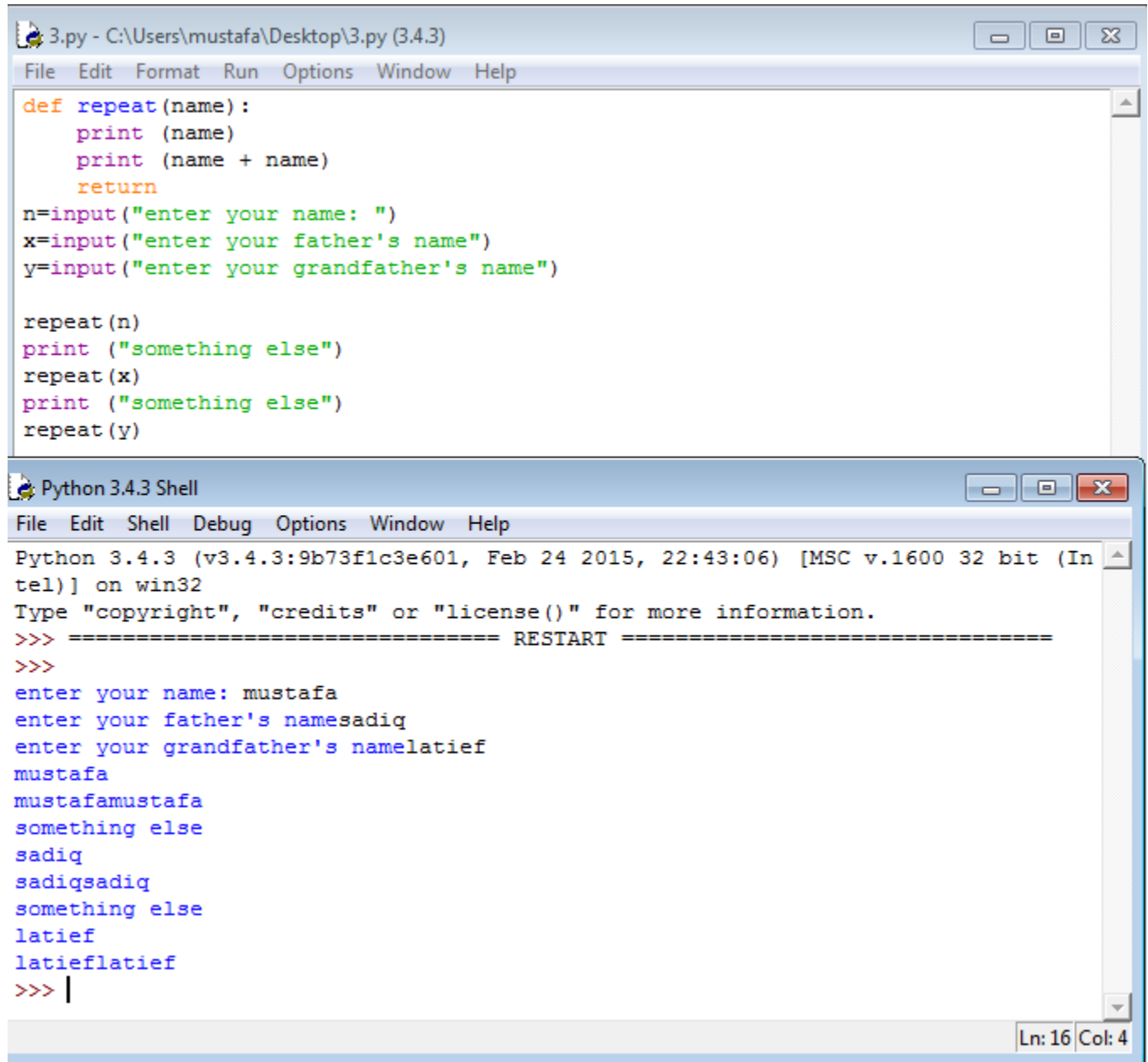
```
def repeat(name):  
    print (name)  
    return  
n=input("enter your name: ")  
x=input("enter your father's name")  
y=input("enter your grandfather's name")  
  
repeat(n)  
print ("something else")  
repeat(x)  
print ("something else")  
repeat(y)
```

Below the editor is a "Python 3.4.3 Shell" window showing the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (In  
tel)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>> ===== RESTART =====  
>>>  
enter your name: mustafa  
enter your father's namesadiq  
enter your grandfather's namelatief  
mustafa  
something else  
sadiq  
something else  
latief  
>>> |
```

The shell window also shows the status "Ln: 13 Col: 4" at the bottom right.

في هذا المثال نلاحظ ان التنفيذ لم يختلف عن المثال السابق بإضافة عبارة (Return) في نهاية الدالة وذلك لأنها هنا عبارة فارغة تقول للمفسر: قم بإرجاع نتيجة تنفيذ الدالة الى البرنامج الرئيسي حين يتم استدعاء الدالة. ولكن لهذه الدالة فوائد أخرى وكما يوضحها المثال التالي:



The image shows a screenshot of a Python IDE with two windows. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
def repeat(name):
    print (name)
    print (name + name)
    return
n=input("enter your name: ")
x=input("enter your father's name")
y=input("enter your grandfather's name")

repeat(n)
print ("something else")
repeat(x)
print ("something else")
repeat(y)
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter your name: mustafa
enter your father's namesadiq
enter your grandfather's namelatief
mustafa
mustafamustafa
something else
sadiq
sadiqsadiq
something else
latief
latieflatief
>>> |
```

The status bar at the bottom right of the shell window indicates 'Ln: 16 Col: 4'.

ولاحظ الفرق في التنفيذ الان:

The screenshot shows a Python IDE window titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)'. The code defines a function 'repeat' and uses it with user input. Below the code is a 'Python 3.4.3 Shell' window showing the execution output.

```

def repeat(name) :
    print (name)
    print (name + name)
    return name
    print (name)
n=input("enter your name: ")
x=input("enter your father's name")
y=input("enter your grandfather's name")

repeat(n)
print ("something else")
repeat(x)
print ("something else")
repeat(y)

```

```

Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter your name: mustafa
enter your father's namesadiq
enter your grandfather's namelatief
mustafa
mustafamustafa
something else
sadiq
sadiqsadiq
something else
latief
latieflatief
>>> |

```

Ln: 16 Col: 4

هنا تم التنفيذ بنفس الطريقة تماماً كما في المثال السابق رغم وجود سطر إضافي للطباعة بعد عبارة (Return) داخل الدالة وهذه من فوائد عبارة الارجاع (return) حيث أنها توّشر نهاية الدالة وتُخبر المفسر ان الدالة قد انتهت وان كل ما ورائها من العبارات المزخفة (indented) ليست ذات أهمية بل ان المفسر يهملها مباشرة.

الان بعد ان شرحنا كيفية انشاء واستدعاء الدوال المعرفة من قبل المبرمجين (user defined functions) قد يتساءل البعض عن ماهية الدوال المبنية بداخل اللغة (built in functions) ولتوضيح فكرتها نقوم بالتذكير بالدوال التالية:

```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 99/100
0.99
>>> int(99/100)
0
>>> float(30/5)
6.0
>>> type(0.99)
<class 'float'>
>>> type(33)
<class 'int'>
>>> type("hello")
<class 'str'>
Ln: 25 Col: 4

```

حيث تعتبر هذه الدوال هي من الدوال الجاهزة المخزونة بداخل لغة البرمجة لتحديد نوع المتغيرات والثوابت والتحويل بينها. المزيد من الأمثلة عن الدوال:

```

3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
x = 5
print ('Hello')

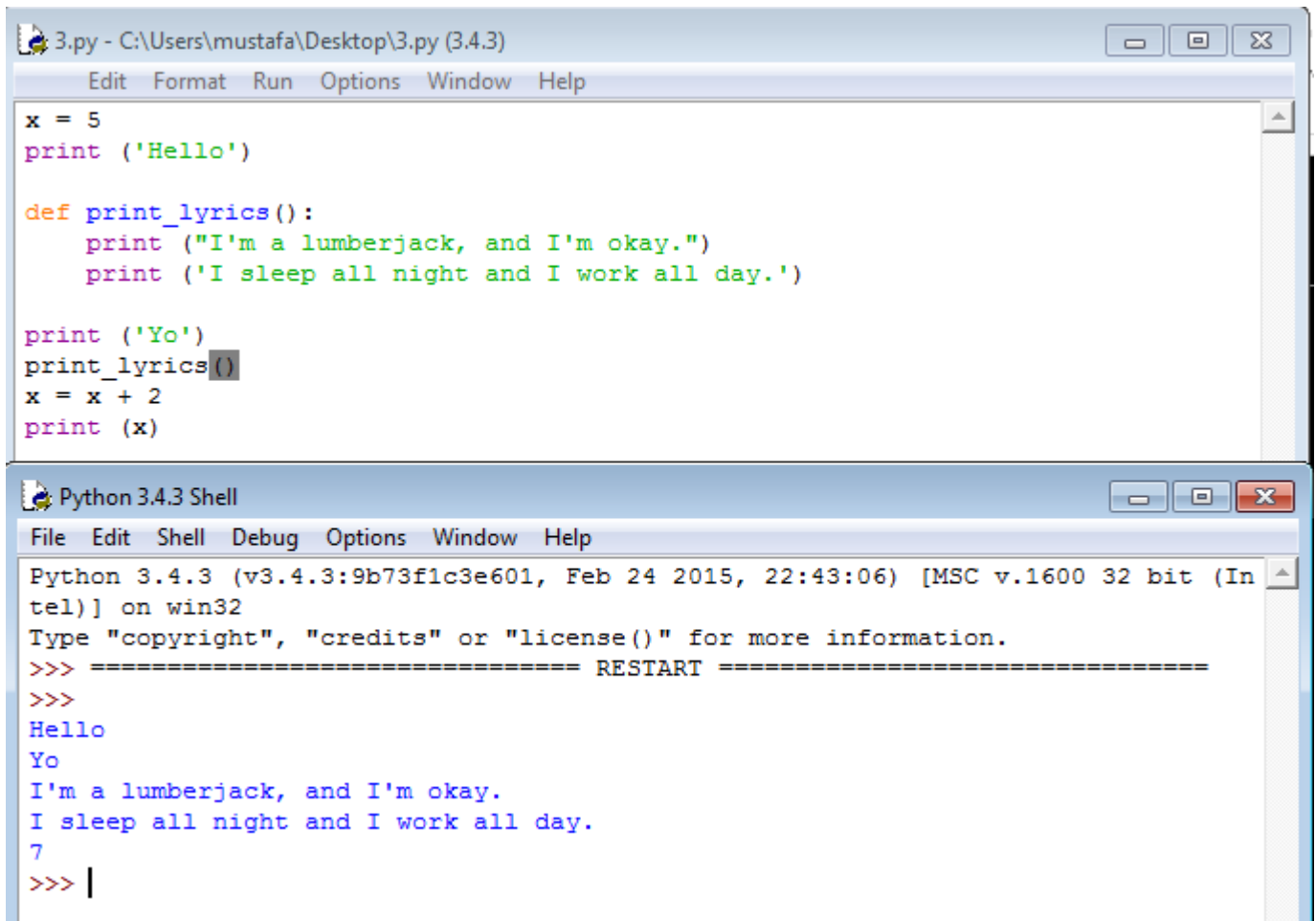
def print_lyrics():
    print ("I'm a lumberjack, and I'm okay.")
    print ('I sleep all night and I work all day.')

print ('Yo')
x = x + 2
print (x)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Hello
Yo
7
>>> |

```

المثال أعلاه لدالة بدون استدعاء.



The image shows two windows from a Python IDE. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
x = 5
print ('Hello')

def print_lyrics():
    print ("I'm a lumberjack, and I'm okay.")
    print ('I sleep all night and I work all day.')

print ('Yo')
print_lyrics()
x = x + 2
print (x)
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Hello
Yo
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
7
>>> |
```

هنا تم استدعاء الدالة فظهرت نتيجة تنفيذها.

والان مثال عن كيفية توظيف الشروط بداخل الدوال:

The screenshot shows a Python IDE window titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)'. The code in the editor is as follows:

```
def greet(lang):
    if lang == 'es':
        print ('Hola')
    elif lang == 'fr':
        print ('Bonjour')
    else:
        print ('Hello')
    return
greet("es")
greet("fr")
greet("xyz")
```

Below the editor is a 'Python 3.4.3 Shell' window showing the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Hola
Bonjour
Hello
>>> |
```

مثال اخر عن العمليات الرياضية واستدعاء الدالة لأكثر من متغير في نفس الوقت:

The screenshot shows a Python IDE window titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)'. The code in the editor is as follows:

```
def addtwo(a, b):
    added = a + b
    return added

x = addtwo(3, 5)
print (x)
```

Below the editor is a 'Python 3.4.3 Shell' window showing the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
8
>>> |
```

- يجدر بالذكر ان بعض الدوال لا ترجع أي قيمة الى البرنامج الرئيسي وانما تستخدم للعمل بشكل مستقل عن البرنامج الرئيسي وفي هذه الحالة تسمى تلك الدوال بالدوال غير المثمرة (unfruitful functions) او بتعبير المبرمجين (void functions).

واخيراً ولمن لم يستوعب فوائد الدوال نذكر النقاط التالية:

- ١- يفضل استخدام الدوال لتنظيم الكود وتمييز مكوناته المتعددة.
- ٢- يفضل استخدام الدوال لتجنب إعادة وتكرار كتابة نفس الاكواد مرات متعددة داخل البرنامج.
- ٣- إذا أصبح البرنامج كبيراً جداً او طويلاً فيفضل تجزئته الى دوال متعددة صغيرة للسيطرة عليه أكثر وتسهيل اكتشاف الأخطاء ومعالجتها.
- ٤- تستخدم الدوال ايضاً في انشاء مكتبات المستخدم (user defined libraries) كما سنتعلم ذلك لاحقاً للكودات التي تحتاج اليها بشكل متكرر وكذلك لتسهيل مشاركة تلك المكتبات مع المبرمجين الاخرين.

4.6 Write a program to prompt the user for hours and rate per hour using `raw_input` to compute gross pay. Award time-and-a-half for the hourly rate for all hours worked above 40 hours. Put the logic to do the computation of time-and-a-half in a function called `compute_pay()` and use the function to do the computation. The function should return a value. Use 45 hours and a rate of 10.50 per hour to test the program (the pay should be 498.75). You should use `raw_input` to read a string and `float()` to convert the string to a number. Do not worry about error checking the user input unless you want to - you can assume the user types numbers properly.

Check Code

Reset Code



Exit

Grade updated on server.

```

1 def computepay(h,r):
2     if h>40:
3         pay=(1.5*r*(h-40))+(40*r)
4         return pay
5     else:
6         pay=r*h
7         return pay
8
9 hrs = raw_input("Enter Hours:")
10 hours=float(hrs)
11 rat=raw_input("enter rate:")
12 rate=float(rat)
13 p = computepay(hours,rate)
14 print p

```

الدرس العاشر: الحلقات التكرارية (While)

بعد ان شرحنا اساسيات البرمجة بلغة بايثون في الدروس السابقة، نصل اليوم الى شرح اول أداة من أدوات الحلقات التكرارية والتي تختص بالحلقات التكرارية الغير محددة المدى (indefinite loops) حيث لا نعرف كم مرة سيتم تنفيذ الحلقة التكرارية بالضبط وانما يعتمد ذلك على نوع مدخلات المستخدمين وظروف التنفيذ والاداة المستخدمة لذلك هي نفسها المستخدمة في الكثير من لغات البرمجة الأخرى مثل السي والسي بلس بلس والجافا وهي (while) واما كيفية استخدامها فيوضحها المثال التالي:

```

3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help

n = 5
while n > 0 :
    print (n)
    n = n-1
print ('Blastoff!')
print (n)

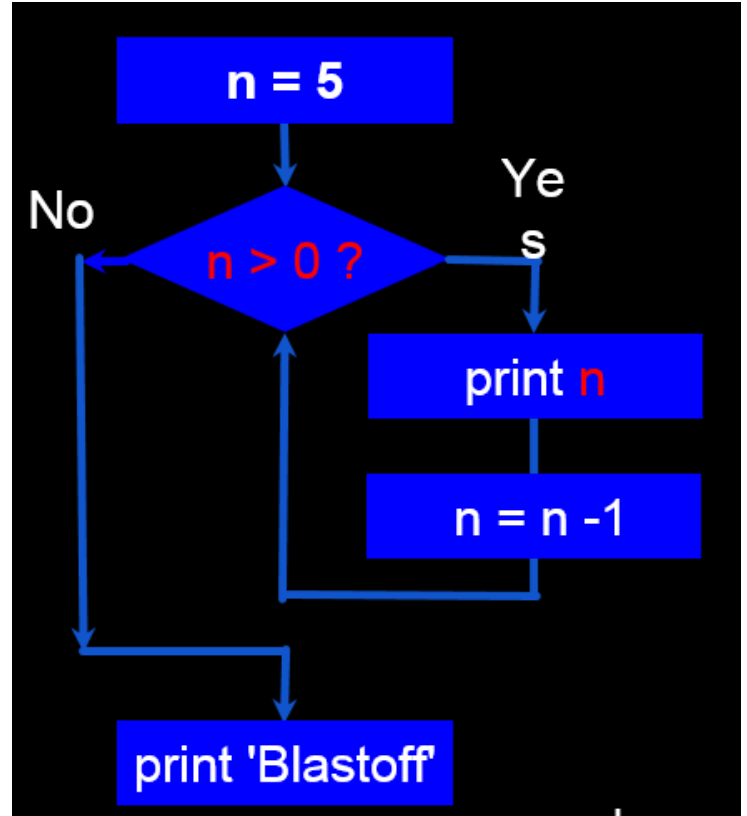
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help

Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
5
4
3
2
1
Blastoff!
0
>>>

```

وهنا نلاحظ اننا اعطينا قيمة أولية للمتغير (n) ثم كتبنا العبارة (while n>0:) والتي تقول للمفسر: طالما ان ال(n) أكبر من السفر استمر في التنفيذ وكرر كل ما بداخل عبارة (while). وهنا وكما ذكرنا سابقاً لا تمتاز عبارة (while) عن بقية البرنامج بأقواس تحدد بداية ونهاية محتوياتها وانما بالمسافات (indents). وبعد ان يختبر المفسر كون الشرط صحيح يقوم بالدخول الى داخل عبارة (while) وينفذ محتوياتها وهي هنا طباعة قيمة (n) ثم إنقاصها واحد وتكرار اختبار الشرط وهكذا حتى يصبح الشرط غير صحيح (حين تصبح n صفر) فيخرج المفسر من اللوب (الحلقة التكرارية) وينفذ ما بعده وهي عبارات طباعة فقط.

لتوضيح الامر أكثر لاحظ المخطط التوضيحي التالي:

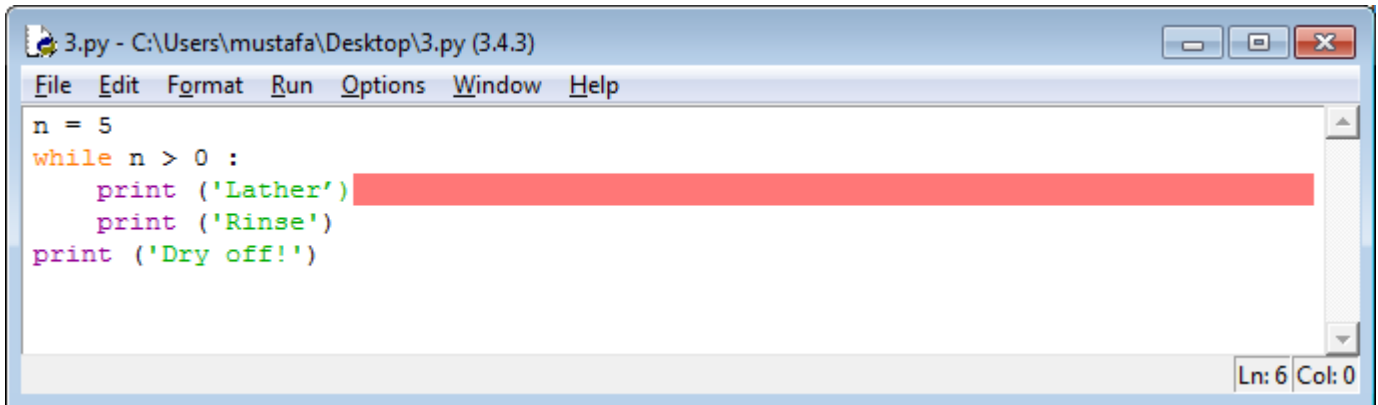


وهنا قمنا بتوضيح الية التنفيذ بشكل رسومي للتوضيح أكثر.

مشاكل التعامل مع عبارة (while):

لكيلا يتم فهم العنوان هنا بشكل خاطئ، فإن عبارة **while** ممتازة في الكثير من الحالات ولا اشكال في استخدامها في البرمجة ولكن بشروط:

- تحديد نهاية متوقعة للتنفيذ ومنع الحلقات اللامتناهية (infinity loops) وكما سنرى.
- تحديد شروط أولية للعبارة تسمح بالدخول الى داخل عبارة **while** والا فاستخدامها عبثي وكما سنرى ايضاً.

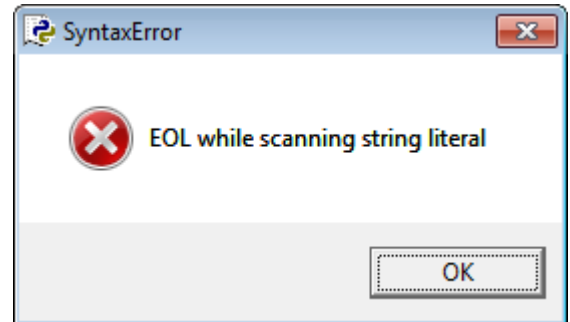
الحلقة اللانهائية:


```

3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
n = 5
while n > 0 :
    print ('Lather')
    print ('Rinse')
print ('Dry off!')
Ln: 6 Col: 0

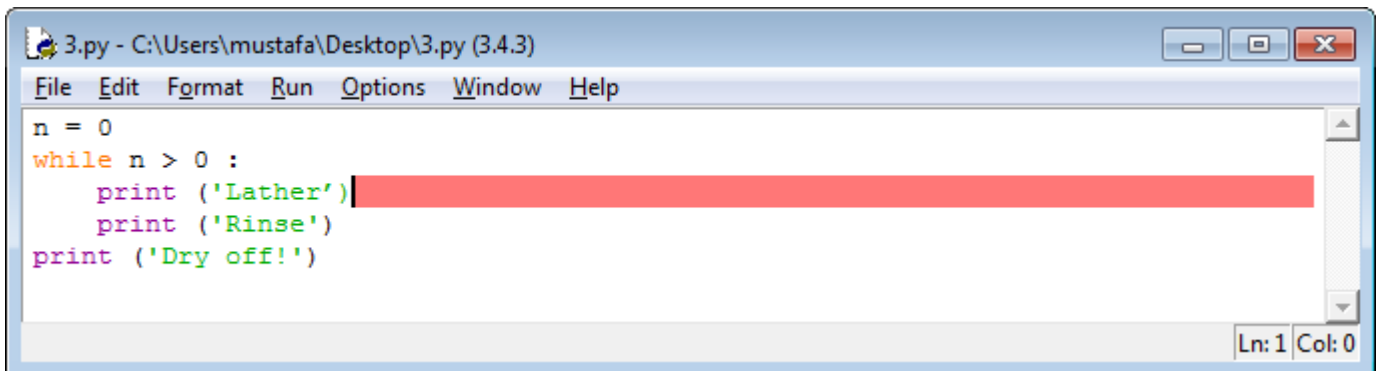
```

هنا نلاحظ ان شرط `while` سيبقى صحيحاً الى ما لا نهاية ولذا تظهر الرسالة التالية عند محاولة التنفيذ:



وهنا يخبرنا المفسر ان نهاية الحلقة التكرارية غير معرفة والمشكلة هنا اننا لم نضع عبارة تحديد نهاية اللوب مثل

(`n=n-1`) او غيرها من العبارات التي تغير قيمة المتغير في الشرط.

عبارة (while) بلا فائدة:


```

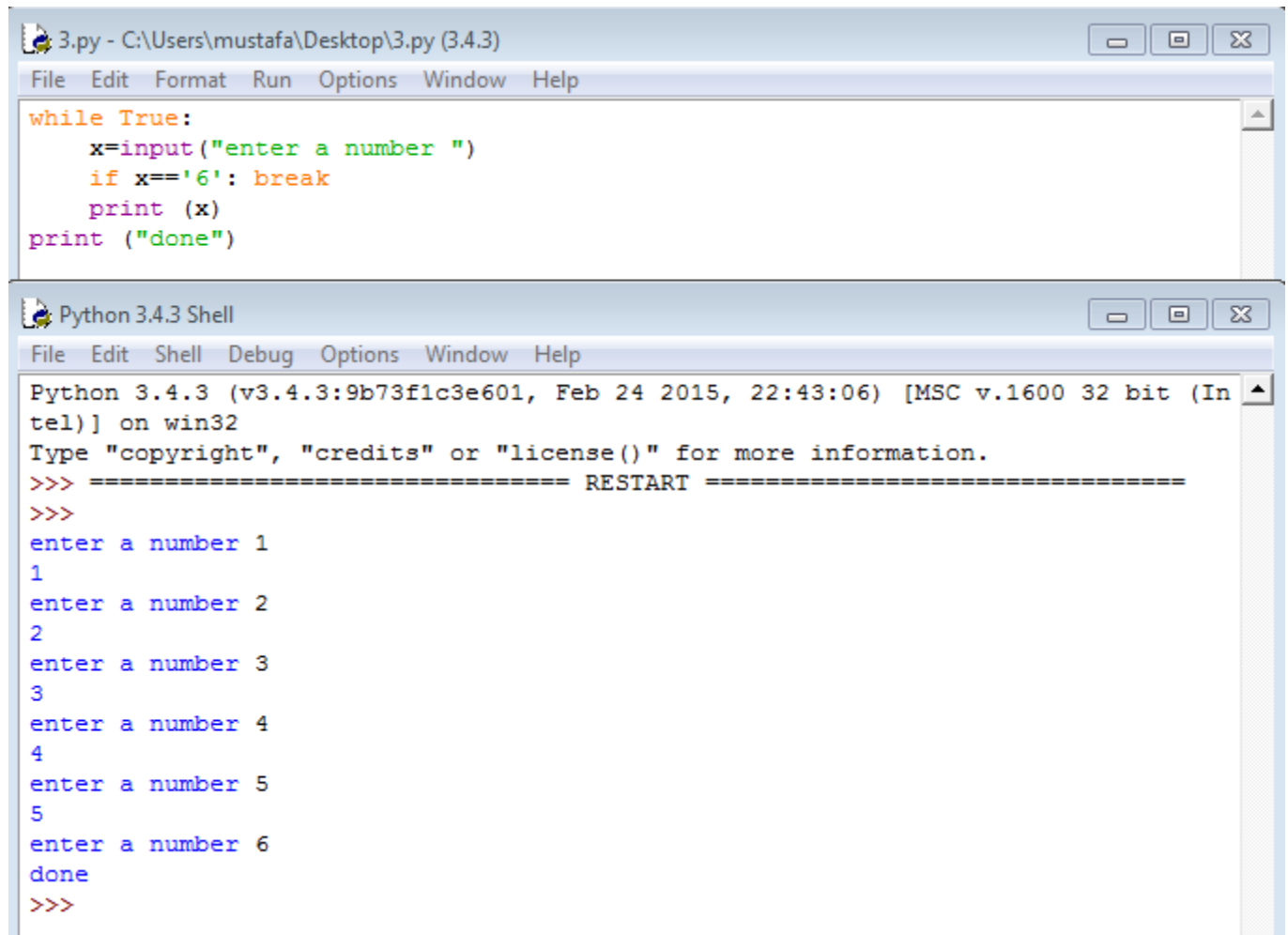
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
n = 0
while n > 0 :
    print ('Lather')
    print ('Rinse')
print ('Dry off!')
Ln: 1 Col: 0

```

هنا نلاحظ ان شرط (while) لن يتحقق ابداً ولذا فلا فائدة من وجودها اصلاً. وهذه امثلة على الأخطاء التي يقع فيها المبرمجون حين التعامل مع عبارة (While) لذا يجب الحذر.

كسر حلقة التكرار بشرط معين:

نحتاج بعض الأحيان الى إيقاف الحلقة التكرارية حين يتحقق شرط معين مثل وجود رقم معين كنا نبحث عنه او ادخال المستخدم لقيمة معينة كنا نبحث عنها وهنا نستخدم عبارة (break) والتي نقول للمفسر: قم بإنهاء الحلقة التكرارية واقفز الى ما بعدها. وكما في المثال التالي:



```

3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
while True:
    x=input("enter a number ")
    if x=='6': break
    print (x)
print ("done")

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter a number 1
1
enter a number 2
2
enter a number 3
3
enter a number 4
4
enter a number 5
5
enter a number 6
done
>>>

```

هنا قمنا بوضع شرط لكسر حلقة التكرار وهو ادخال قيمة ٦ وقد حصل ذلك كما في نافذة التنفيذ أعلاه.

ترك تنفيذ الحلقة في المنتصف بشرط معين:

في بعض الأحيان نحتاج الى ترك الحلقة التكرارية الحالية في المنتصف والقفز الى بداية الحلقة من جديد لاستئناف التنفيذ حين يتحقق شرط معين ونستخدم عبارة (continue) لهذا الغرض وكما في المثال ادناه:

The screenshot shows a Python IDE window titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)'. The code in the editor is as follows:

```
while True:
    x=input("enter a number ")
    if x=='6': break
    if x== 'jump': continue
    print (x)
print ("done")
```

Below the editor is a 'Python 3.4.3 Shell' window showing the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter a number 1
1
enter a number 2
2
enter a number 3
3
enter a number 4
4
enter a number 4
4
enter a number 66
66
enter a number 77
77
enter a number 88
88
enter a number jump
enter a number 6
done
>>> |
```

The status bar at the bottom right of the shell window shows 'Ln: 24 Col: 4'.

هنا استخدمنا نفس المثال السابق مع إضافة شرط يقول (إذا كانت قيمة الإدخال تساوي jump فقم بالقفز الى نهاية الحلقة التكرارية بدون اكمالها) وهذا ما حصل حيث اننا حين قمنا بإدخال عبارة jump قام المفسر بالقفز الى بداية حلقة جديدة

بدون اكمال الحلقة الحالية ويتضح ذلك برؤية انه لم يطبع عبارة `jump` أي انه أهمل الايعاز (`print (x)`) بعد عبارة `continue` حين تحقق شرطها.

مثال اخر على كيفية التعامل مع (`continue`) و (`break`):

```

3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help

while True:
    line = input("enter a name ")
    if line[0] == '#' : continue
    if line == 'done' : break
    print (line)
print ('Done!')

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help

Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter a name ali
ali
enter a name ahmad
ahmad
enter a name kalid
kalid
enter a name mustafa
mustafa
enter a name #hello
enter a name #father
enter a name done
Done!
>>> |

```

والشرط للقفز هنا كان ادخال عبارة اول حرف فيها (`line[0]`) هو الرمز (`#`) حيث لا يتم طباعته واما لكسر الحلقة فيكفي ادخال عبارة (`done`) وهكذا.

ملاحظة: كل ما شرحناه هنا عن عبارة (`while`) يسمى الحلقات التكرارية غير المحددة لأنها كما رأينا لا تلتزم بشرط توقف معروف وانما تعتمد في اغلب الأحيان على الدخالات المستخدم وظروف التنفيذ ونتائج العمليات السابقة واما ما سنشرحه ان شاء الله في الدرس القادم فسيكون متركزاً على الحلقات التكرارية المحددة (`definite loops`) والتي يمكن برمجتها باستخدام الأداة الشهيرة في كل لغات البرمجة تقريباً (`for`) فتابعوا معنا ☺

مثال أخير لاعتماد شرط التوقف على العمليات الرياضية المتكررة:

```

3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help

while True:
    x=input("enter a number ")
    y=input("enter a second number ")
    x1=int(x)
    y1=int(y)
    z=x1+y1
    if z>10: break
    if z==10: continue
    if z<10: print (z)
print ("done!")

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help

Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
enter a number 1
enter a second number 2
3
enter a number 3
enter a second number 4
7
enter a number 3
enter a second number 5
8
enter a number 5
enter a second number 5
enter a number 6
enter a second number 7
done!
>>> |

```

هنا كان شرط الاستمرار (continue) ان يكون ناتج الجمع ١٠ و شرط كسر الحلقة التكرارية (break) ان يكون الناتج أكثر من ١٠ واما التنفيذ الطبيعي للحلقة التكرارية كاملة فيكون لقيم ناتج الجمع أصغر من ١٠.

الدرس الحادي عشر: الحلقات التكرارية باستخدام (for)

بعد ان شرحنا كيفية برمجة الحلقات التكرارية الغير محددة المدى (indefinite loops) باستخدام عبارة (while) في الدرس الماضي، نأتي اليوم الى شرح عبارة (for) التي تستخدم لعمل حلقات تكرارية محددة (definite loops) وصيغتها العامة تختلف قليلاً عما كانت عليه في لغات سي وسي بلس بلس وجافا وهي كالآتي:

for (iterations variable) in (list of numbers, names, ...etc.) :

statement

statement

the rest of the program

حيث يمثل (iteration variable) متغير التكرار وهو المتغير الذي سيحدد كم مرة سيتم تكرار تنفيذ ما بداخل عبارة (for).

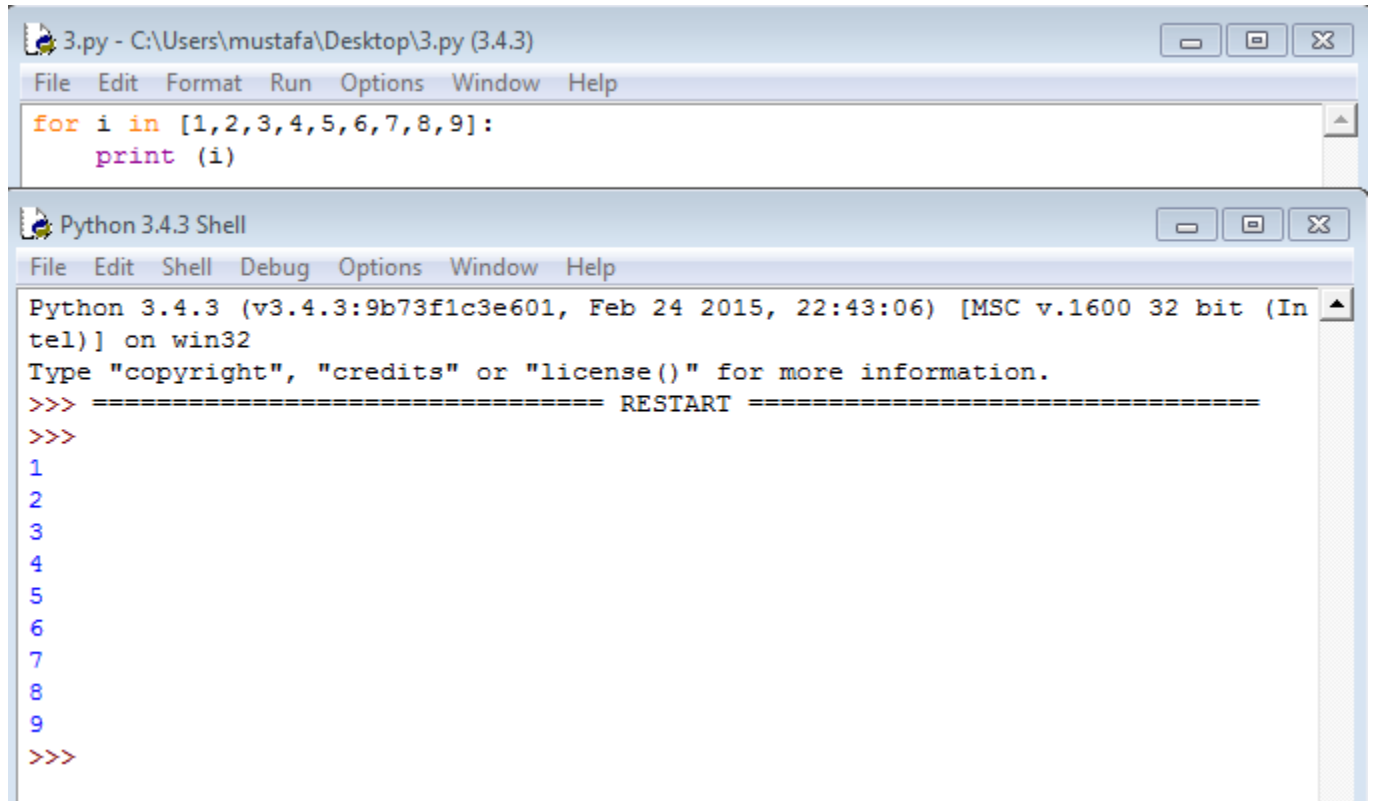
واما (list of numbers, names, ...etc.) فهو المدى او قائمة الأرقام او الأسماء او المتغيرات التي سيتحرك ضمنها متغير التكرار ليأخذ قيمها كل مرة.

واخيراً (statement) هي العبارات داخل عبارة (for) والتي تتكرر اعتماداً على متغير التكرار ونلاحظ انها مزحفة الى اليمين بمقدار 4 فراغات (4 spaces) كما في كل العبارات الأخرى لتحديد بداية ونهاية العبارات التابعة لل (for).

اما (the rest of the program) فهو تكملة البرنامج ونلاحظ انه غير مزحف أي انه يبدأ من بداية السطر ليحدد انه غير تابع لعبارة (for).

ملاحظة مهمة جداً: يجب مراعاة عدم نسيان (:) في نهاية كل من عبارات الشروط والتكرار فهي مهمة جداً ولا ينفذ البرنامج بدونها.

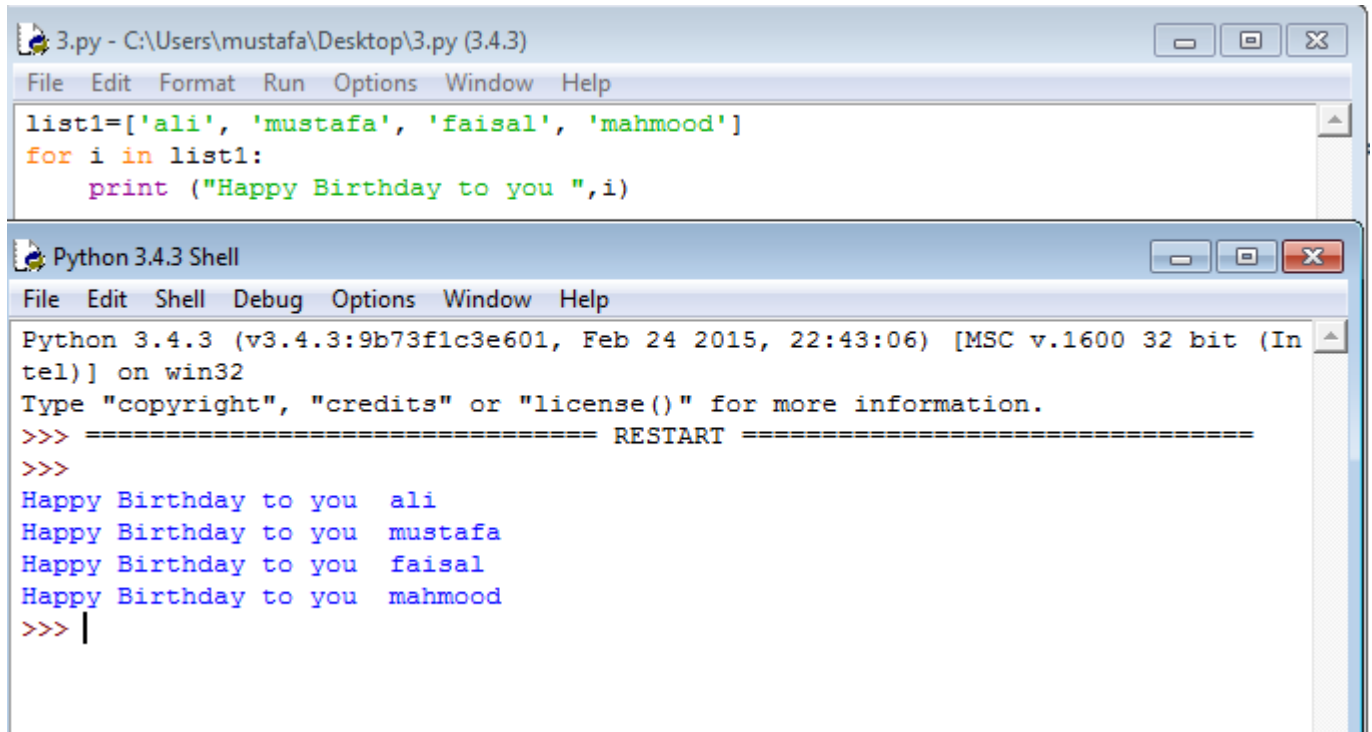
وفي ادناه مثال بسيط يوضح الفكرة:



```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
for i in [1,2,3,4,5,6,7,8,9]:
    print (i)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
1
2
3
4
5
6
7
8
9
>>>
```

وهذا أبسط مثال على كيفية استخدام عبارة (for) حيث عرفنا متغير التكرار على ان اسمه (i) وهو يأخذ قيمه من القائمة التي تبدأ من 1 وتنتهي بال 9 واخيراً وبداخل ال(For) قلنا للمفسر فقط اطبع لنا قيم (i).
امثلة أخرى أكثر تفصيلاً:



The image shows a screenshot of a Python IDE. The top window is titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)' and contains the following code:

```
list1=['ali', 'mustafa', 'faisal', 'mahmood']
for i in list1:
    print ("Happy Birthday to you ",i)
```

The bottom window is titled 'Python 3.4.3 Shell' and shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Happy Birthday to you  ali
Happy Birthday to you  mustafa
Happy Birthday to you  faisal
Happy Birthday to you  mahmood
>>> |
```

هنا كانت قيم متغير التكرار (i) هي أسماء من قائمة أسماء وليست ارقام.

The image shows a screenshot of a Python IDE. The top window is titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)' and contains the following Python code:

```

first=[1,2,3,4,5]
second=[1,2,3,4,5]
print ("Multiplication Table for 1-5")
for i in first:
    for j in second:
        print (i, " * ", j, " = ", i*j)

```

The bottom window is titled 'Python 3.4.3 Shell' and shows the execution output:

```

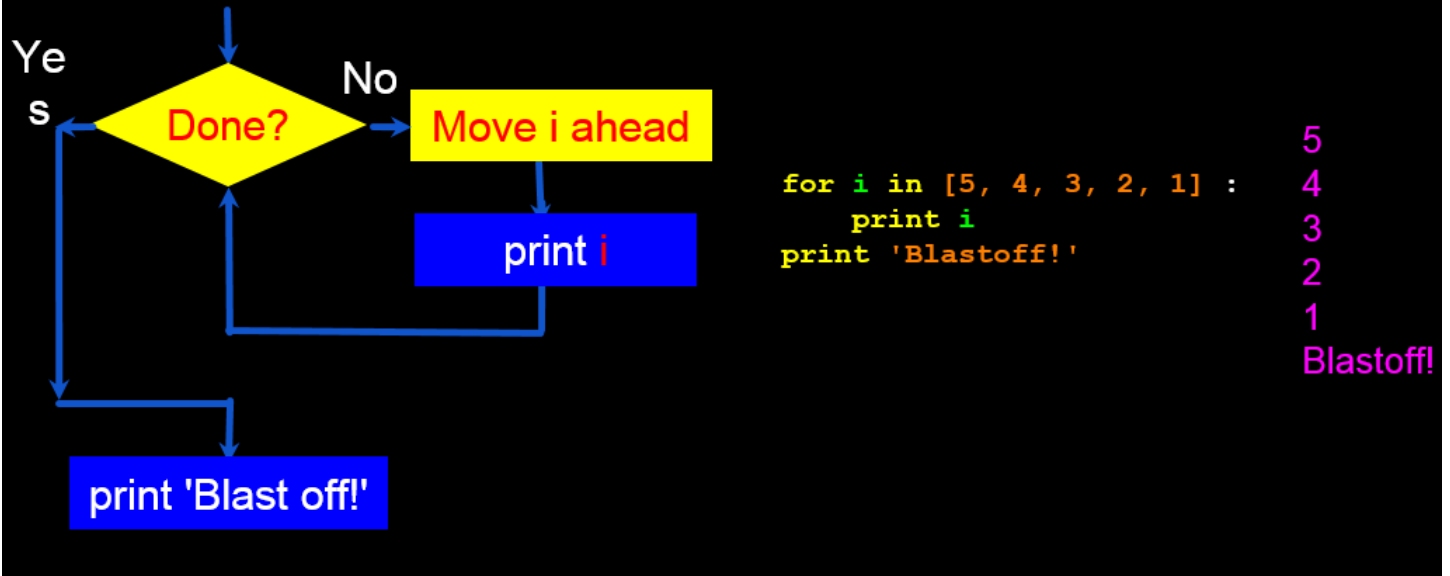
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Multiplication Table for 1-5
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
>>>

```

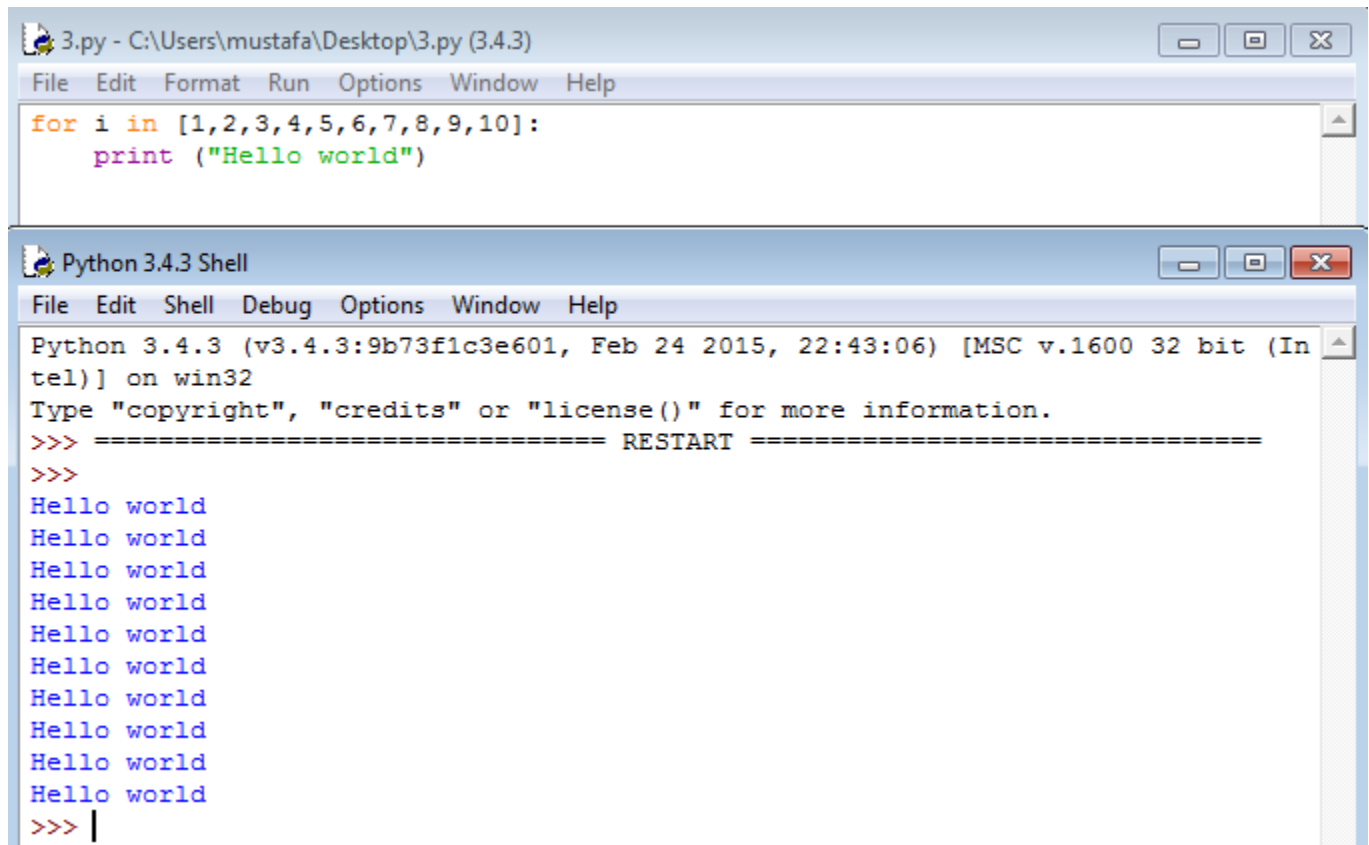
جدول الضرب للأعداد من ١ الى ٥ موضحاً في المثال أعلاه.

واما لمعرفة كيفية فهم المفسر لعمل عبارة (for) وكيفية تسلسل تنفيذ عباراتها الداخلية فالمخطط التالي يشرح ذلك ببساطة:

تعريف مبسط لتسلسل التنفيذ



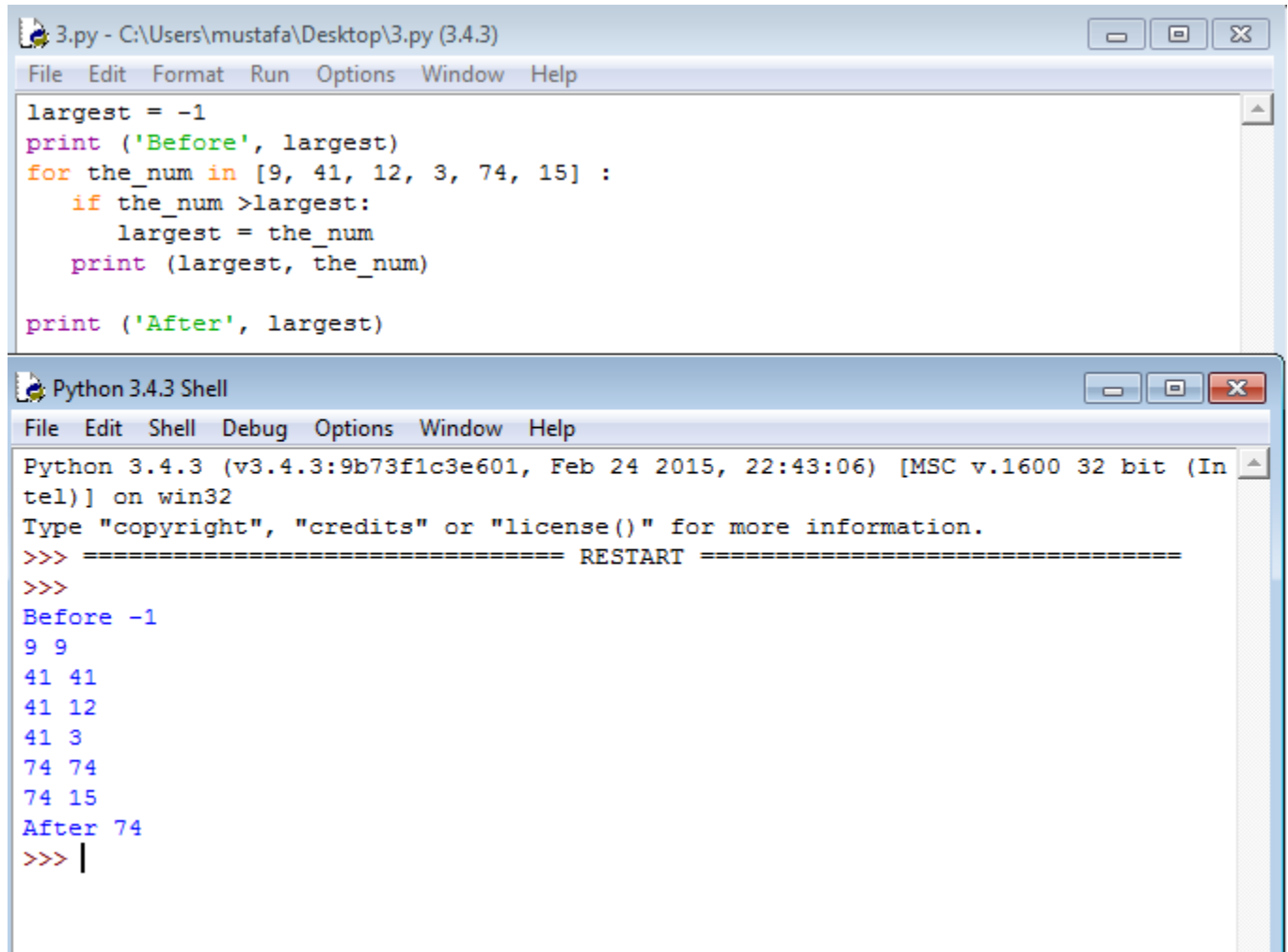
حيث ان الامر كما هو مسمى (حلقة تكرارية) يقوم فيها المفسر بأسناد قيم الى متغير التكرار من القائمة في كل مرة احد القيم وينفذ العبارات داخل ال (For) ثم يعود ليسند له القيمة التالية وهكذا:



```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
for i in [1,2,3,4,5,6,7,8,9,10]:
    print ("Hello world")

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
>>> |
```

المثال أعلاه يبين ان متغير التكرار لا يشترط ان يكون موجوداً في عبارات داخل عبارة (For) بل انه يمكن ان يستخدم كمتغير تكرار للمفسر فقط وليس للمستخدمين.



The image shows two windows from a Python IDE. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
largest = -1
print ('Before', largest)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num >largest:
        largest = the_num
    print (largest, the_num)

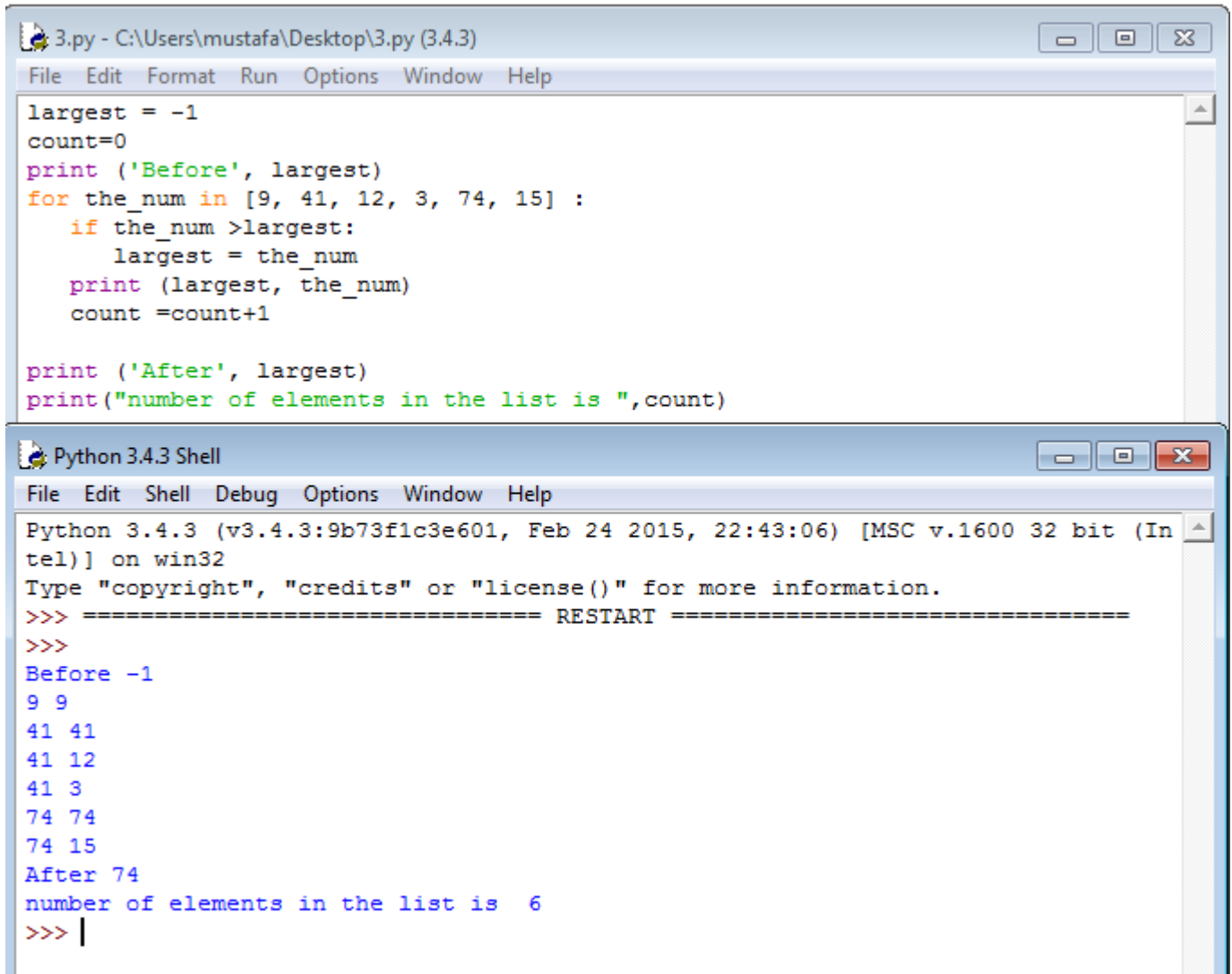
print ('After', largest)
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Before -1
9 9
41 41
41 12
41 3
74 74
74 15
After 74
>>> |
```

برنامج لإيجاد العدد الأكبر في قائمة اعداد في الصورة أعلاه.

والان نفس البرنامج مع إضافة قابلية حساب عدد مرات التكرار للحلقة التكرارية وكما في ادناه:



The image shows a screenshot of a Python IDE with two windows. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

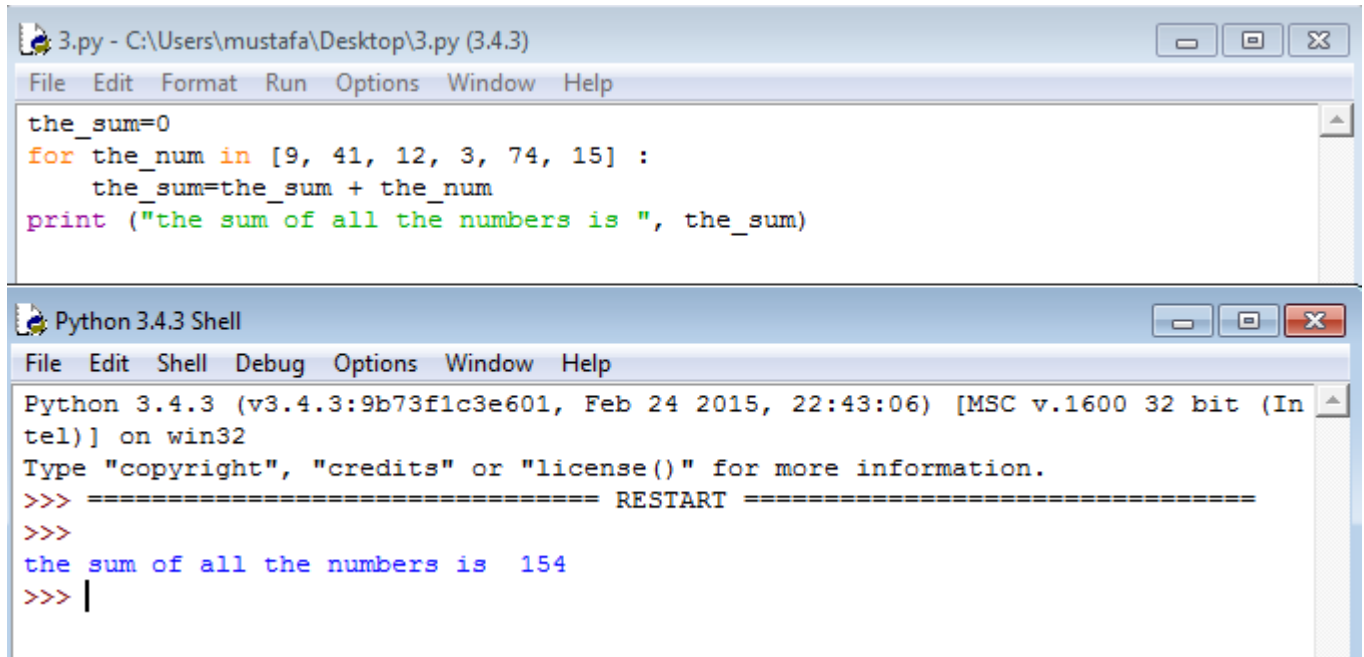
```
largest = -1
count=0
print ('Before', largest)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num >largest:
        largest = the_num
        print (largest, the_num)
        count =count+1

print ('After', largest)
print("number of elements in the list is ",count)
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Before -1
9 9
41 41
41 12
41 3
74 74
74 15
After 74
number of elements in the list is 6
>>> |
```

لأيجاد مجموع قيم متغير التكرار يمكن الاستعانة بالبرنامج التالي:



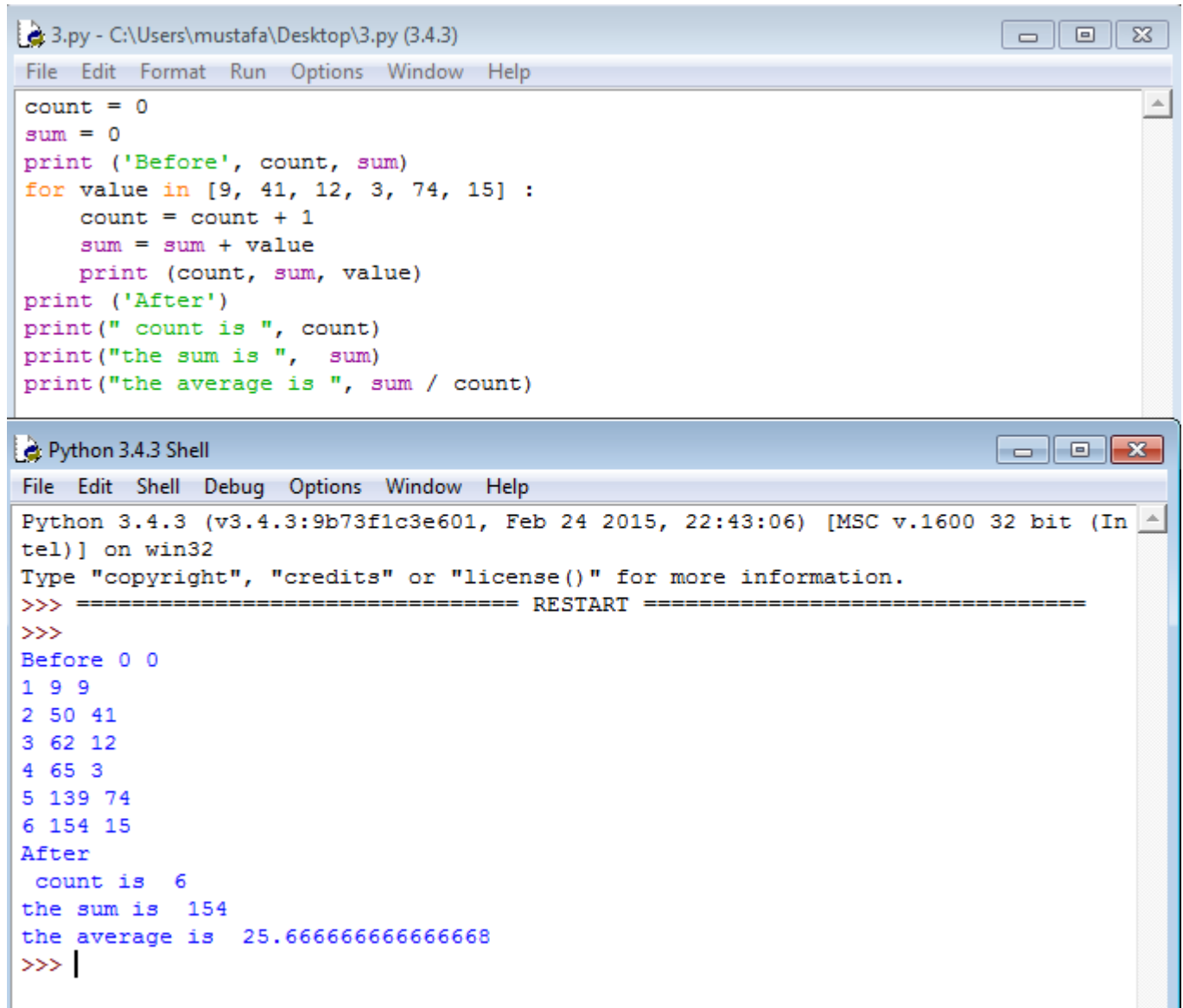
The image shows two windows from a Python IDE. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
the_sum=0
for the_num in [9, 41, 12, 3, 74, 15] :
    the_sum=the_sum + the_num
print ("the sum of all the numbers is ", the_sum)
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
the sum of all the numbers is 154
>>> |
```

والان لحساب معدل مجموعة من الأرقام نلاحظ المثال التالي:



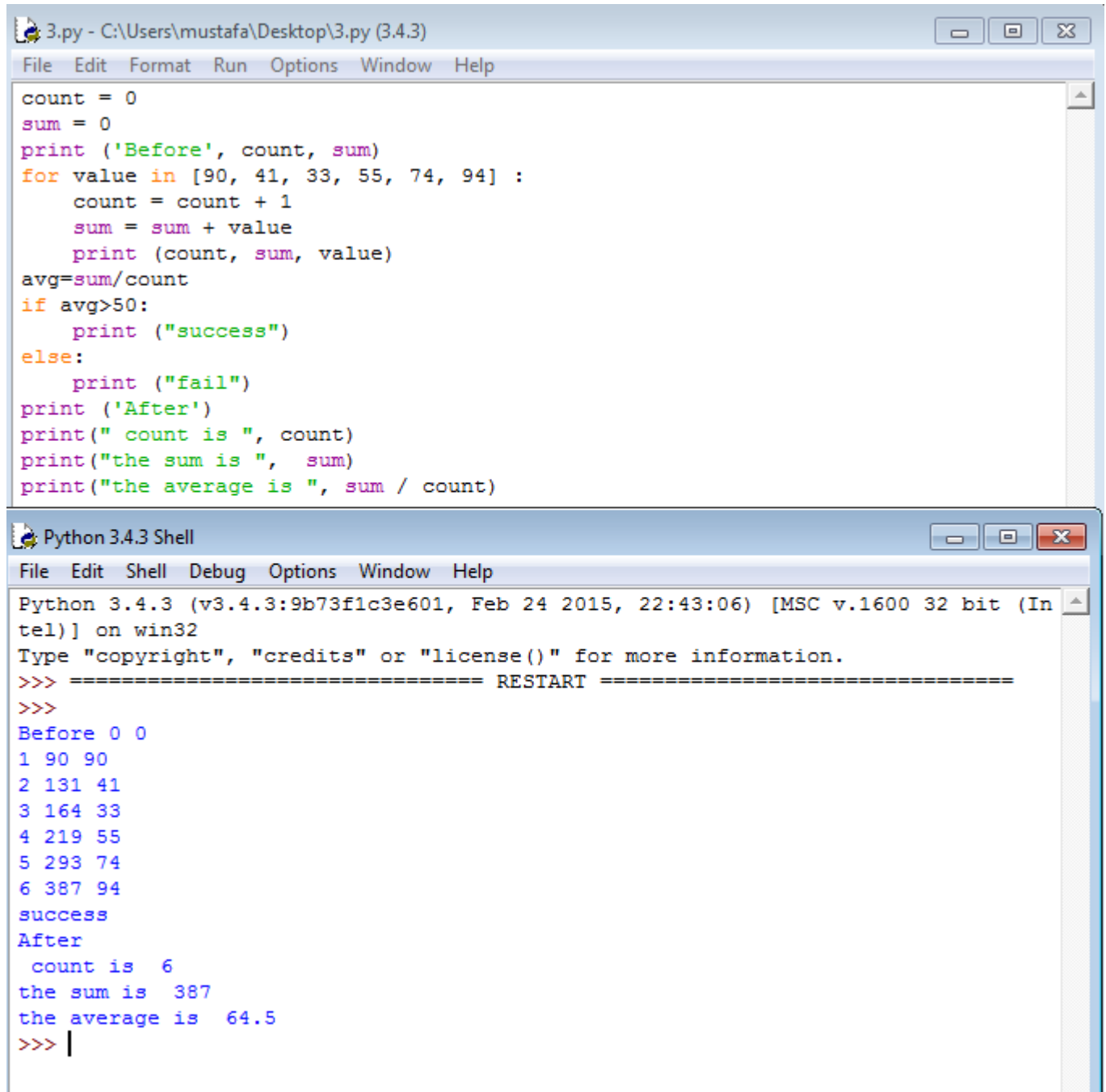
The image shows two windows from a Python IDE. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
count = 0
sum = 0
print ('Before', count, sum)
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print (count, sum, value)
print ('After')
print(" count is ", count)
print("the sum is ", sum)
print("the average is ", sum / count)
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Before 0 0
1 9 9
2 50 41
3 62 12
4 65 3
5 139 74
6 154 15
After
 count is  6
the sum is  154
the average is  25.666666666666668
>>> |
```

ولأستخدام كل من عبارات التكرار والشروط يمكن ملاحظة المثال التالي:



The image shows two windows from a Python IDE. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
count = 0
sum = 0
print ('Before', count, sum)
for value in [90, 41, 33, 55, 74, 94] :
    count = count + 1
    sum = sum + value
    print (count, sum, value)
avg=sum/count
if avg>50:
    print ("success")
else:
    print ("fail")
print ('After')
print(" count is ", count)
print("the sum is ", sum)
print("the average is ", sum / count)
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Before 0 0
1 90 90
2 131 41
3 164 33
4 219 55
5 293 74
6 387 94
success
After
 count is  6
the sum is  387
the average is  64.5
>>> |
```

الدرس الثاني عشر: المزيد عن الحلقات التكرارية

بعد ان شرحنا العبارات المستخدمة في الحلقات التكرارية مثل (while) و (for) نأتي اليوم الى مناقشة مشكلة يوضحها المثال التالي:

شرحنا في المحاضرة السابقة كيفية إيجاد العدد الأكبر من ضمن سلسلة (قائمة او مصفوفة) من الاعداد وكما في الصورة التالية:

<pre> 3.py - C:\Users\mustafa\Desktop\3.py (3.4.3) File Edit Format Run Options Window Help largest_so_far = -1 print ('Before', largest_so_far) for the_num in [9, 41, 12, 3, 74, 15] : if the_num > largest_so_far : largest_so_far = the_num print (largest_so_far, the_num) print ('After', largest_so_far) </pre>	<pre> Python 3.4.3 Shell File Edit Shell Debug Options Window Help Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) tel)] on win32 Type "copyright", "credits" or "license()" for more inform >>> ===== RESTART ===== >>> Before -1 9 9 41 41 41 12 41 3 74 74 74 15 After 74 >>> </pre>
--	--

وهنا كما هو واضح اعتمدنا على إعطاء قيمة أولية للمتغير الأكبر (largest_so_far) وهي قيمة صغيرة مقدارها سالب واحد وقد حصلنا على نتائج صحيحة ولكن ماذا لو كانت كل الاعداد في القائمة سالبة واصغر من السالب واحد؟

دعونا نرى ماذا سيكون الإخراج:

<pre> 3.py - C:\Users\mustafa\Desktop\3.py (3.4.3) File Edit Format Run Options Window Help largest_so_far = -1 print ('Before', largest_so_far) for the_num in [-9, -41, -12, -3, -74, -15] : if the_num > largest_so_far : largest_so_far = the_num print (largest_so_far, the_num) print ('After', largest_so_far) </pre>	<pre> Python 3.4.3 Shell File Edit Shell Debug Options Window Help Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC tel)] on win32 Type "copyright", "credits" or "license()" for more informatio >>> ===== RESTART ===== >>> Before -1 -1 -9 -1 -41 -1 -12 -1 -3 -1 -74 -1 -15 After -1 >>> </pre>
--	--

كما هو متوقع تماماً، قام المفسر بمقارنة القيمة الأولية وهي سالب واحد بكل القيم الأخرى التي يصدف أنها كلها اصغر منه فقام بالقول بأن اكبر قيمة في القائمة هي سالب واحد مع العلم ان القائمة لا تحتوي هذا الرقم وهو خطأ كبير يؤشر حاجتنا الى شيء اخر لتصحيح البرنامج.

مثال اخر على نفس المشكلة: وهو البحث عن اصغر عدد في القائمة وكما في الصورة ادناه:

The screenshot shows a Python IDE with two windows. The left window is titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)' and contains the following code:

```
smallest_so_far = -1
print ('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print (smallest_so_far, the_num)

print ('After', smallest_so_far)
```

The right window is titled 'Python 3.4.3 Shell' and shows the output of the script:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.16
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Before -1
-1 9
-1 41
-1 12
-1 3
-1 74
-1 15
After -1
>>> |
```

نفس المشكلة، لأن القيمة الأولية للمتغير (`smallest_so_far`) اصغر من كل القيم في القائمة فقد اظهر المفسر نتيجة تقول بأن الأصغر في القائمة هو سالب واحد علماً ان القائمة لا تحتوي هذا المتغير وهو برنامج خاطيء اخر!

إذا ما الحل لذلك؟

الجواب ببساطة: القيمة الافتراضية (`None`). نعم انها كلمة مفتاحية محجوزة في لغة بايثون وتستخدم للتعامل مع هذا النوع من المشاكل حيث يتم إعطاء قيمة أولية للمتغير هي (`None`) ثم يقوم المتغير بأخذ قيمة اول متغير او ثابت في القائمة او المصفوفة المراد البحث بداخلها عن الأكبر او الأصغر وكما في المثال التالي:

```

3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
smallest = None
print ('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print (smallest, value)
print ('After', smallest)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1
tel]) on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Before
9 9
9 41
9 12
3 3
3 74
3 15
After 3
>>> |

```

والان لنشرح خطوات البرنامج:

السطر الأول: إعطاء قيمة أولية للمتغير (smallest) مقدارها (None).

السطر الثاني: طباعة عبارة (before)

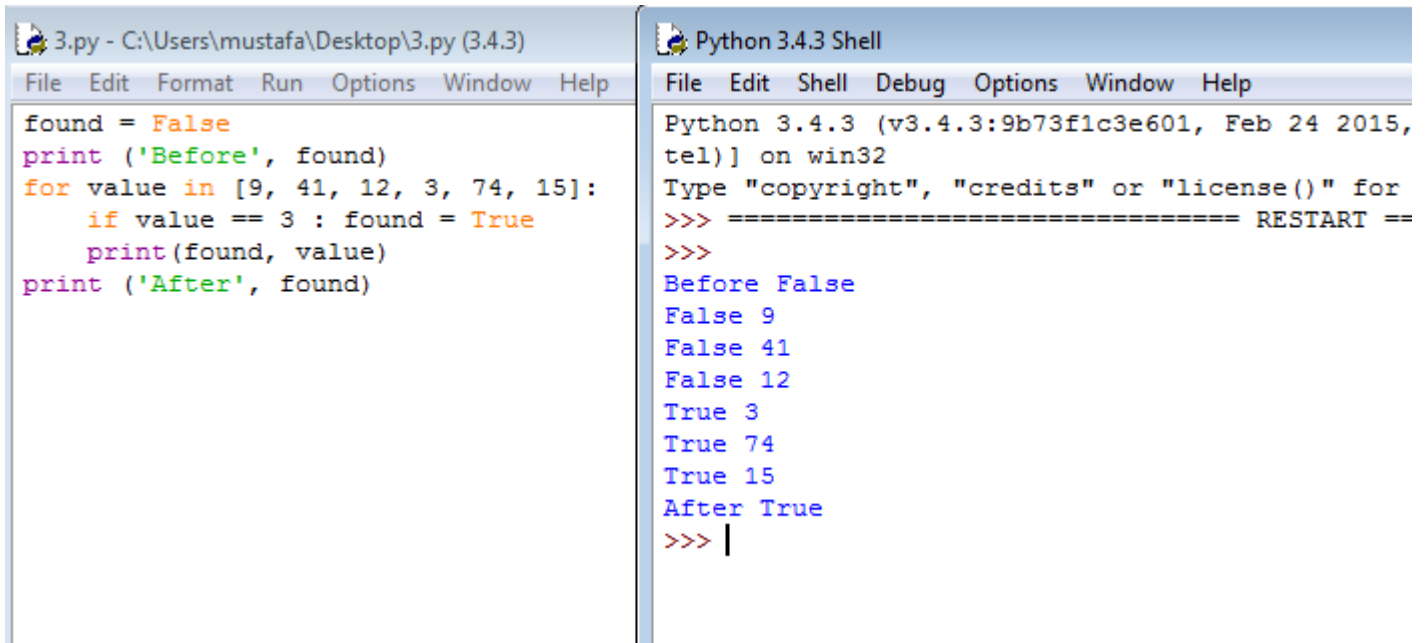
السطر الثالث: عبارة for للبحث بداخل القائمة

السطر الرابع: وهو مهم جداً والهدف منه اختبار شرط يتحقق مرة واحدة فقط عند بداية اللوب عندما تكون (smallest=None) فيقوم بأسناد قيمة المتغير الأول في القائمة (الذي قيمته الان مخزونة في value) الى المتغير (smallest) في السطر الخامس ثم يبدأ بعدها بالمقارنة.

السطر السادس: عملية المقارنة المتكررة لكل عنصر في القائمة بالمتغير (smallest) حتى اذا تحقق الشرط ان احد عناصر القائمة اصغر من (smallest) فيقوم المتغير (smallest) بأخذ قيمة ذلك العنصر كما في السطر السابع.

واخيراً السطر الثامن طباعة نتيجة كل حلقة تكرارية قبل البدء بها من جديد والسطر الأخير طباعة النتيجة النهائية بعد اكمال الحلقة التكرارية.

والان نأخذ مثال على كيفية البحث عن عنصر معين في قائمة باستخدام القيم المنطقية صح (True) وخطأ (False):



The image shows a screenshot of a Python IDE with two windows. The left window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
found = False
print ('Before', found)
for value in [9, 41, 12, 3, 74, 15]:
    if value == 3 : found = True
    print(found, value)
print ('After', found)
```

The right window, titled 'Python 3.4.3 Shell', shows the output of the script:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015,
tel)] on win32
Type "copyright", "credits" or "license()" for
>>> ===== RESTART ==
>>>
Before False
False 9
False 41
False 12
True 3
True 74
True 15
After True
>>> |
```

وهنا نلاحظ ان القيمة الأولية ليست رقم ولا قيمة (None) وانما القيمة المنطقية (False) والتي تصبح (True) حين العثور على المتغير او القيمة المطلوبة وهكذا.

واخيراً في الصورة ادناه مثال محلول عن كيفية استخدام الحلقات التكرارية بشكل احترافي:

5.2 Write a program that repeatedly prompts a user for integer numbers until the user enters 'done'. Once 'done' is entered, print out the largest and smallest of the numbers. If the user enters anything other than a valid number catch it with a try/except and put out an appropriate message and ignore the number. Enter the numbers from the book for problem 5.1 and Match the desired output as shown.

Check Code

Reset Code



Exit

Grade updated on server.

```

1 largest = None
2 smallest = None
3 while True:
4     num = raw_input("Enter an integer number: ")
5     if num == "done" : break
6     try:
7         number=int(num)
8         if largest is None:
9             if smallest is None:
10                largest=number
11                smallest=number
12            elif number>largest:
13                largest=number
14            elif number<smallest:
15                smallest=number
16        except:
17            print "Invalid input"
18    print "Maximum is", largest
19    print "Minimum is", smallest

```

الى هنا ينتهي الجزء الأول من هذه الدورة المستمرة للبرمجة بلغة بايثون والتي ستتضمن ان شاء الله في الجزء الثاني كيفية التعامل مع السلاسل الرمزية (strings) والملفات (files) والقوائم (lists) وصولاً الى البرمجة الموزعة عبر الشبكات.

هذه الدروس وغيرها الكثير تم نشرها في مدونة مصطفى صادق العلمية التي يمكن زيارتها

للاطلاع على المزيد على الرابط التالي:

[/https://mustafasadiq0.wordpress.com](https://mustafasadiq0.wordpress.com)

تفضلوا بزيارتنا للاطلاع على مئات الدروس والشروحات والمحاضرات الاكاديمية والمقالات الثقافية وفي كل المجالات التي

تهم طالب العلم في مجال الحاسوب واللغة الإنكليزية وغيرها من المجالات ☺

انتظروا الجزء الثاني من هذه الدورة على موقع كتب قريباً ان شاء الله او تابعوا الدروس على المدونة اولاً بأول ☺