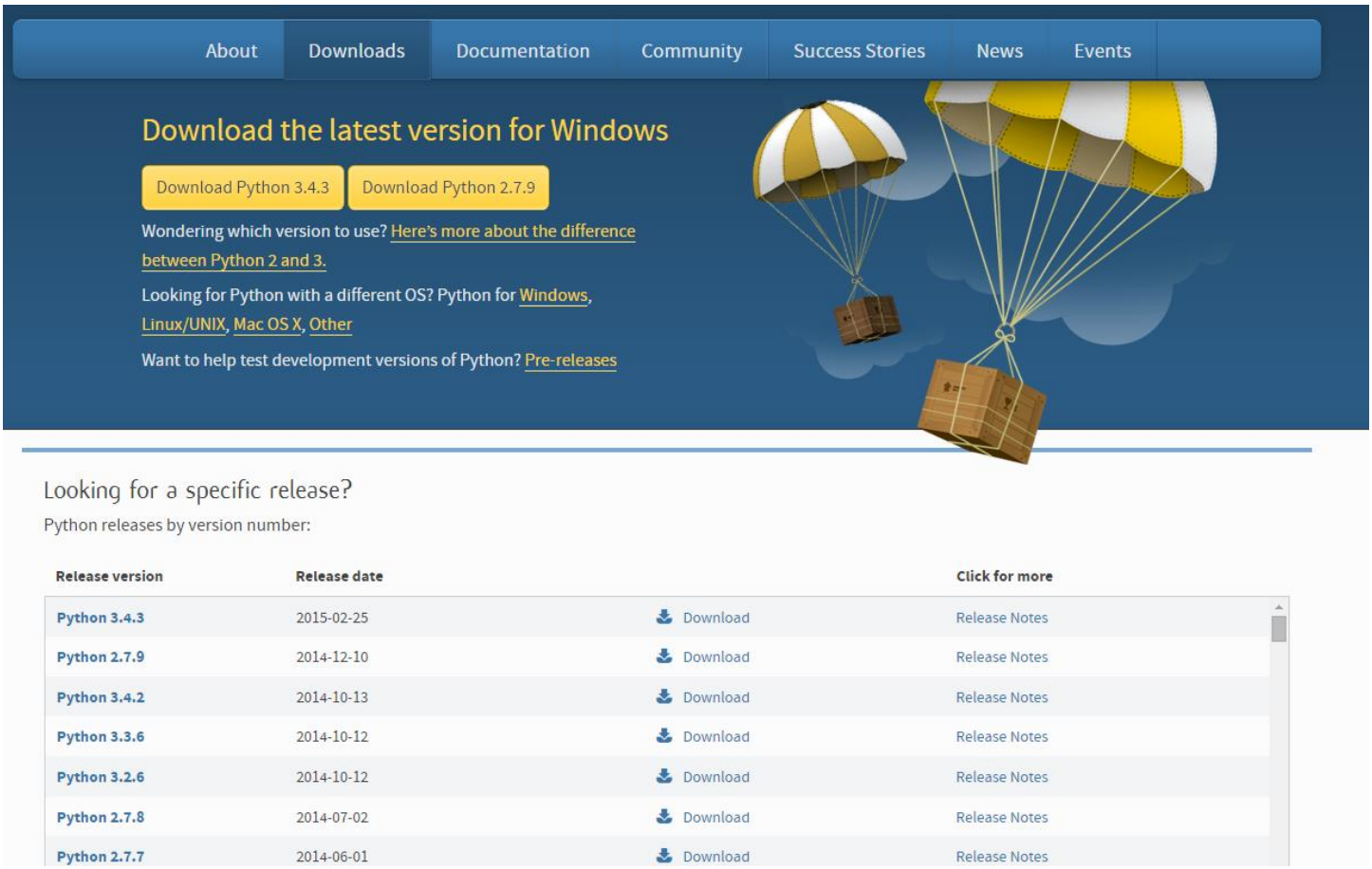
لغة البرمجة بايثون (Python Programming Language)

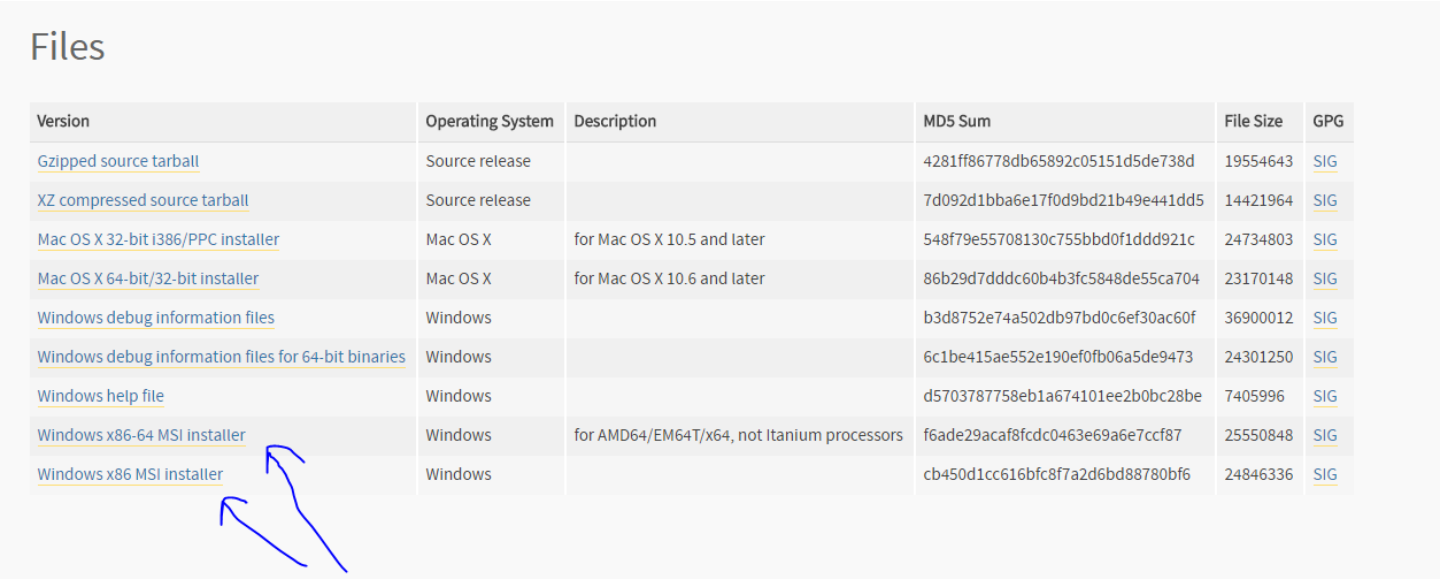
الدرس الأول: مقدمة عامة

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) لأدارة 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:

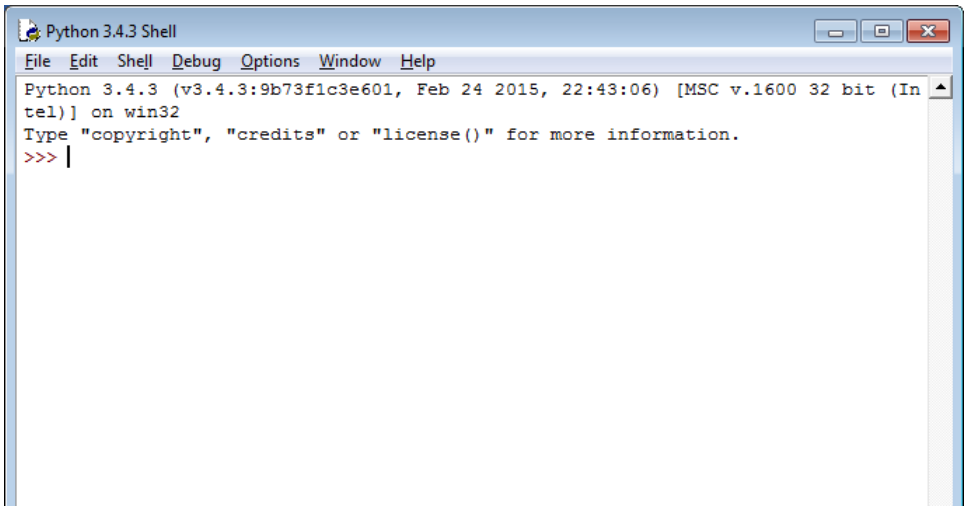


من هذه الواجهة نختار الإصدار المناسب (ويفضل ان يكون الأخير لاحتوائه على اخر التحديثات) وهو الان (وقت كتابة الدرس) الإصدار (python 3.4.3) فننقر عليه لتظهر الواجهة التالية:



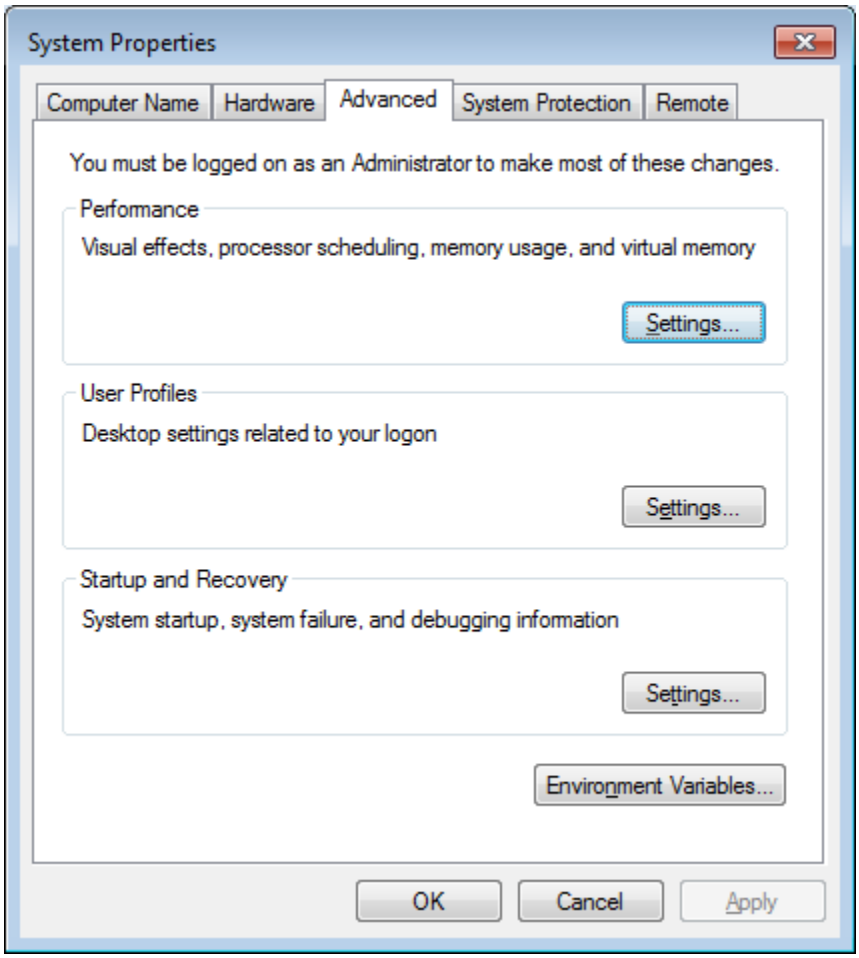
نجد هذه الملفات في أسفل صفحة ونقوم بتنزيل النسخة المناسبة لنظام تشغيلنا (32 or 64) ونقوم بتنصيبها وهي سهلة التنصيب لا تحتاج أي معلومات مسبقة لإكمال تنصيبها.

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

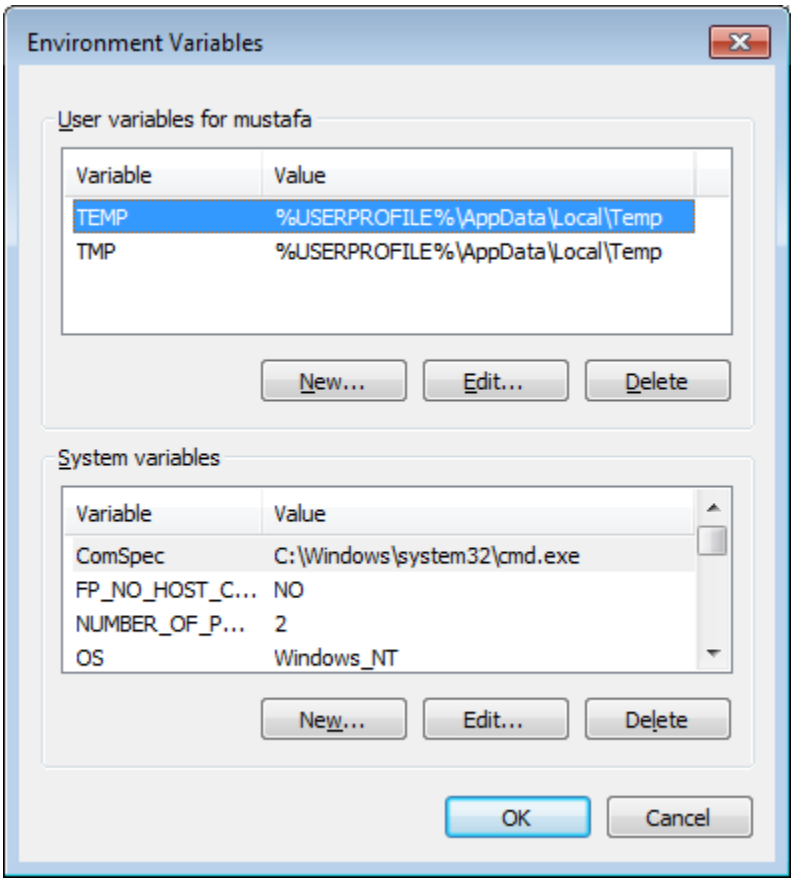


وهذه الواجهة تمثل الواجهة الرسومية لتطوير وتطبيق برامجنا التي سنعمل عليها في هذا الدرس والدروس القادمة ان شاء الله تعالى.

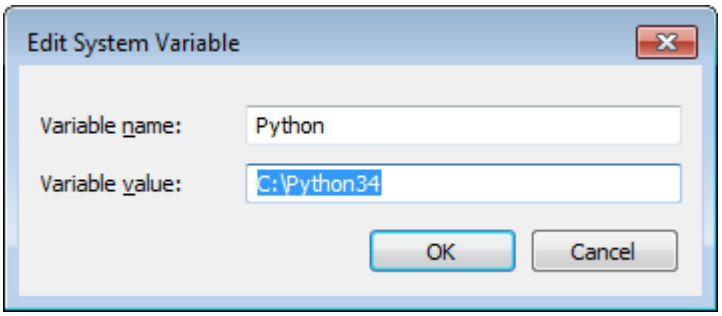
الان ننتقل الى الخطوة المهمة الأخرى وهي ضبط بيئة الويندوز للعمل على البايثون خارج ال (IDLE) الخاص به في سطر الأوامر (Command Prompt) او غيرها ويتم ذلك بالنقر نقرة يمين على ايقونة جهاز الكومبيوتر ثم الذهاب الى properties ثم (advanced system settings) لتظهر النافذة التالية:



الان ننقر على (environment variables) لتظهر النافذة التالية:



الان نذهب الى (system variables) وننقر على (new) لتظهر الواجهة التالية والتي نملأها بالمعلومات المبينة في ادناه:



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).

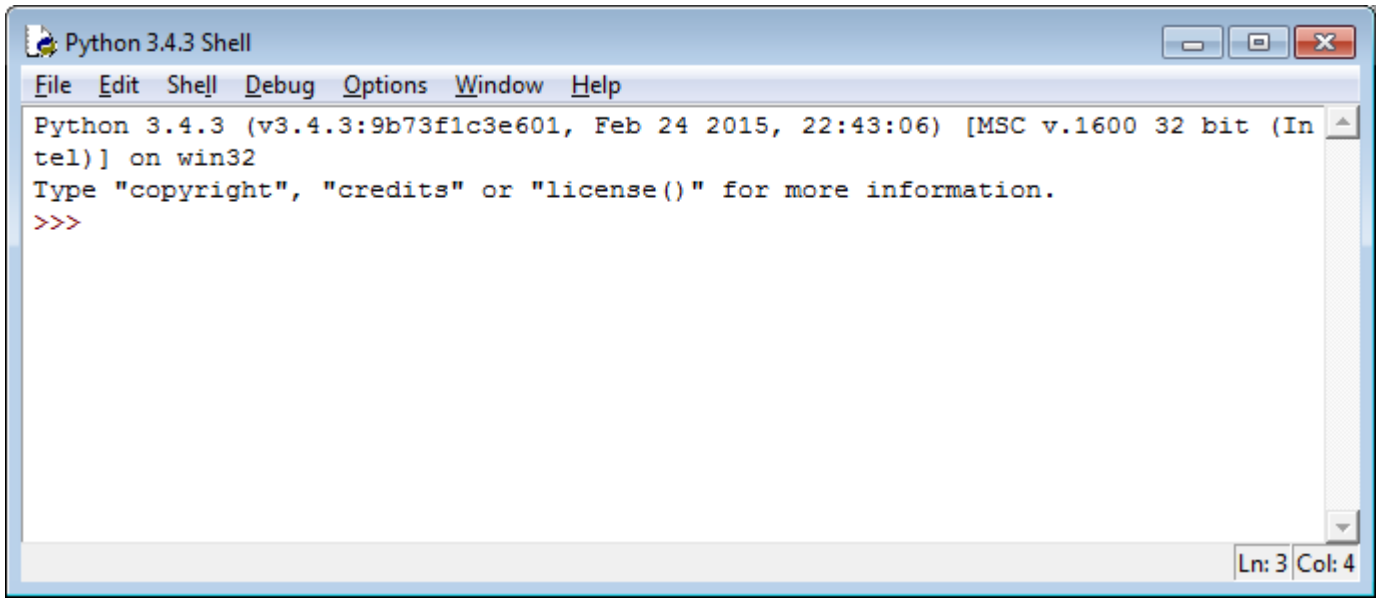
بعدها ننقر على (ok) وبذلك تنتهي عملية تهيئة بيئة الويندوز للتعامل مع البايثون في التطبيقات المختلفة.

الدرس الثاني برنامج (hello world):

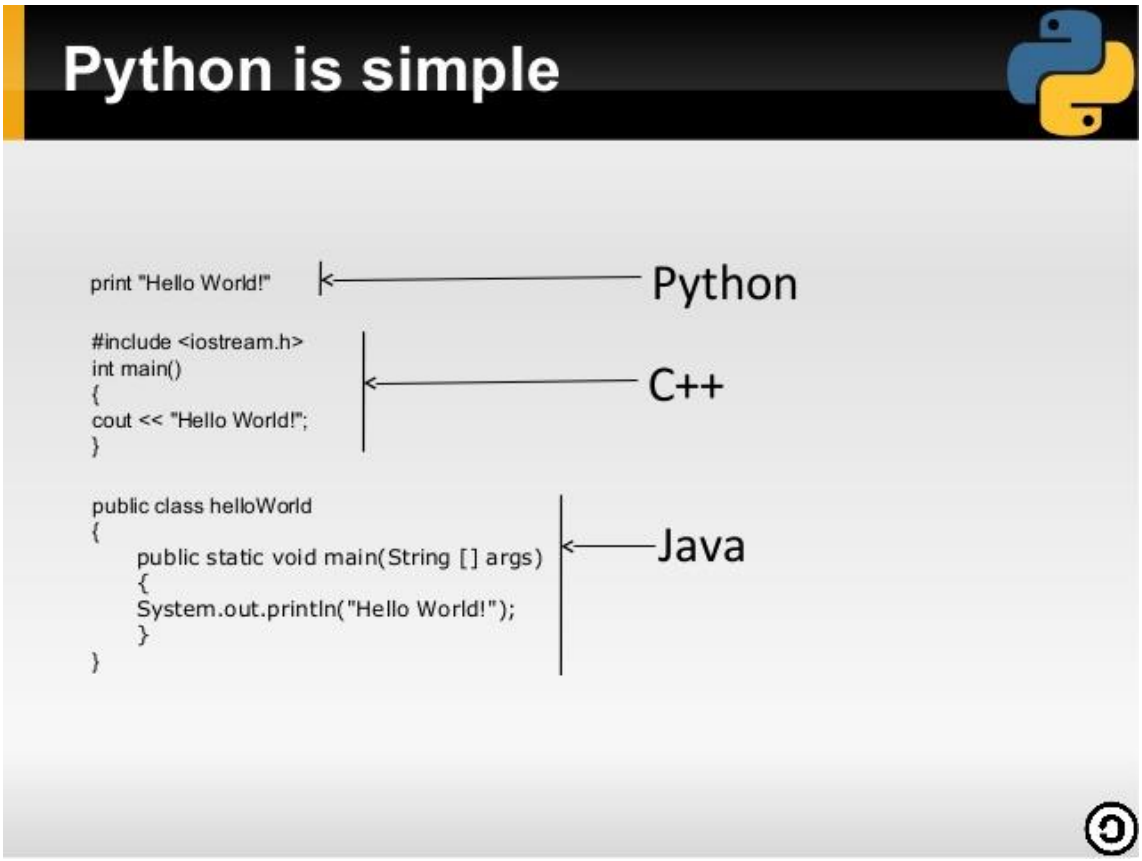
بعد ان شرحنا كيفية تنزيل وتنصيب البرمجيات اللازمة للبدء بالعمل على لغة بايثون في الدرس السابق، وشرحنا ايضاً كيفية اعداد بيئة نظام تشغيل الويندوز للتعامل مع هذه اللغة. نأتي اليوم الى البدء بالتعامل مع هذه اللغة وكيفية كتابة اول برنامج باستخدامها ولكن قبل ذلك لا بد من ذكر ملاحظة مهمة جداً وهي اننا اعتمدنا في شرحنا لهذه الدروس على تطبيق (IDLE Python GUI) ولكن هناك الكثير من التطبيقات الأخرى التي يمكن استخدامها لكتابة برامج بايثون واختبار أدائها 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 احترفوا 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:



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:

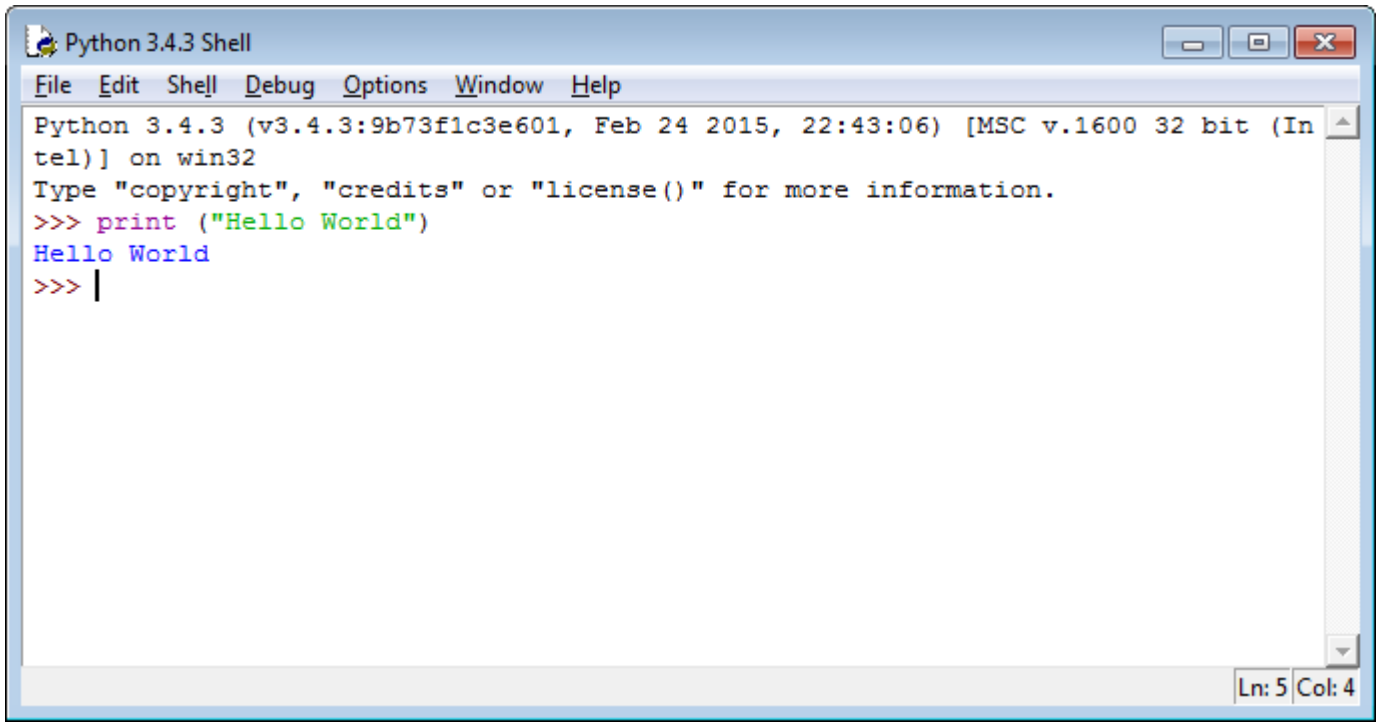


ملاحظة: الصورة أعلاه تحتوي خطأ حيث تم استخدام (print) للغة بايثون بدون اقواس وهو الحال للبايثون ٢،٧ وما قبلها واما بايثون ٣ وما بعدها فتتطلب الاقواس بشك ضروري لذا يجب الانتباه الى النسخة التي تستخدمها وفروقات متطلباتها.

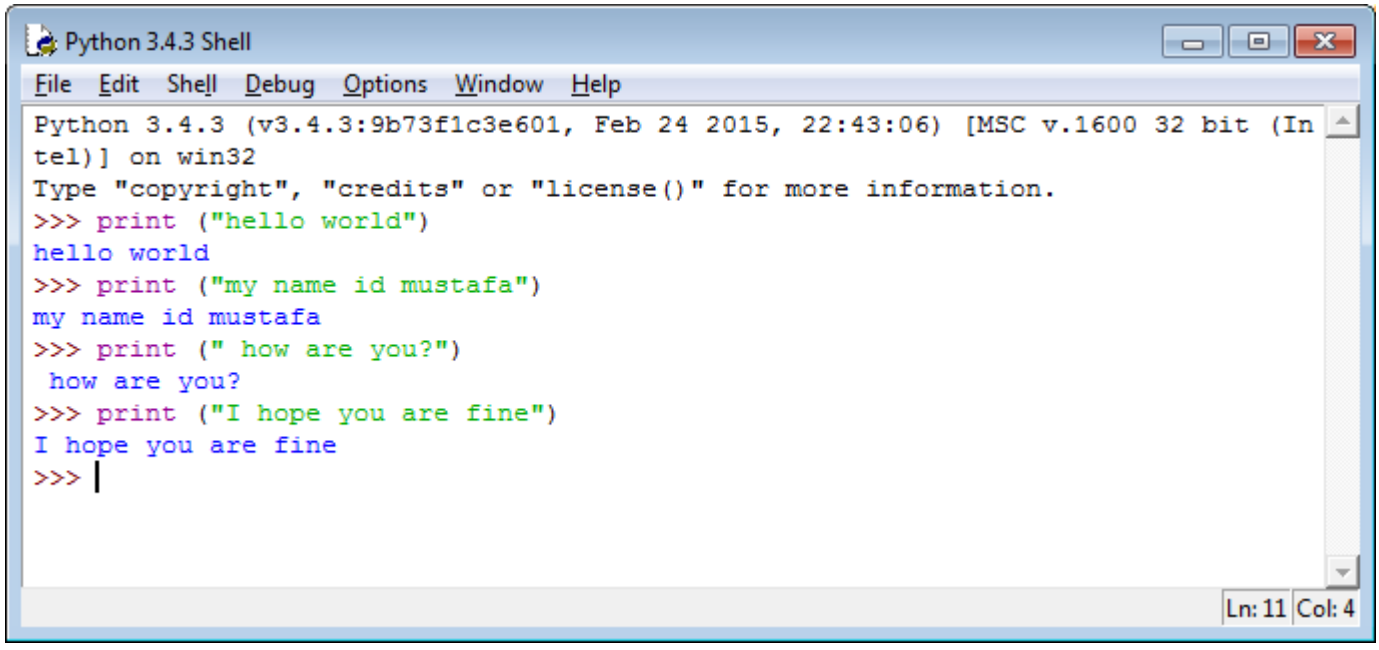
وكما لاحظنا هنا فأن طباعة عبارة (Hello world) تتطلب سطراً واحداً فقط وهو

print (“hello world”)

والذي يتضح في مفسر اللغة كما يلي:



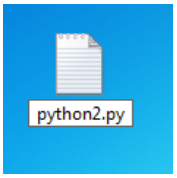
الان عرفنا كيفية طباعة سطر واحد من الكلمات ويمكن طبعاً طباعة أكثر من سطر وذلك بتكرار عبارة (Print) وكما في ادناه:



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:

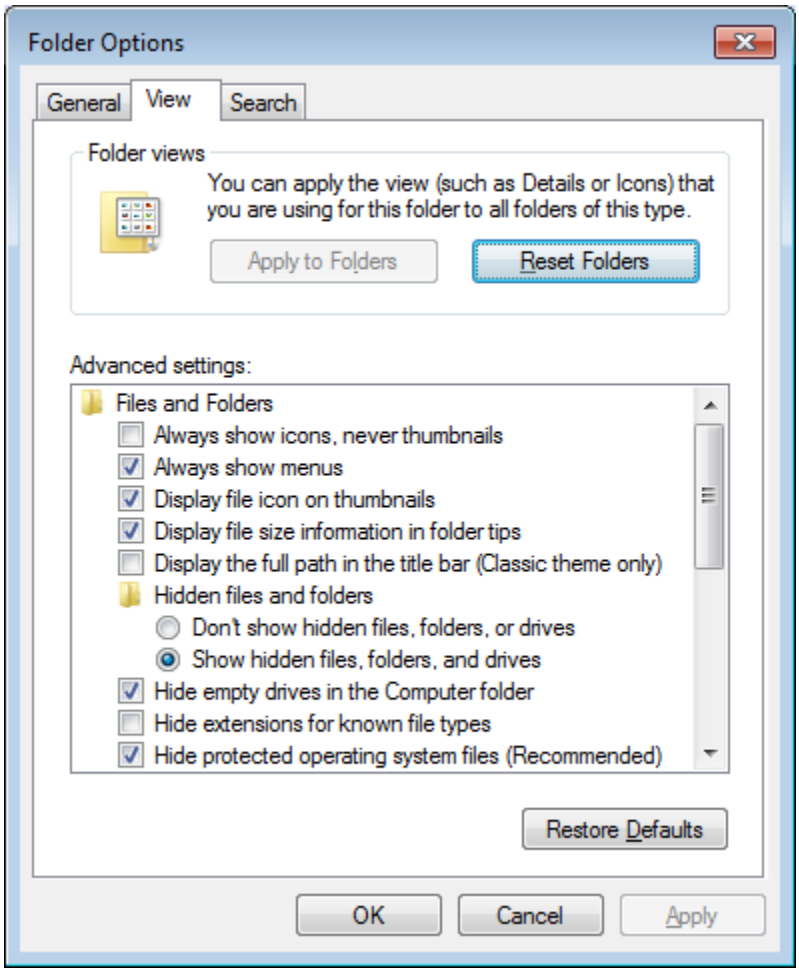
نبدأ خطوات ذلك الان:

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:

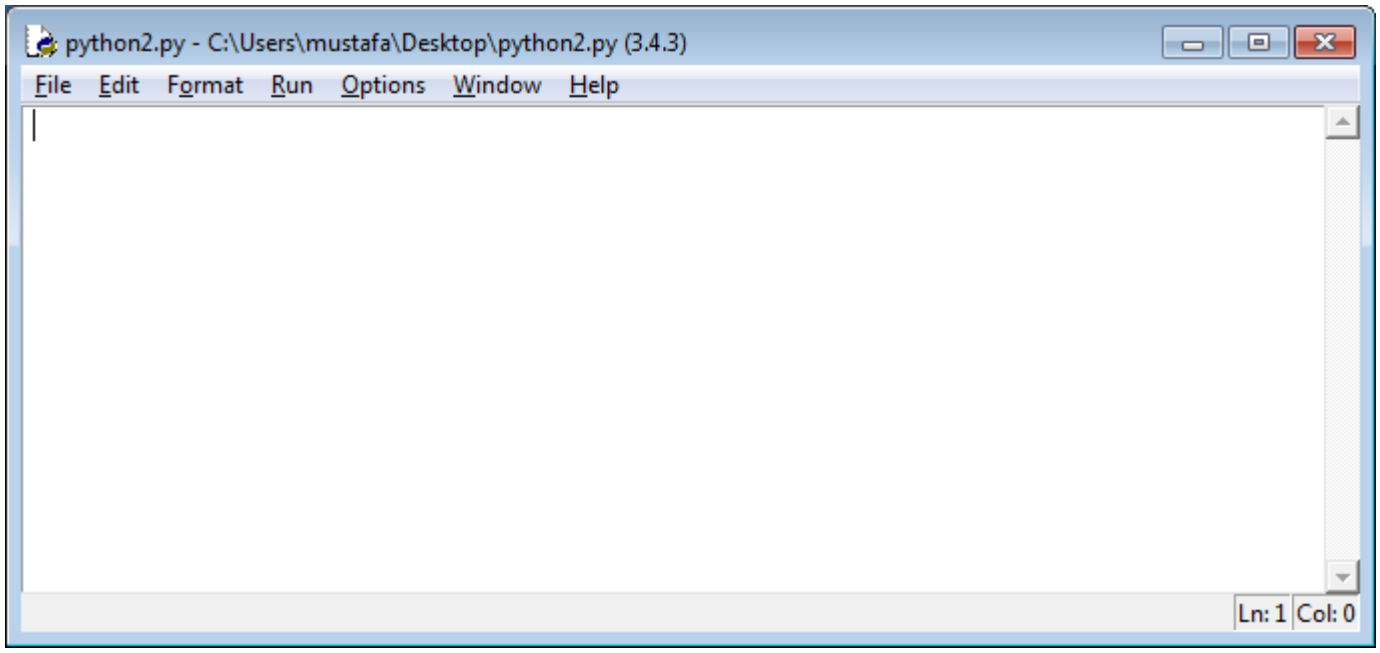
نفتح (my computer) ثم نذهب الى قائمة (tools) ومنها نختار (folder options) ثم (view) لتظهر النافذة التالية:



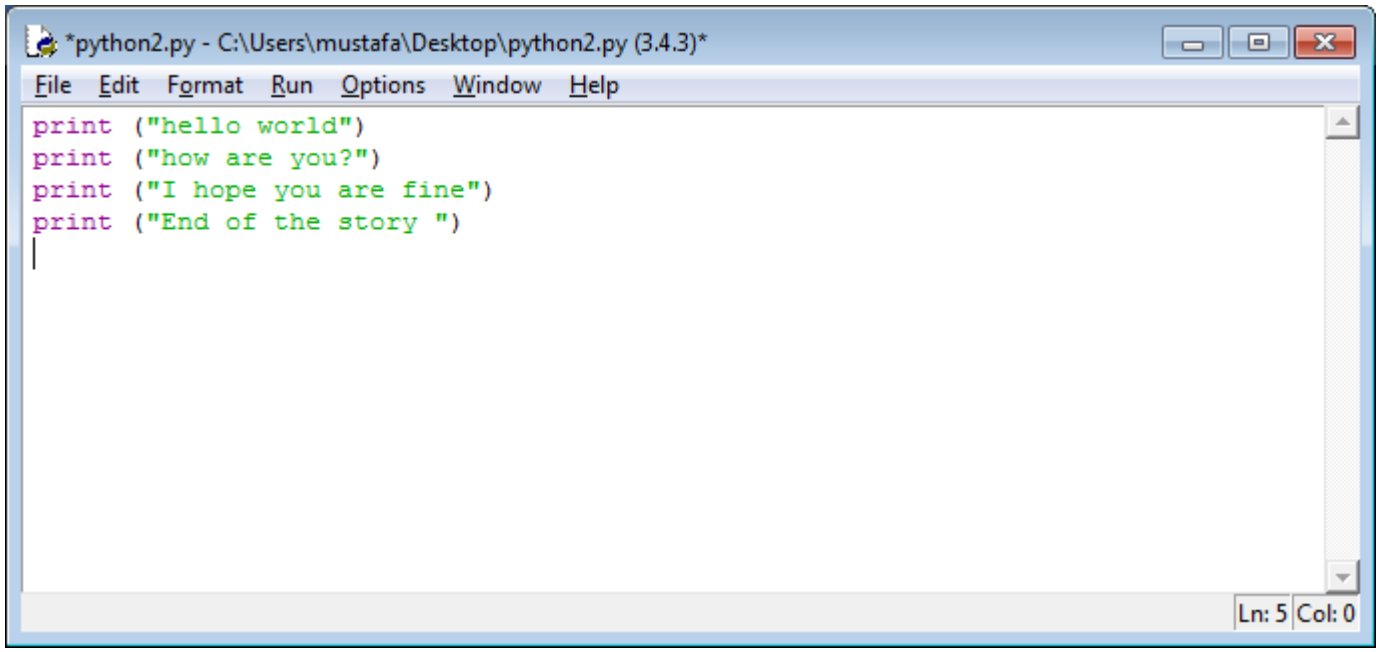
الان نقوم بإزالة علامة الصح (ان كانت موجودة) من المربع امام خيار (Hide extensions for known file types) ثم (ok) وبعدها سنلاحظ ان كل الملفات في الحاسوب ستظهر مع امتداداتها مما يسمح لنا بتغيير امتداد أي ملف بحسب

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

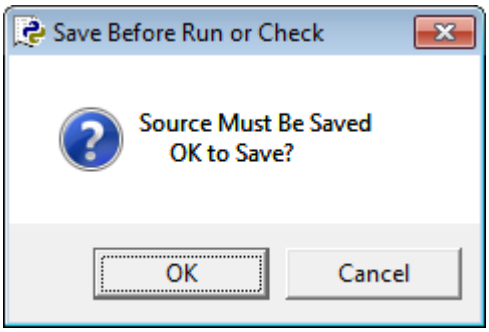
الان نقوم بالنقر نقرة يمين على هذا الملف واختيار (edit with idle) لتفتح النافذة التالية:



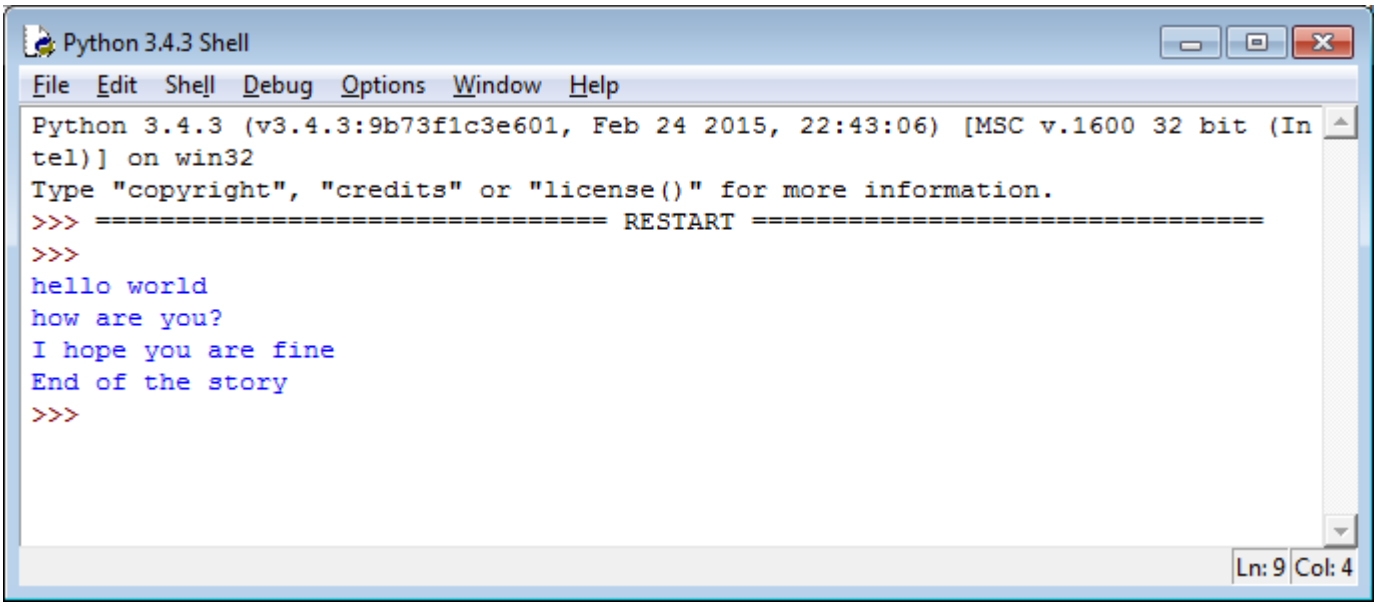
الان نكتب أي عدد من أوامر البايثون هنا ولأننا لحد الان لا نعرف سوى ايعاز الطباعة (print) فسنكتب التالي:



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:



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 الادخال وهي أداة (input) والتي يمكن استخدامها بالطريقة التالية:

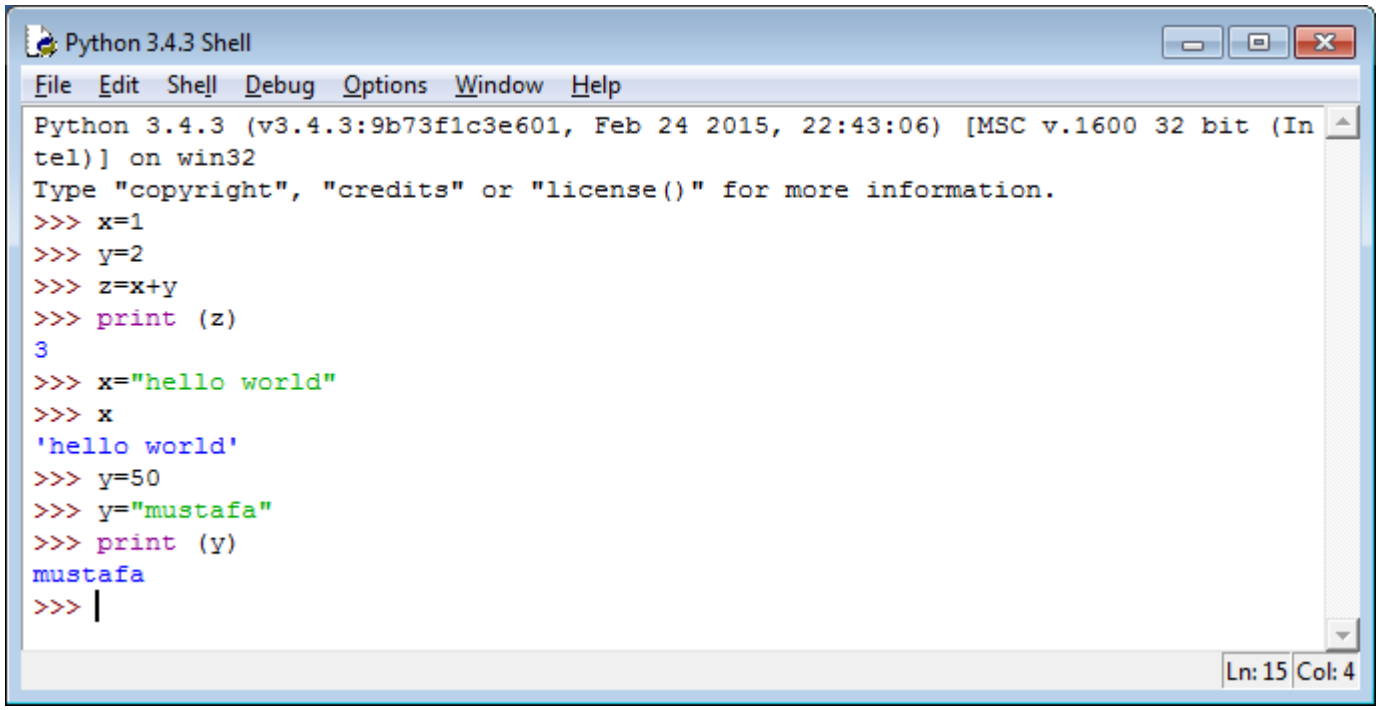
x= input ("enter your name") للبايثون ٣ وما بعدها

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

وكما تلاحظون فأن الادخال 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 تركيبة 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:



من هنا نلاحظ الأمور التالية:

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

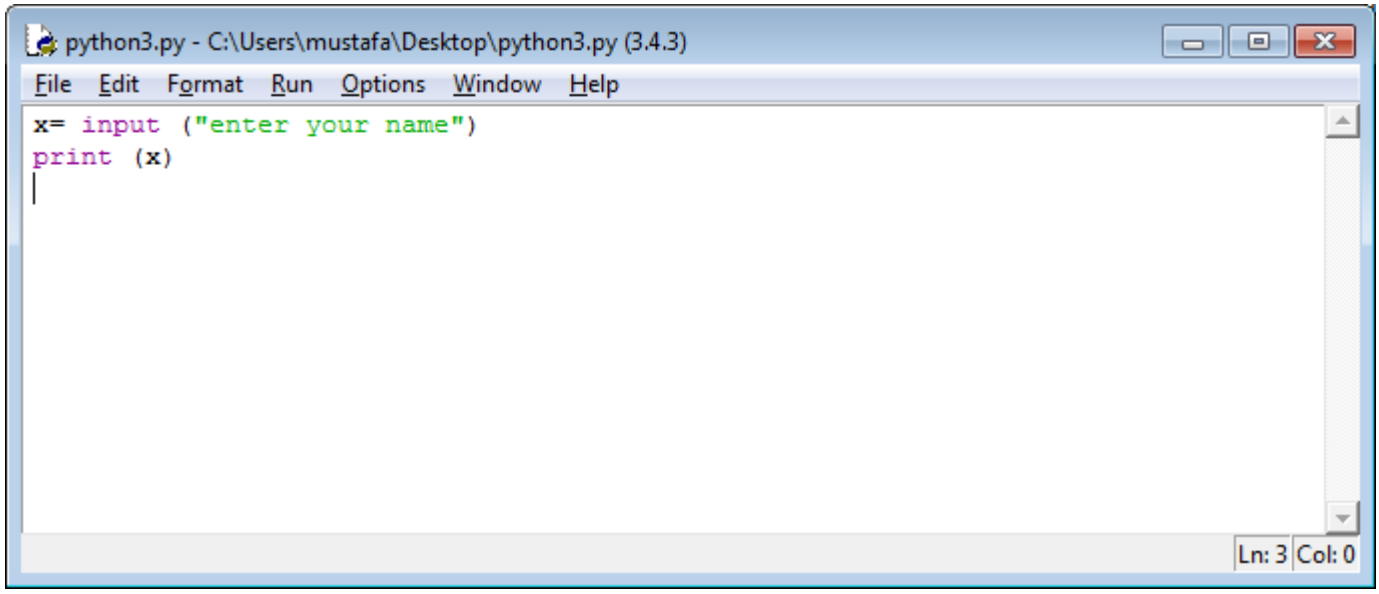
٢- إمكانية اجراء العمليات الحسابية مباشرة وبدون مقدمات ولا استدعاء لمكتبات الرياضيات او غيرها كما في بقية لغات البرمجة الأخرى.

٣- طباعة قيمة أي متغير باستخدام ايعاز (print) في أي مكان من البرنامج.

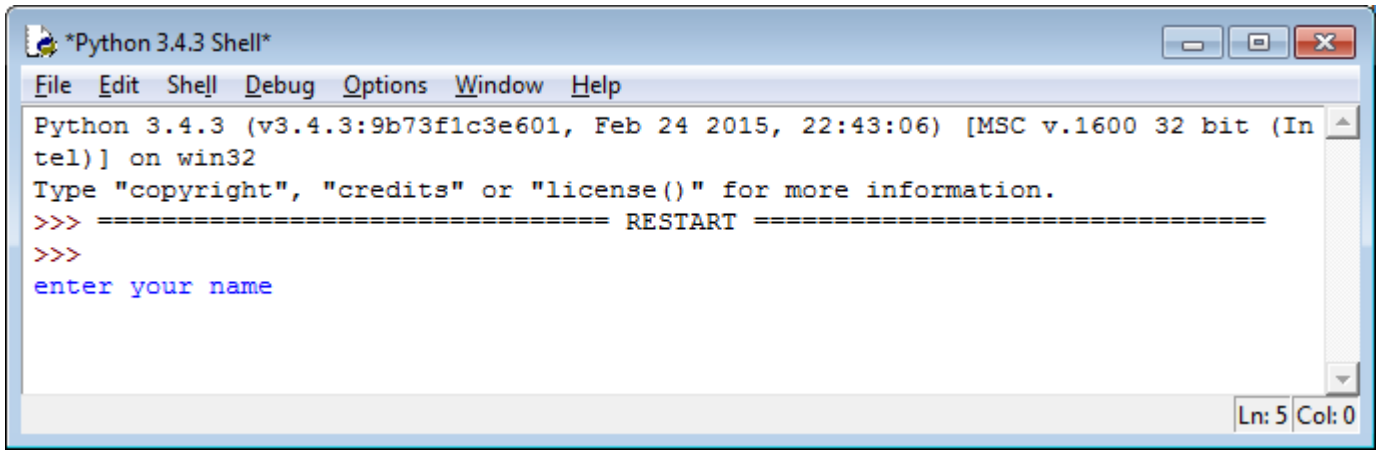
٤- يمكن اسناد قيم جديدة للمتغير والذي يأخذ دائماً اخر قيمة أسندت له وينسى القيم القديمة فمثلاً ال (x) كانت قيمته الأولية ١ ثم وضعنا بداخله (أسندنا له) قيمة رمزية هي ("hello world") وحين طلبنا طباعة قيمته بالإيعاز print(x) قام المفسر بطباعة قيمة المتغير النهائية.

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.

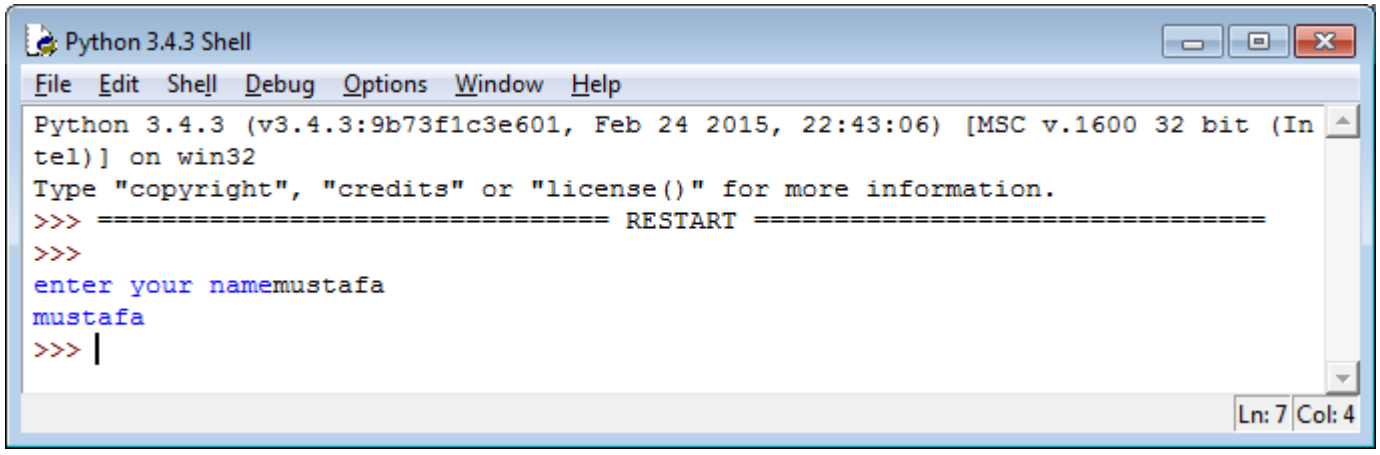
والان نعود الى أداة الادخال موضوع الدرس ونكتب الكود التالي في ملف (.py) كما فعلنا في الدروس السابقة وكما في ادناه:



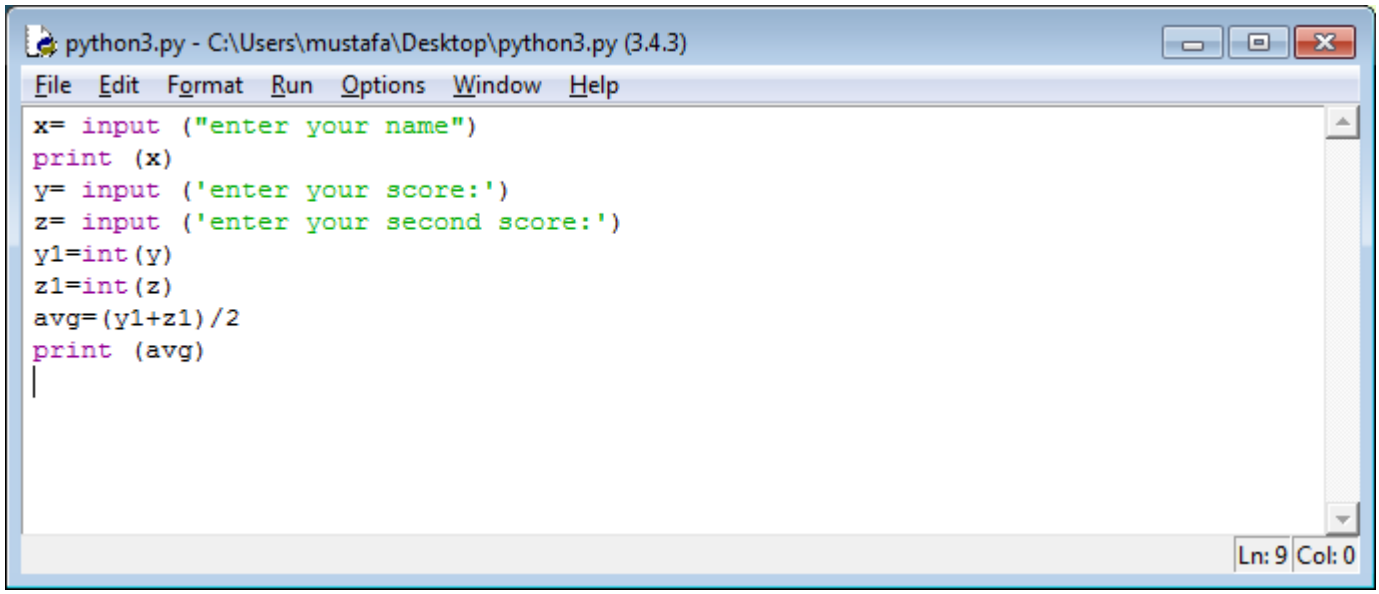
الان نقوم بتنفيذ هذا الكود لنرى النتيجة:



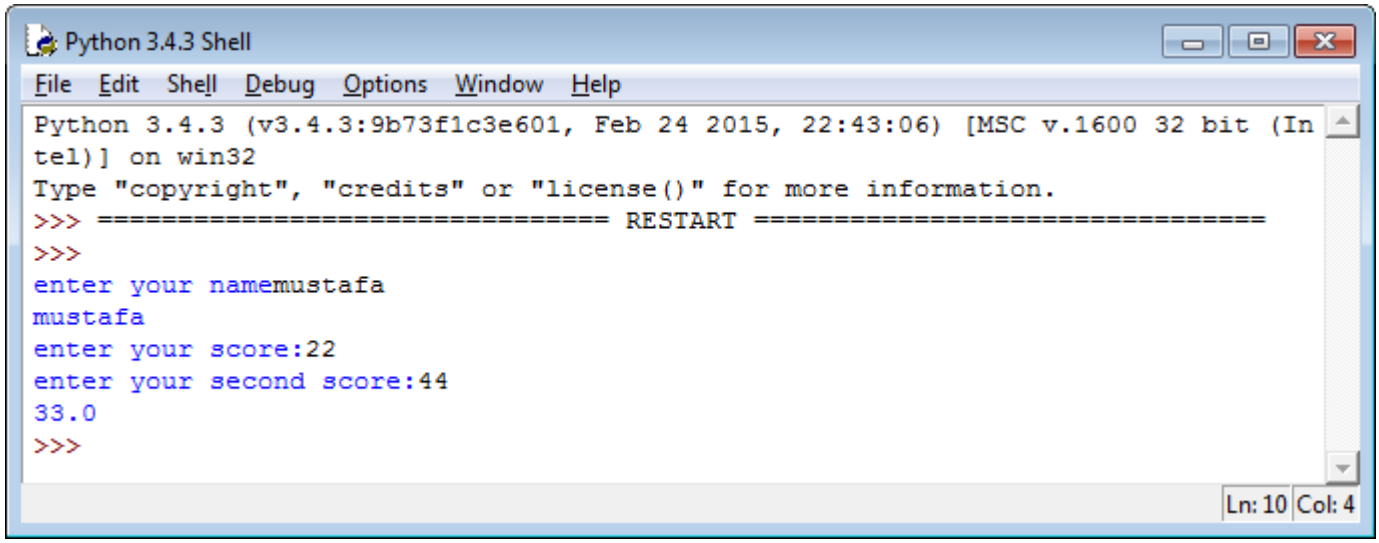
هنا نرى ان المفسر قام بطباعة العبارة داخل ايعاز الادخال (input) منتظراً منا ادخال قيمة معينة ليقوم بأسنادها الى المتغير (x) فنقوم بكتابة أي شيء وننقر (enter) لنرى النتيجة:



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



والان عند تنفيذ هذا البرنامج نجد النتائج التالية:



ولتوضيح البرنامج أعلاه نقول:

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

الايعاز الثاني لطباعة قيمة المتغير x

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

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

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

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

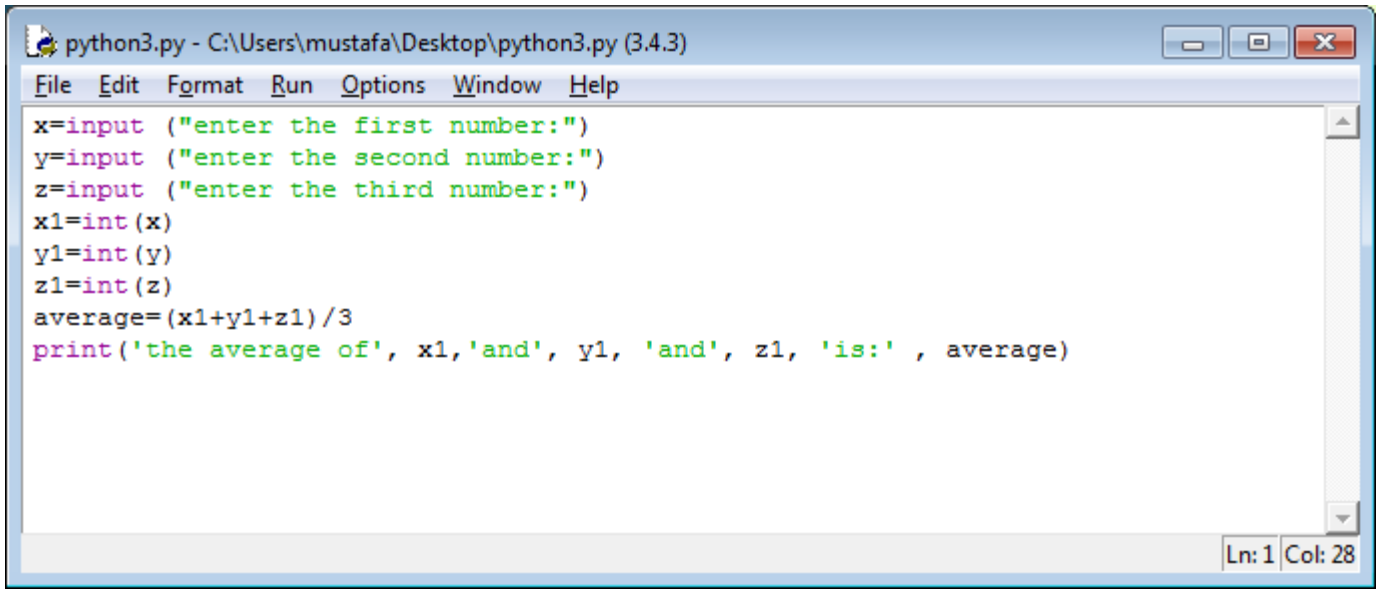
الايعاز السابع هو عملية رياضية لحساب معدل القيمتين التي تم ادخالهما ووضع الناتج في متغير يسمى avg

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

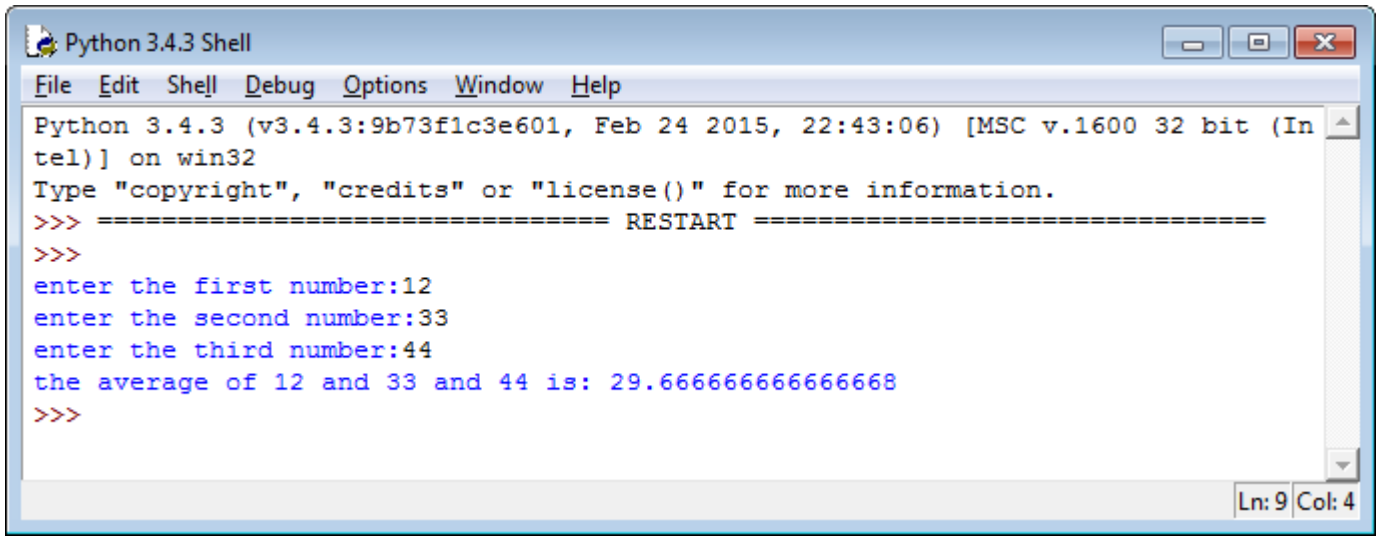
ملاحظة: للخروج من برنامج (IDLE Python GUI) يمكن فقط كتابة exit () في سطر الأوامر او النقر (Ctrl + C) سوية من لوحة المفاتيح. اما محرر النصوص الذي قمنا بالكتابة بداخلة من البداية فيجب الحرص على خزن التغييرات فيه قبل التنفيذ وقبل اغلاقه تجنباً لضياع الشفرات البرمجية التي قمنا بكتابتها.

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:



وعند التنفيذ تظهر النتائج التالية:



وايضاً لتوضيح البرنامج نذكر التالي:

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

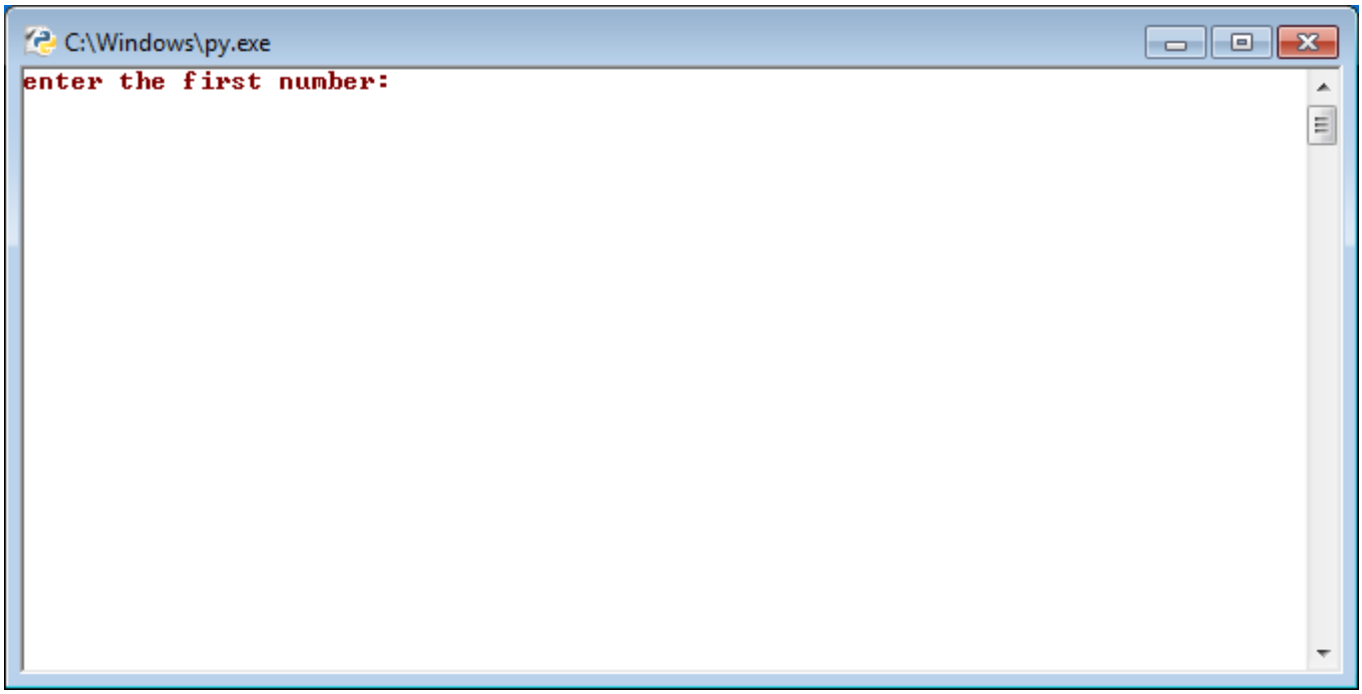
الاسطر الرابع والخامس والسادس هي لتحويل قيم تلك المتغيرات من سلاسل رمزية (string) الى قيم صحيحة (int).

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 بخزنة في البداية تحت اسم (*.py) لنرى النتيجة التالية:

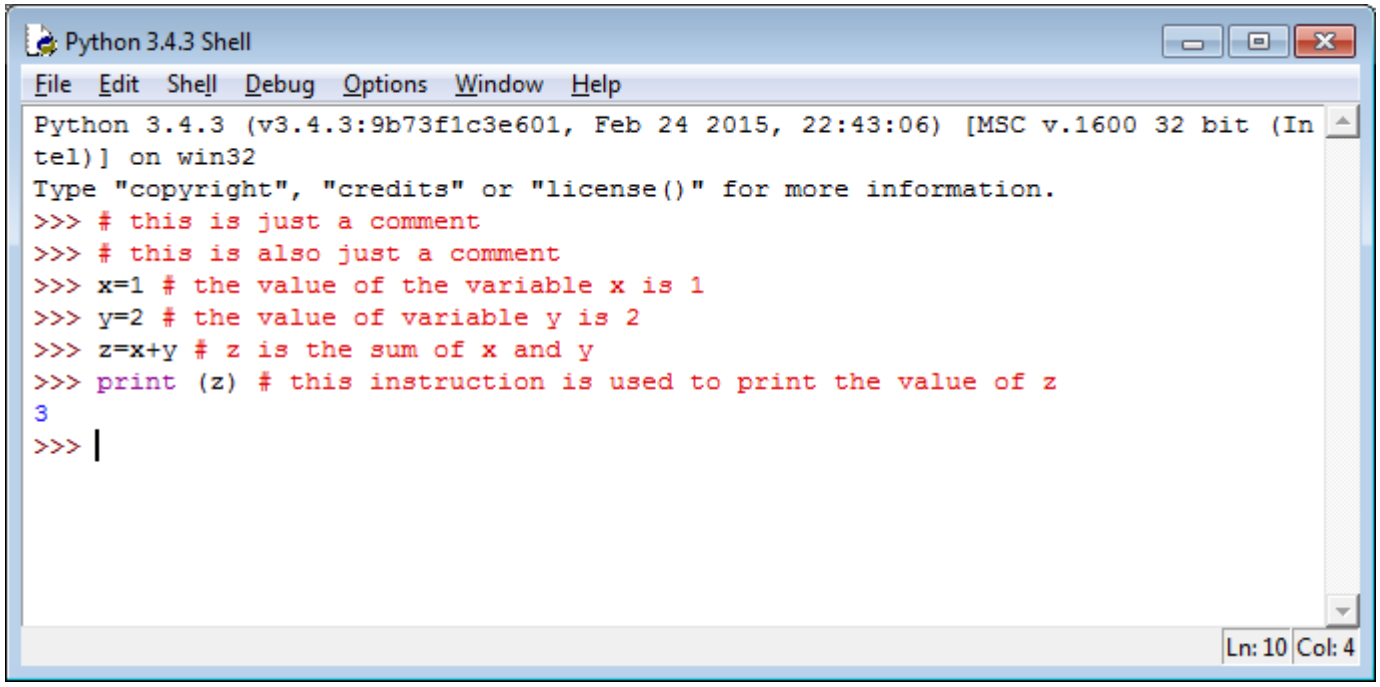


مما يعني ان برنامجنا الذي قمنا بكتابته وحفظه قد أصبح ملف قابل للتنفيذ بعيداً عن اللغة ومفسرها أي اننا نستطيع وضع أي شفرة برمجية بداخله وحفظه ليتم تنفيذه بشكل مباشر فيما بعد وهو أحد المميزات الممتازة للغة البايثون مقارنة ببقية لغات البرمجة الأخرى التي يتطلب انشاء ملف تنفيذي فيها خطوات كثيرة.

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

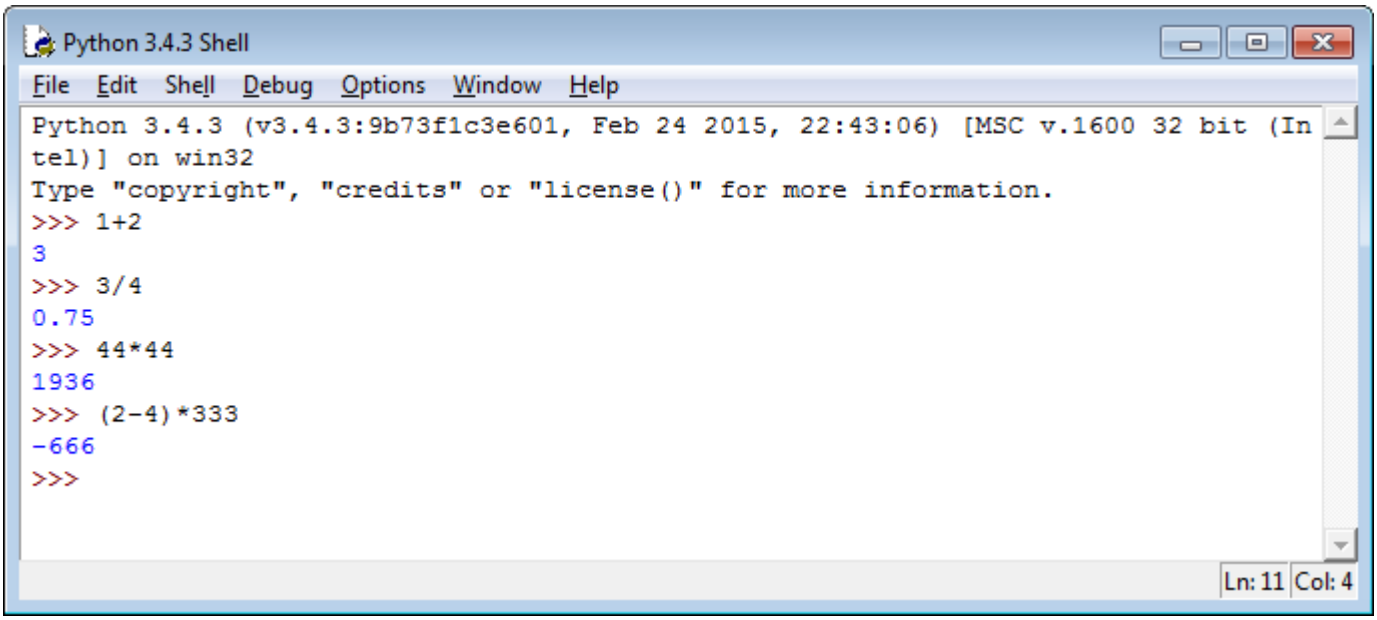


هنا رأينا كيف ان الكلمات التي بعد العلامة (#) لم يتم اخذها بعين الاعتبار عند التنفيذ فالمفسر لا ينظر اليها اصلاً وهي للمبرمج فقط وليس للمستخدمين وهي ذات أهمية كبيرة للمبرمجين خصوصاً للبرامج الكبيرة والمتشعبة.

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

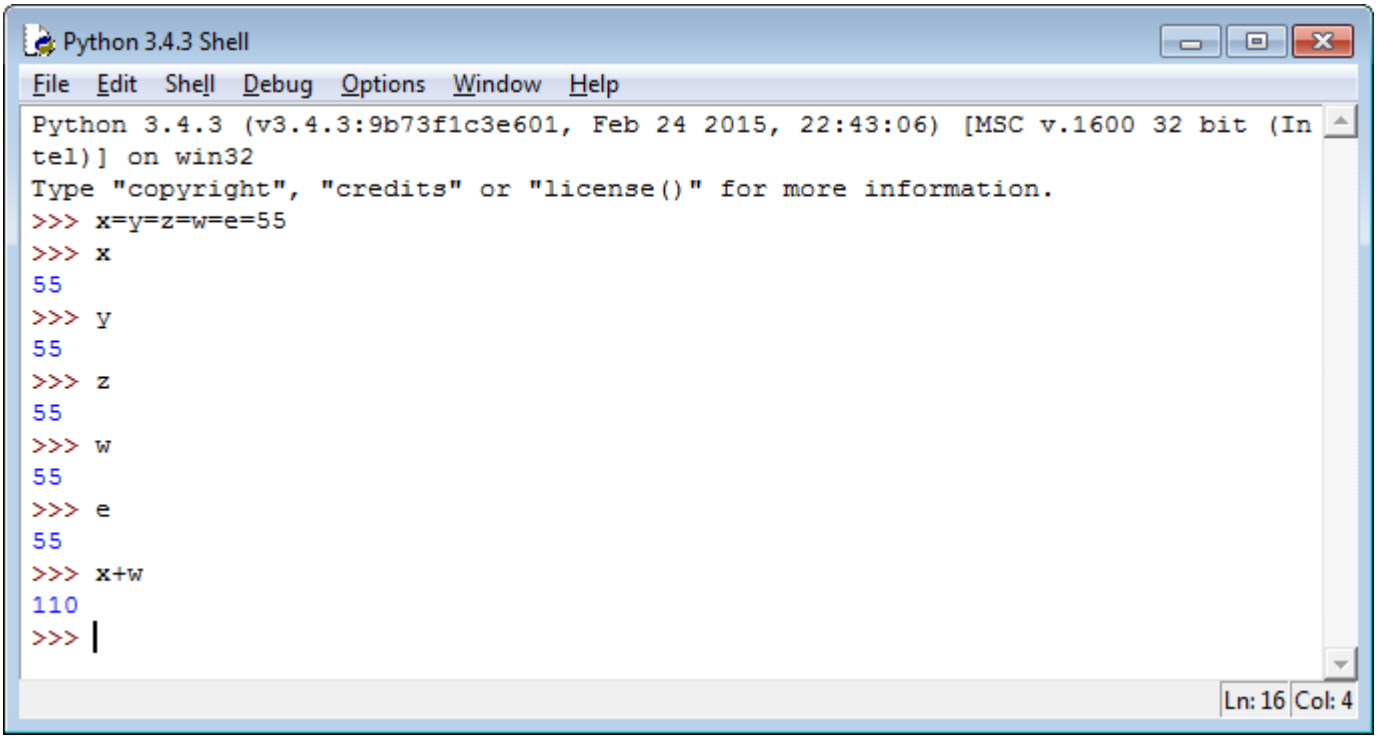
- استخدام البايثون كألة حاسبة:

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



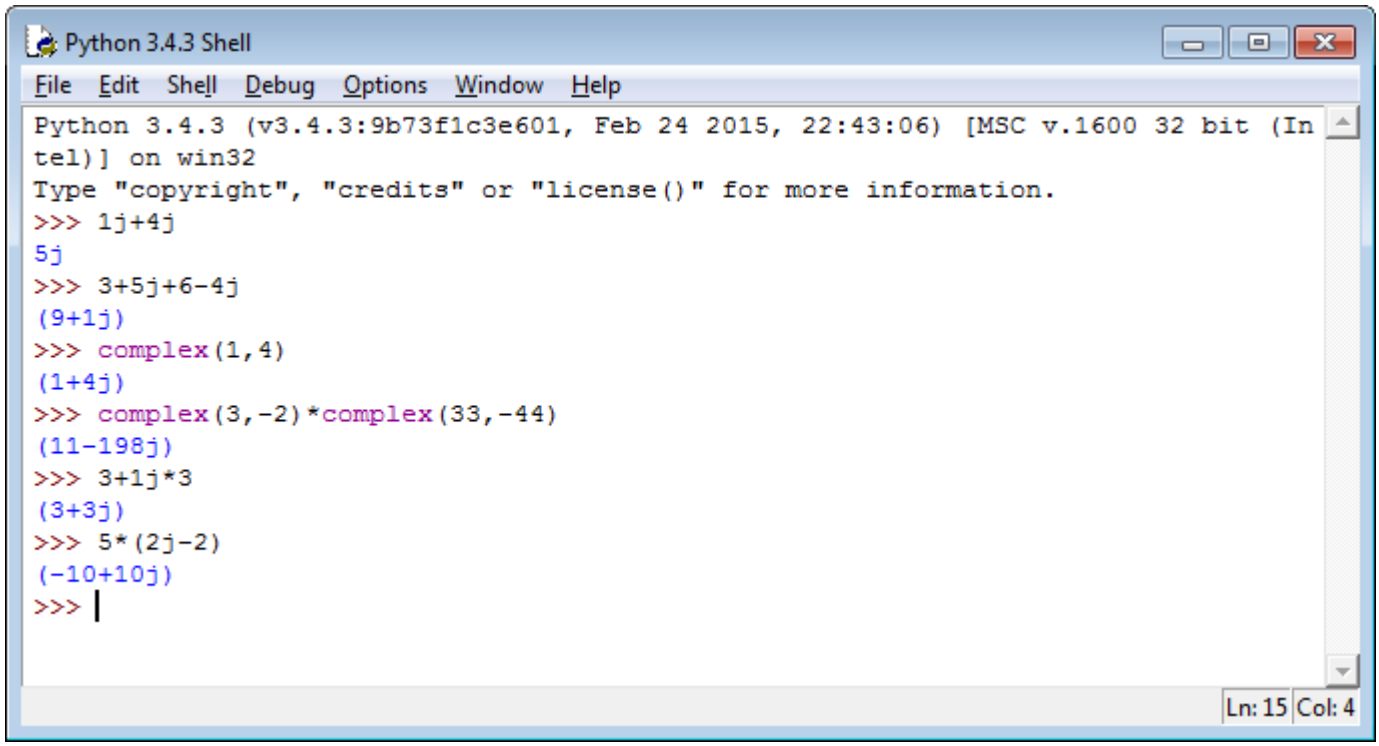
كما نرى فقد تم تنفيذ كل الايعازات بمجرد النقر على (enter) بعرض نتائج العمليات الرياضية مباشرة مع عدم اهمال أي كسور عشرية.

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



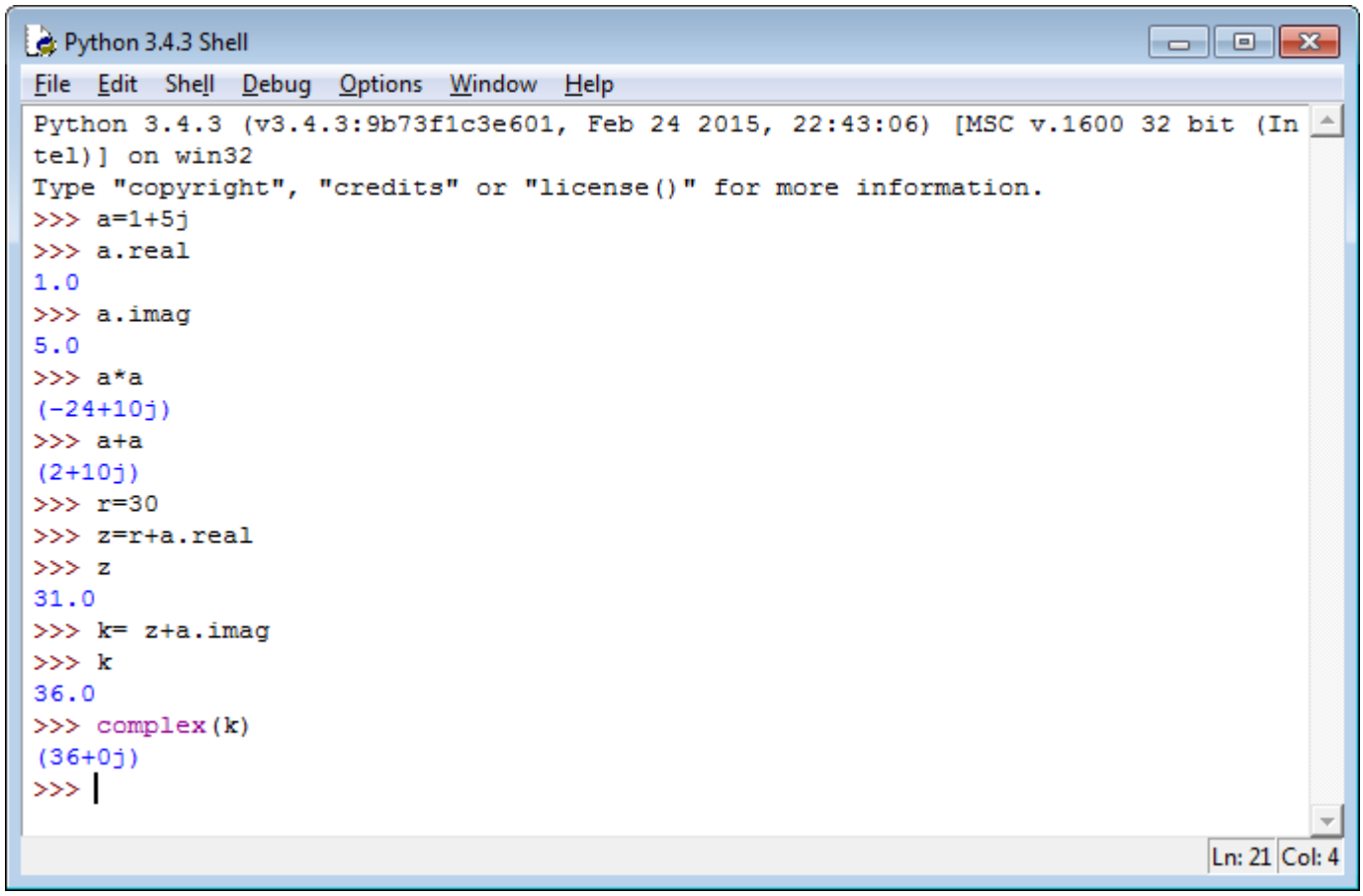
ويعلم مبرمجي لغات الجافا والسي بلس بلس ان هذا غير مقبول في تلك اللغات مما يميز لغة بايثون عنها في هذا المجال.

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:



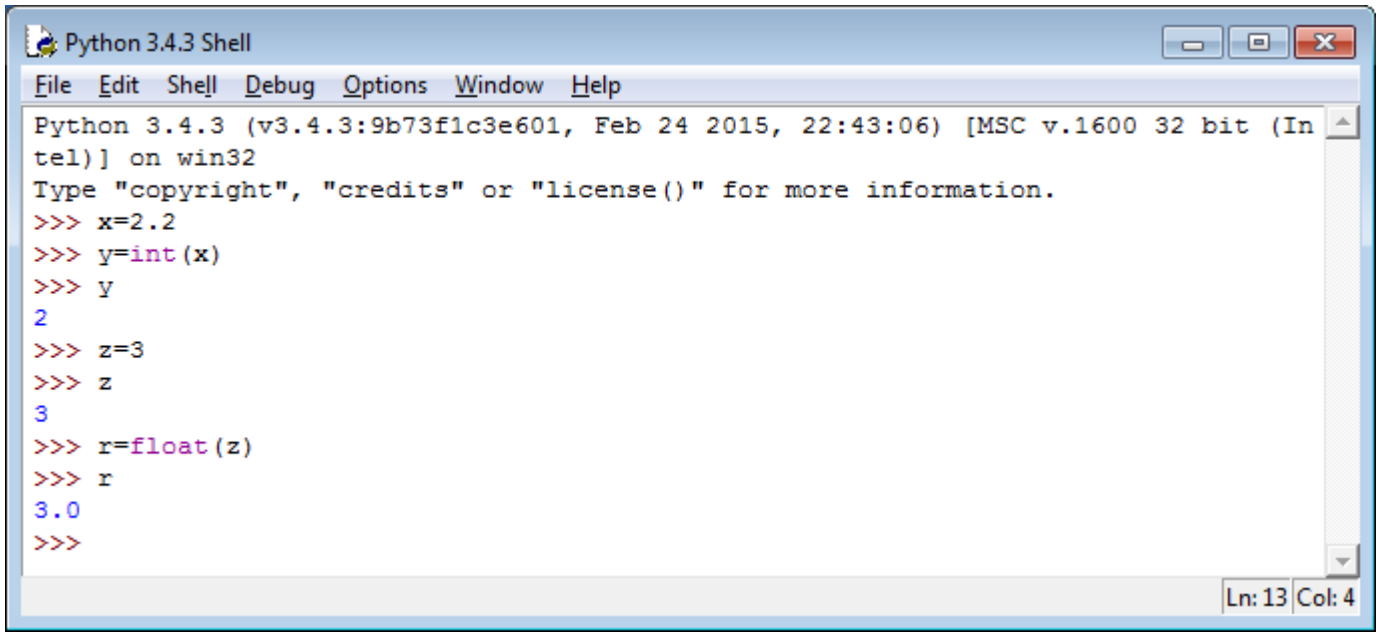
ومن الأمثلة أعلاه نلاحظ اننا نستطيع التعبير عن الاعداد المركبة بعدة طرق واجراء كل العمليات الرياضية عليها بسهولة.

كذلك يمكن التعامل مع أجزاء الاعداد المركبة الحقيقي (Real) والتخيلي (imaginary) وكما في ادناه:

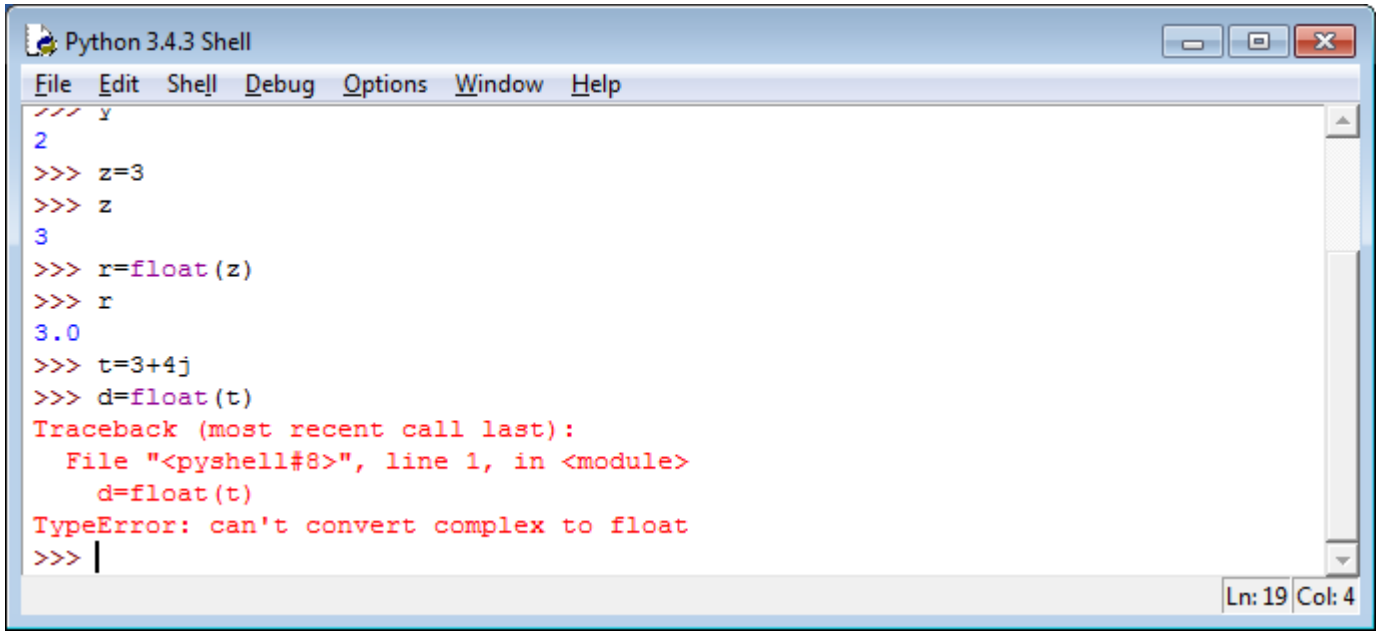


ومن هنا نرى اننا نستطيع تسميه العدد التخيلي باسم معين ثم التعامل مع جزئه الحقيقي والتخيلي كل على حدة.

وكما عرفنا من الدروس السابقة فهناك عدة دوال للتحويل بين الأنواع المختلفة للأعداد ومنها (int(), float()) وكما في ادناه:



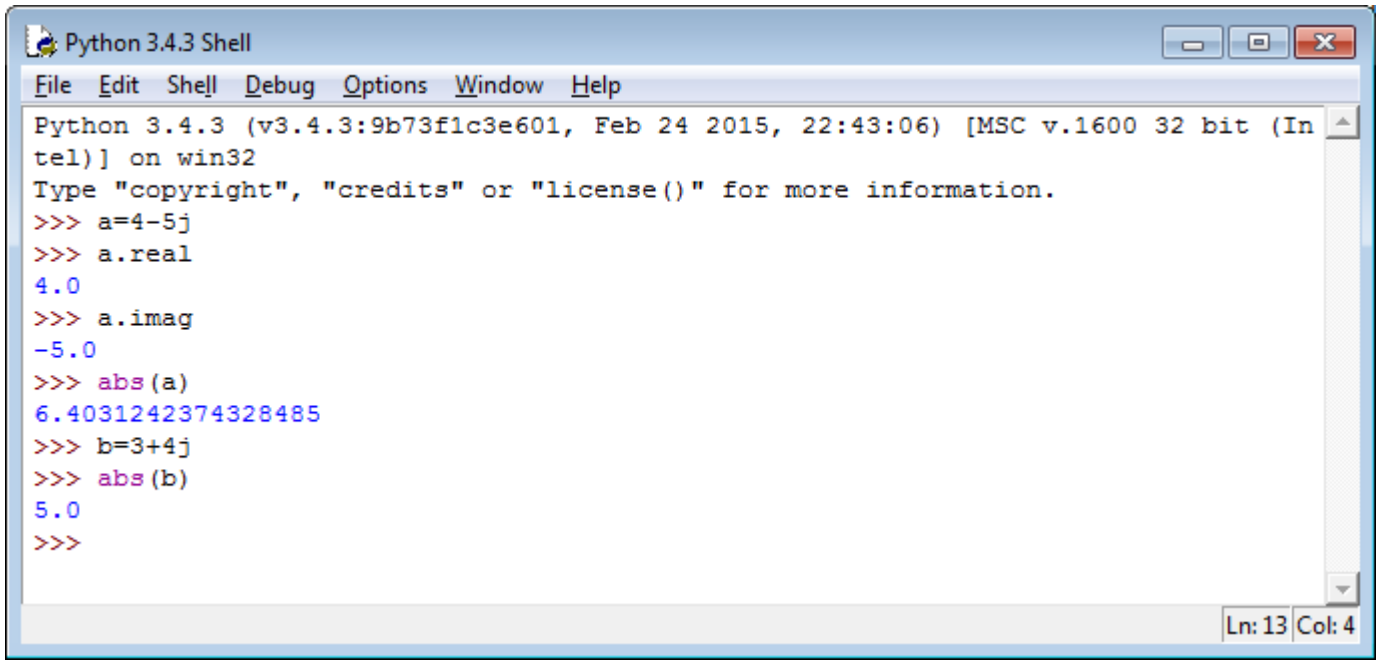
من هنا نرى اننا نستطيع تحويل الاعداد الصحيحة الى عشرية وبالعكس الا ان هذا لا يمكن مع الاعداد المركبة وكما في ادناه:



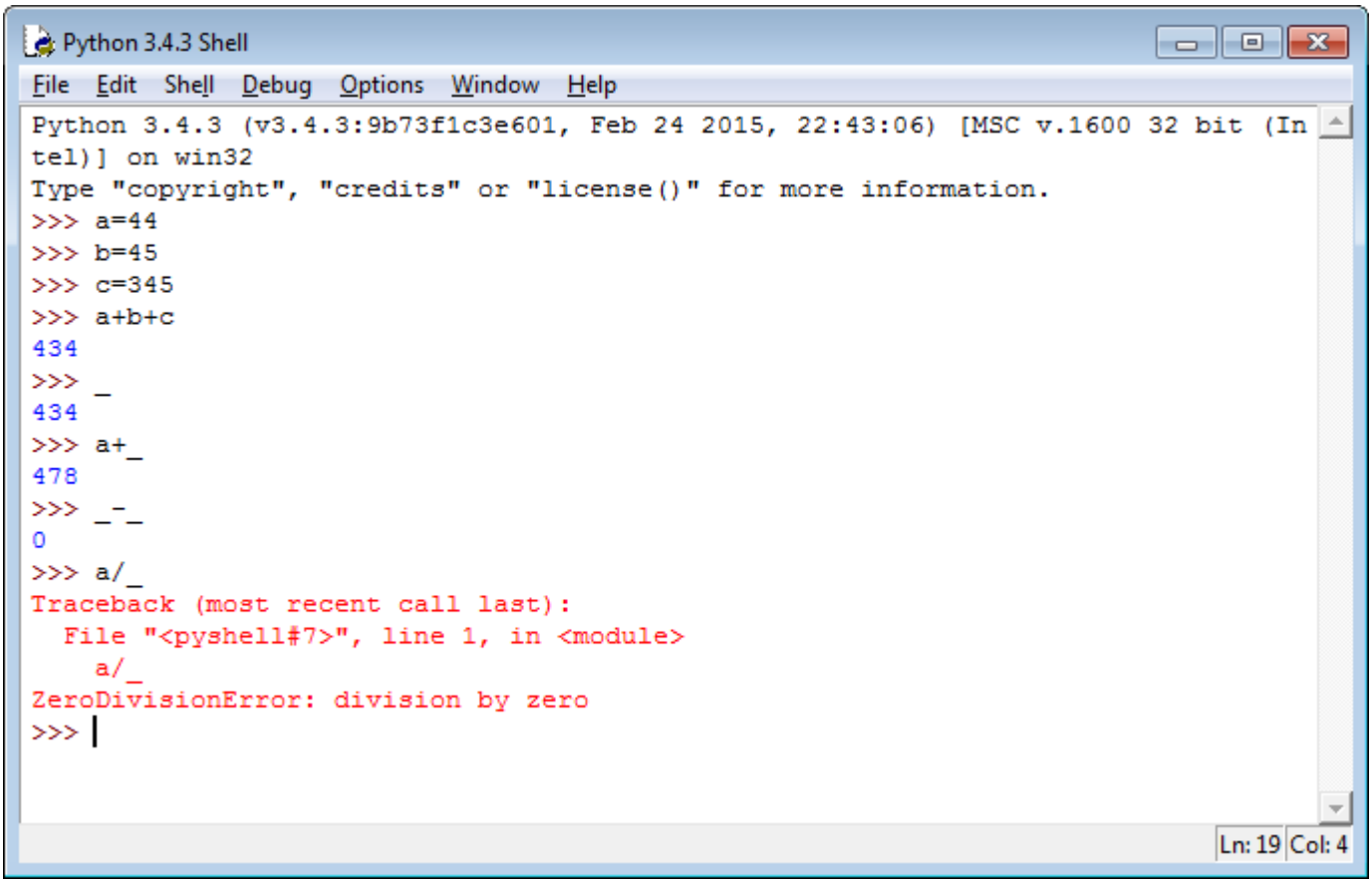
حيث نرى رسالة الخطأ واضحة بعدم قابلية تحويل الاعداد المركبة الى اعداد عشرية ولكننا نستطيع إيجاد القيمة المطلقة للعدد المركب وهي الجذر التربيعي لناتج جمع الجزء الحقيقي تربيع مع الجزء التخيلي تربيع وكما في ادناه:

Abs(a)=sqrt((a.real*a.real)+(a.imag*a.imag))

وكما في ادناه:



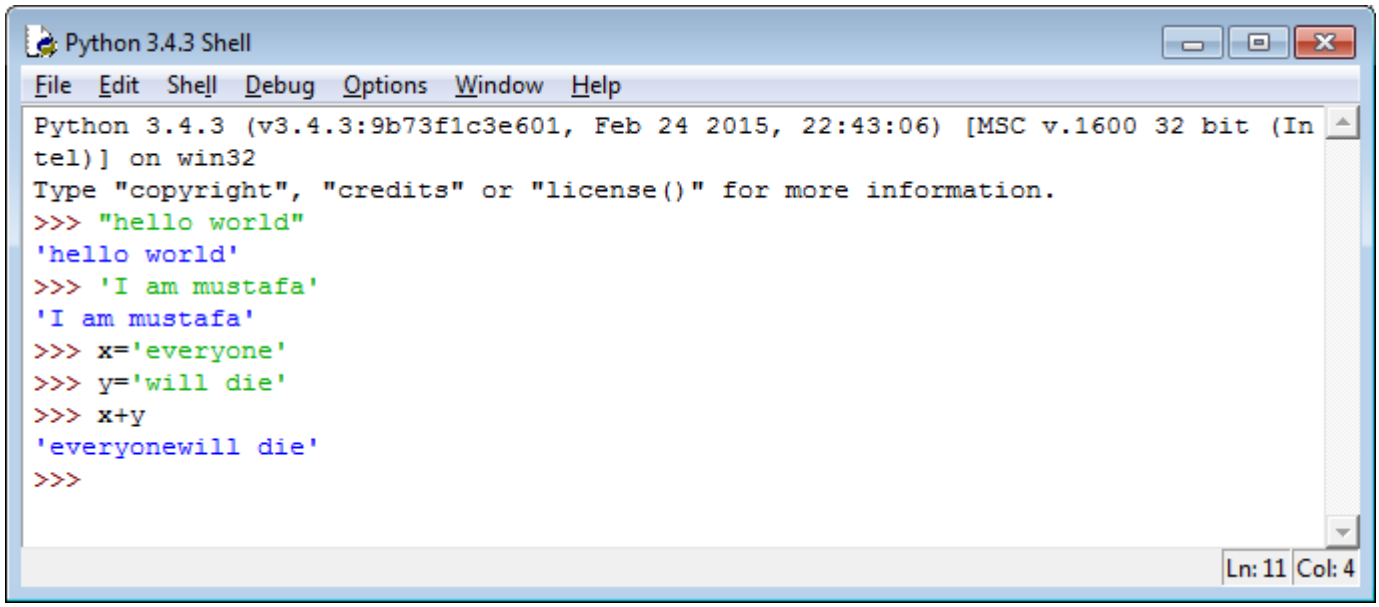
ملاحظة أخرى: عند التعامل مع المفسر كحاسبة لأجراء العمليات الحسابية فأن اخر نتيجة او رقم يتم طباعته يتم حفظه في متغير خاص داخل المفسر اسمه (_) بحيث يمكن التعامل معه على انه متغير يدخل في عمليات حسابية أخرى وكما في ادناه:



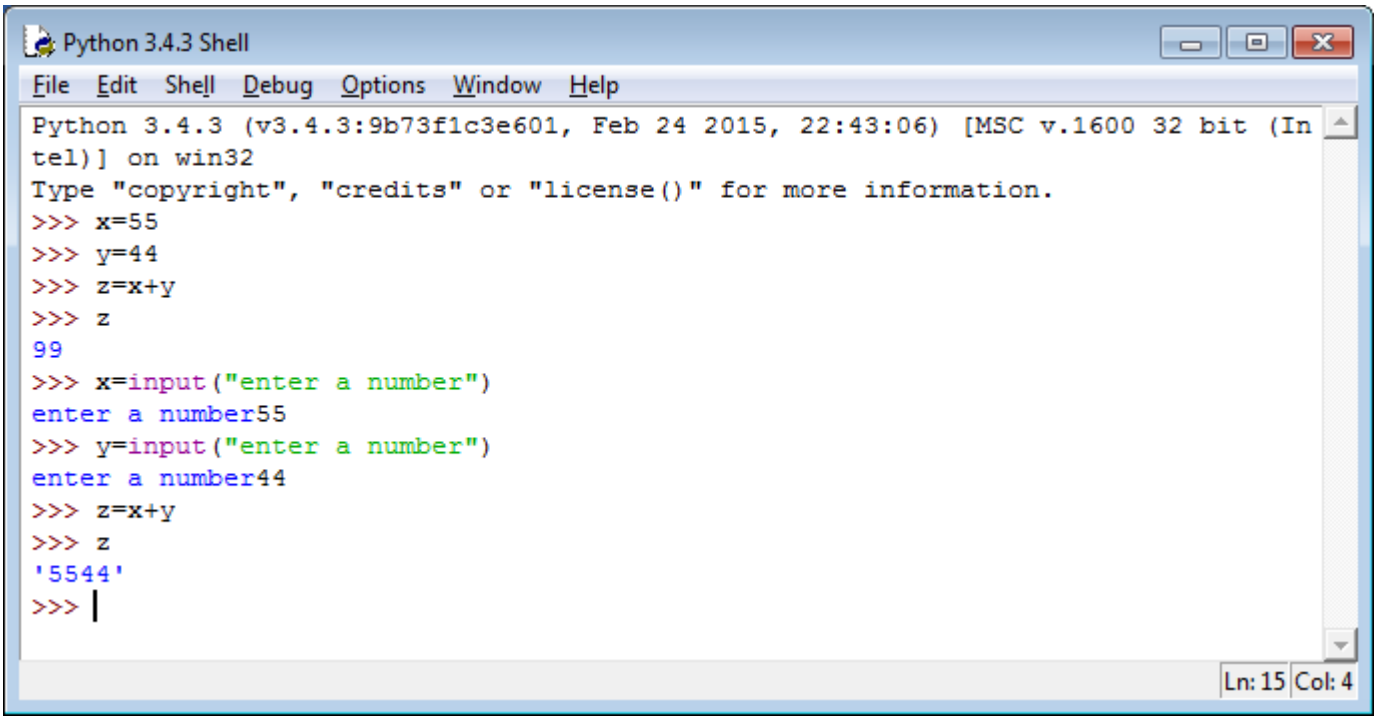
وكما نرى هنا فقد تم التعامل مع المتغير (_) على انه قيمة عددية تدخل في الحسابات وحين قمنا بطرحه من نفسه وكان الناتج صفر، تم خزن الصفر في نفس المتغير وحين حاولنا القسمة عليه ظهر خطأ عدم قابلية القسمة على صفر.

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:

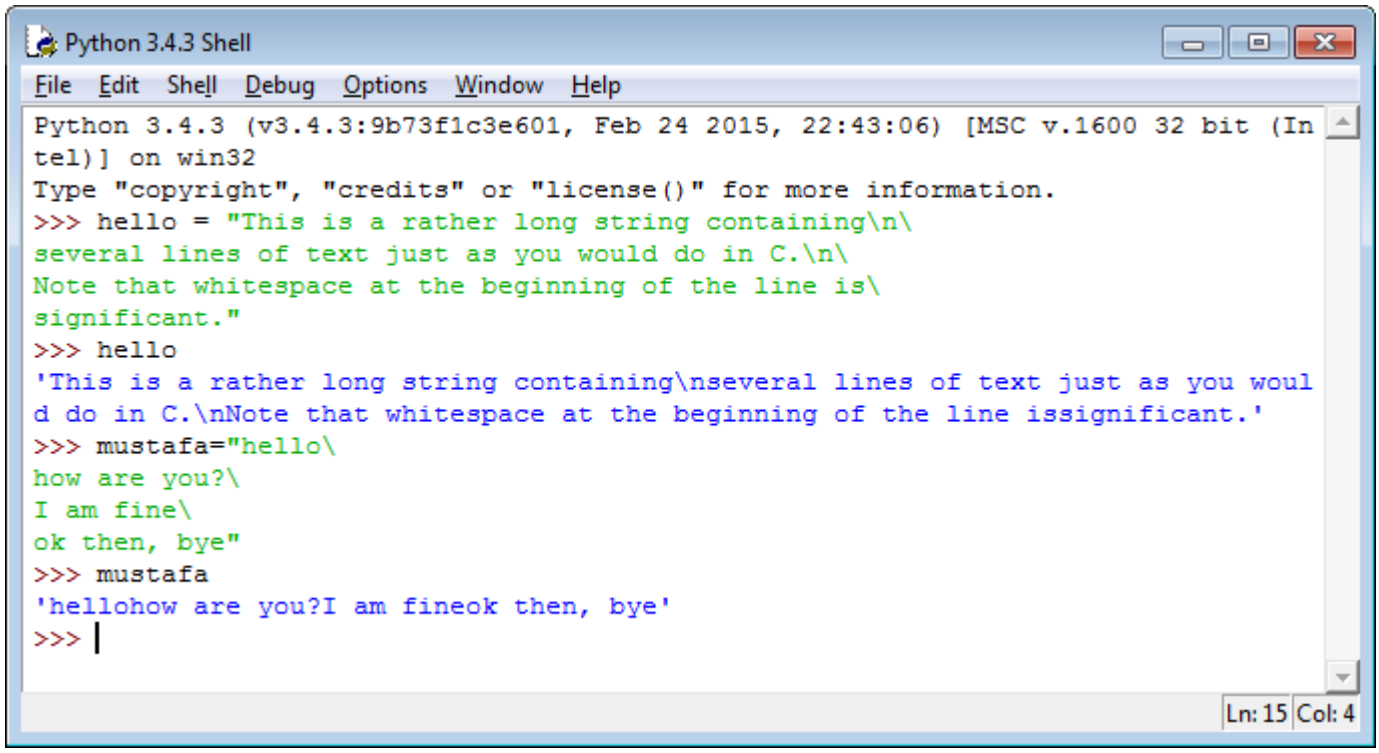


وكما نرى هنا فأن أي شيء محصور بعلامات اقتباس يتم طباعته بدون تصرف من المفسر وكذلك فأن اسناد هذه السلاسل الرمزية الى متغيرات وطباعتها مرفقة مع بعضها البعض يتم باستخدام علامة (+) والتي تمثل هنا علامة ارفاق (appending) وليست جمع حيث يتم طباعة المتغير الأول ثم الثاني وهاكم مثالاً اخر لتوضيح الفكرة:

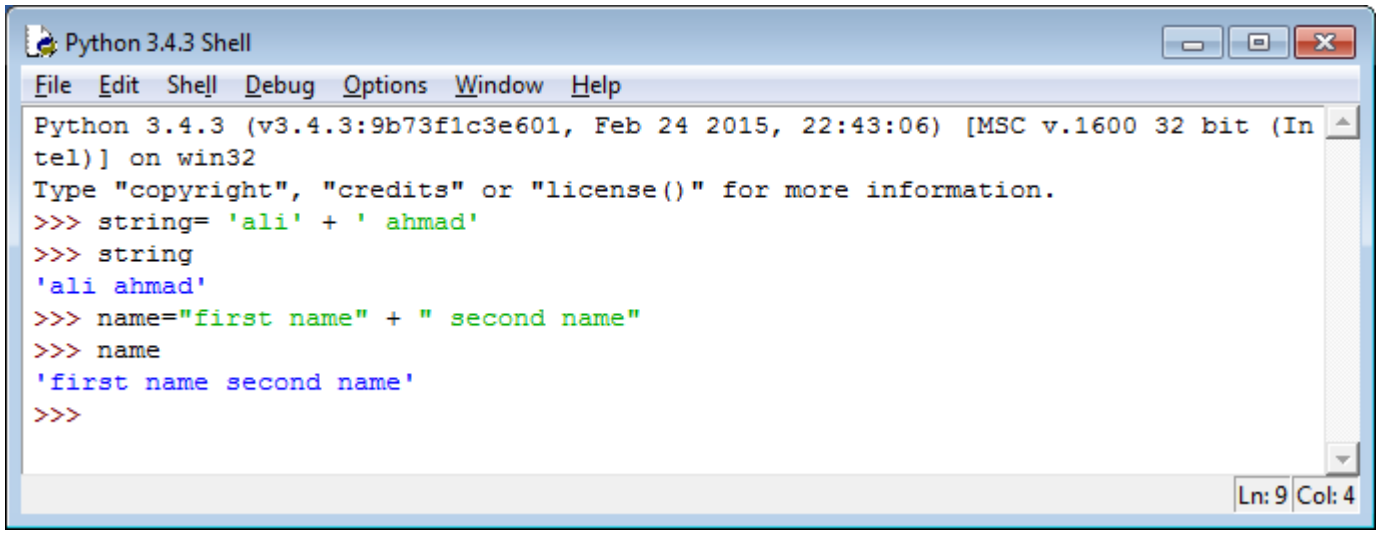


هنا نلاحظ الفرق بين الادخال المباشر وبين الادخال عن طريق أداة (input) ففي المرة الأولى تعامل المفسر مع ال(x) وال (y) على انها متغيرات رقمية وفي المرة الثانية عاملها على انها سلاسل رمزية والتي يمكن تحويلها الى ارقام باستخدام الدالة التي تحدثنا عنها سابقاً ((int(x)) ويمكن مراجعة الدروس السابقة للاطلاع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

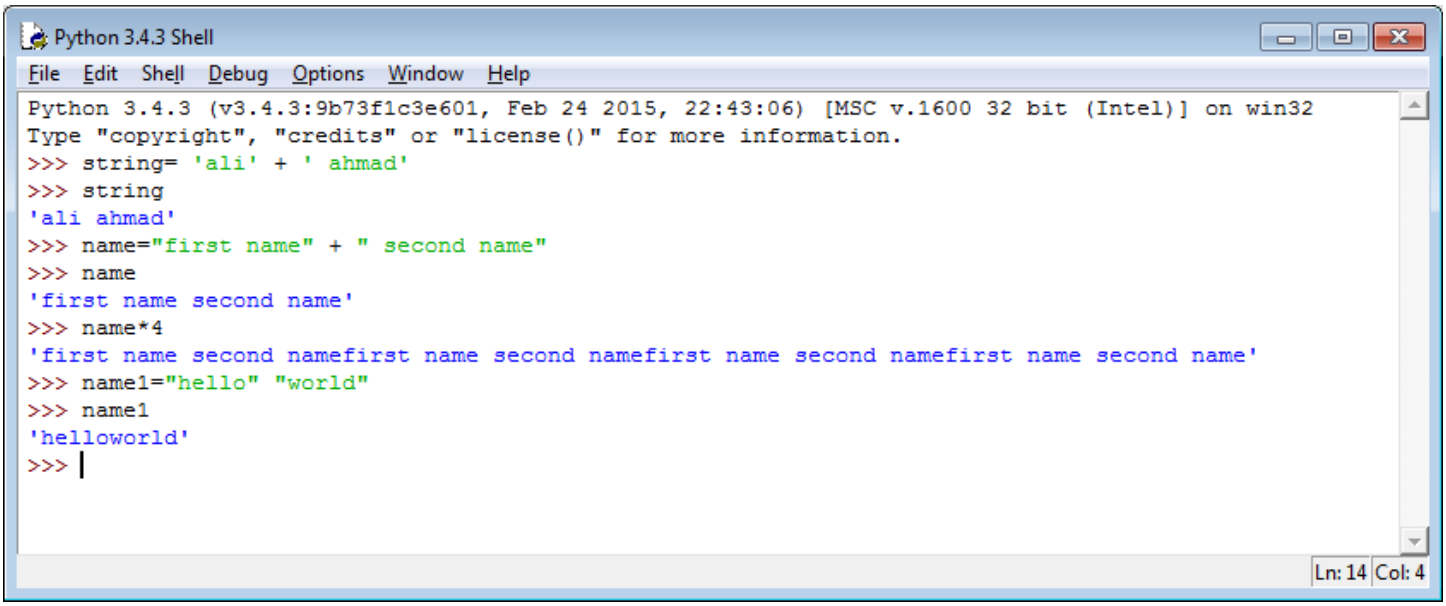
كذلك فأن السلاسل الرمزية يمكن ان تمتد لأكثر من سطر واحد كما هو الحال في بقية لغات البرمجة الأخرى وباستخدام (\n) كما في المثال ادناه:



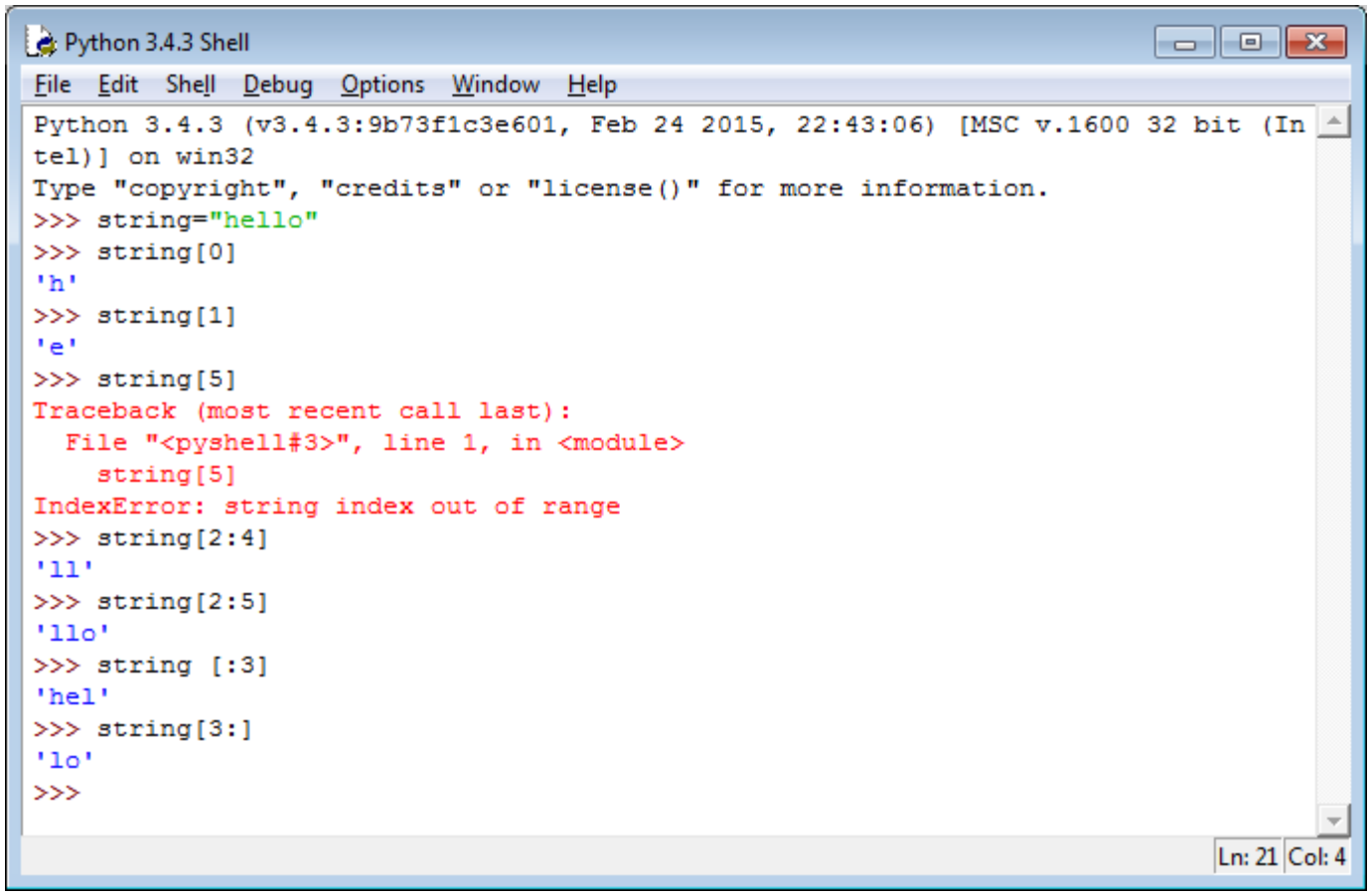
كذلك يمكن ان تتكون السلسلة الرمزية من مجموعة سلاسل رمزية جزئية وكما في ادناه:



لاحظ ايضاً الأمثلة ادناه:



الان وكما في لغة (C) من المهم التعامل مع مكونات السلسلة الرمزية (الرموز) كل على حدة لتغيير او 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 ادناه:



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 الرمزين الثاني والثالث فقط

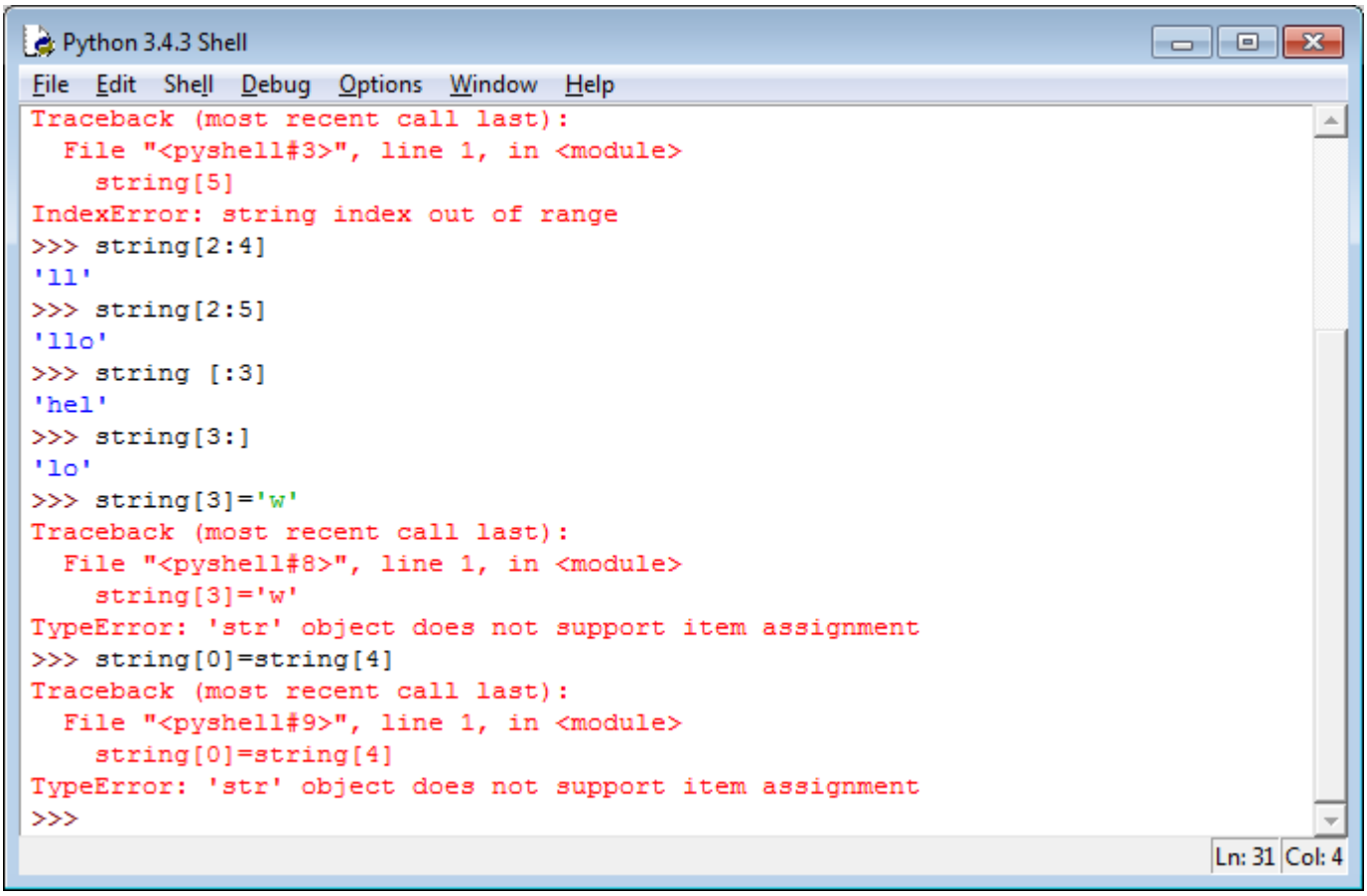
السطر السادس يطلب طباعة الرموز من ٢ الى ٥ مع عدم شمول الرمز الخامس (الغير موجود اصلاً) فقام بطباعة الرموز الثاني والثالث والرابع.

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

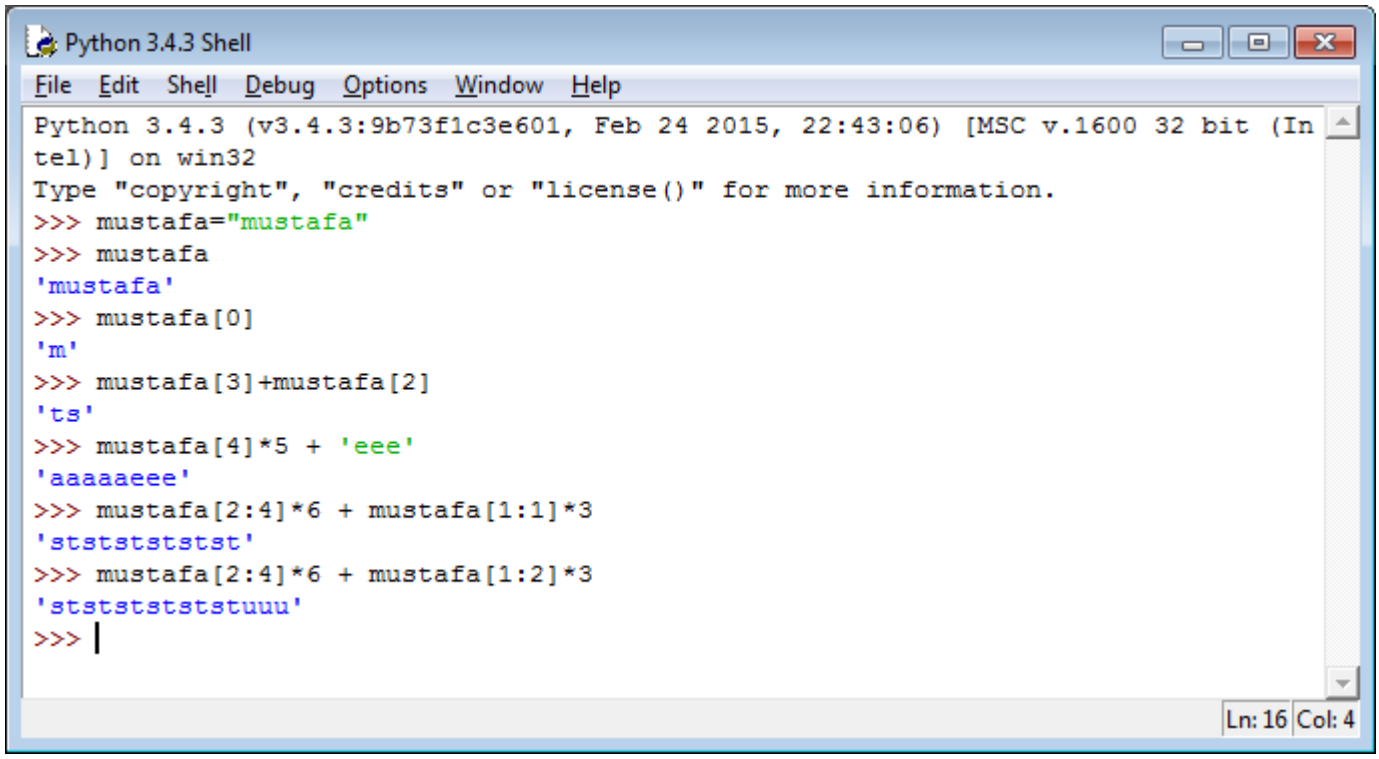
السطر الأخير string[3:] يطلب طباعة الرمز الرابع (ذو الفهرس ٣) وما بعده.

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

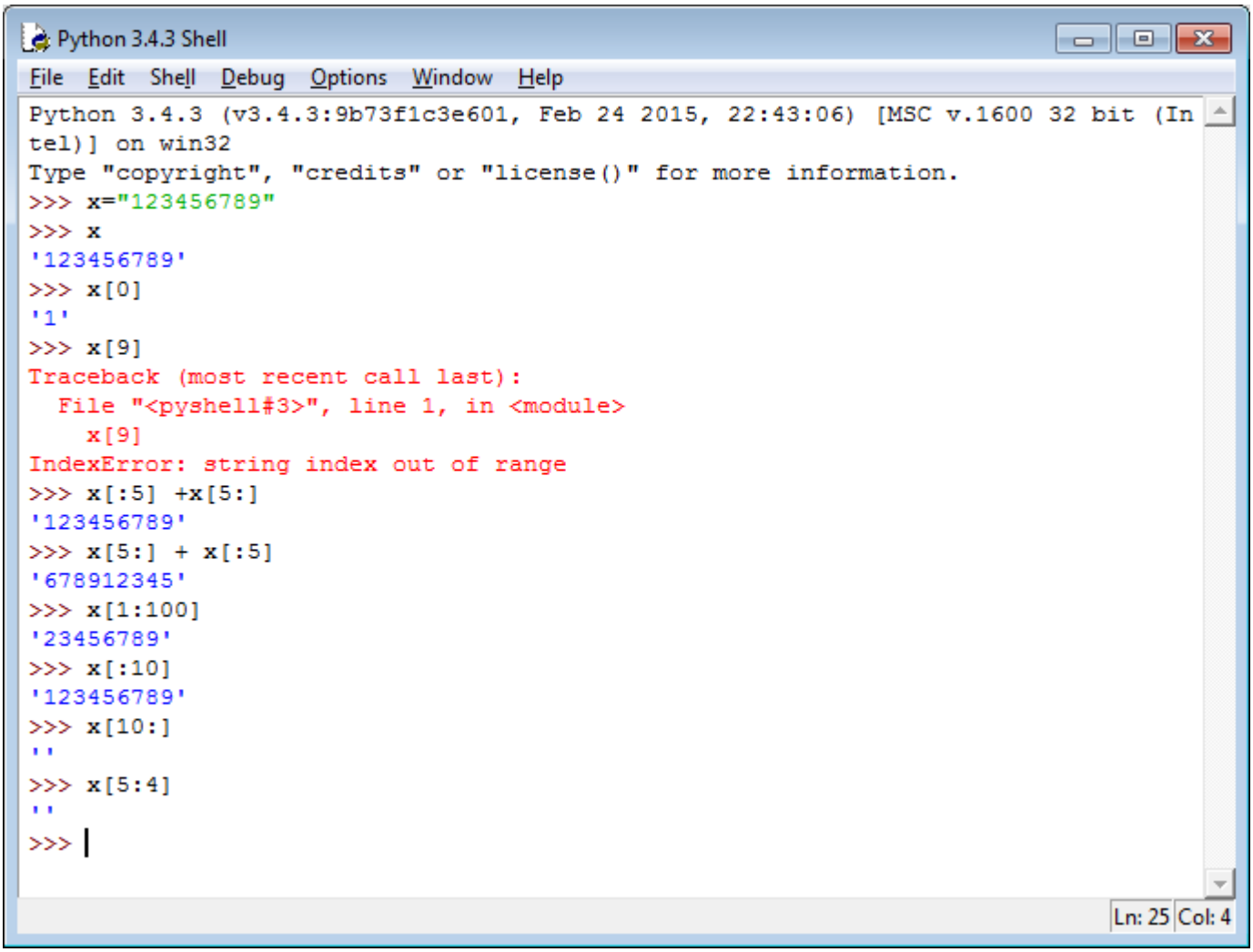
ملاحظة: على خلاف بقية لغات البرمجة فأن لغة بايثون لا تتقبل تغيير قيم رموز داخل السلاسل الرمزية وكما موضح في المثال ادناه:



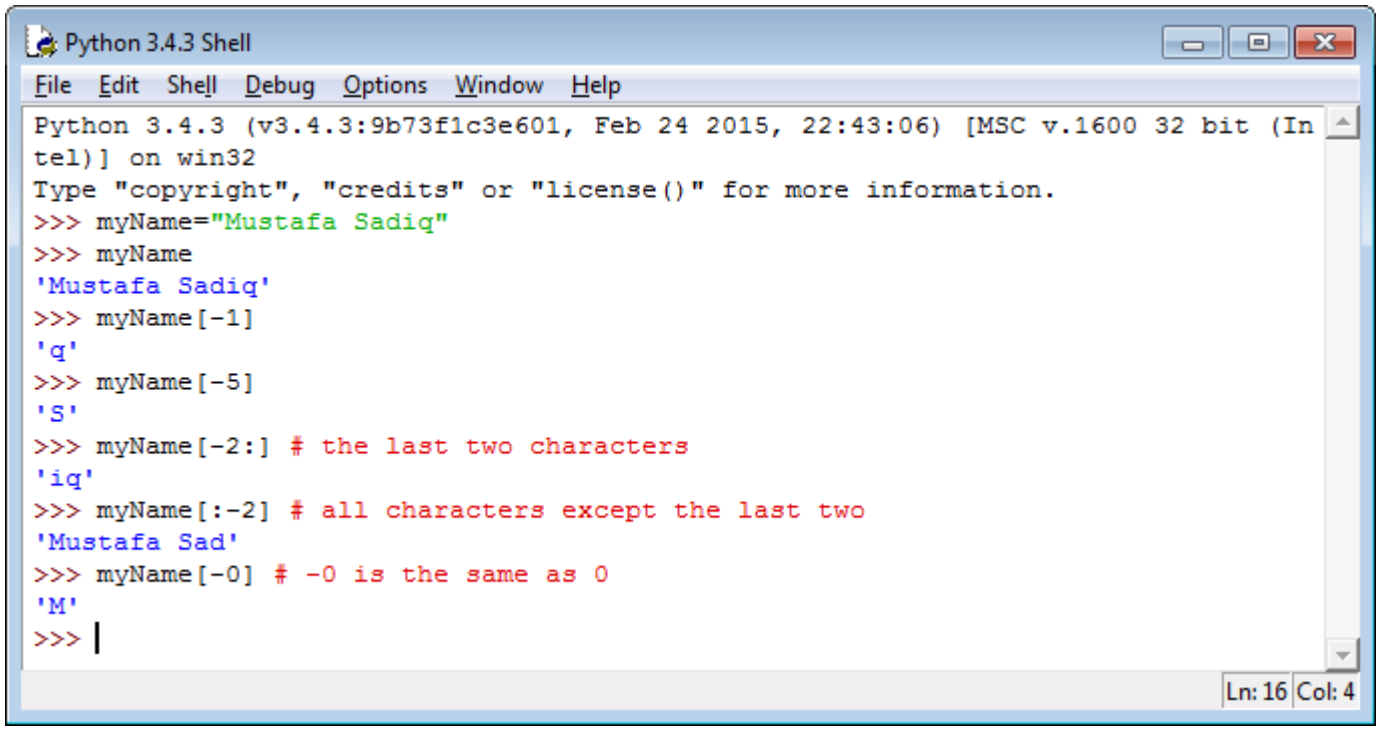
كذلك يمكن انتاج سلاسل رمزية جديدة باستخدام الخصائص أعلاه وكما في المثال التالي:



مزيد من الأمثلة على التعامل مع أجزاء السلاسل الرمزية:

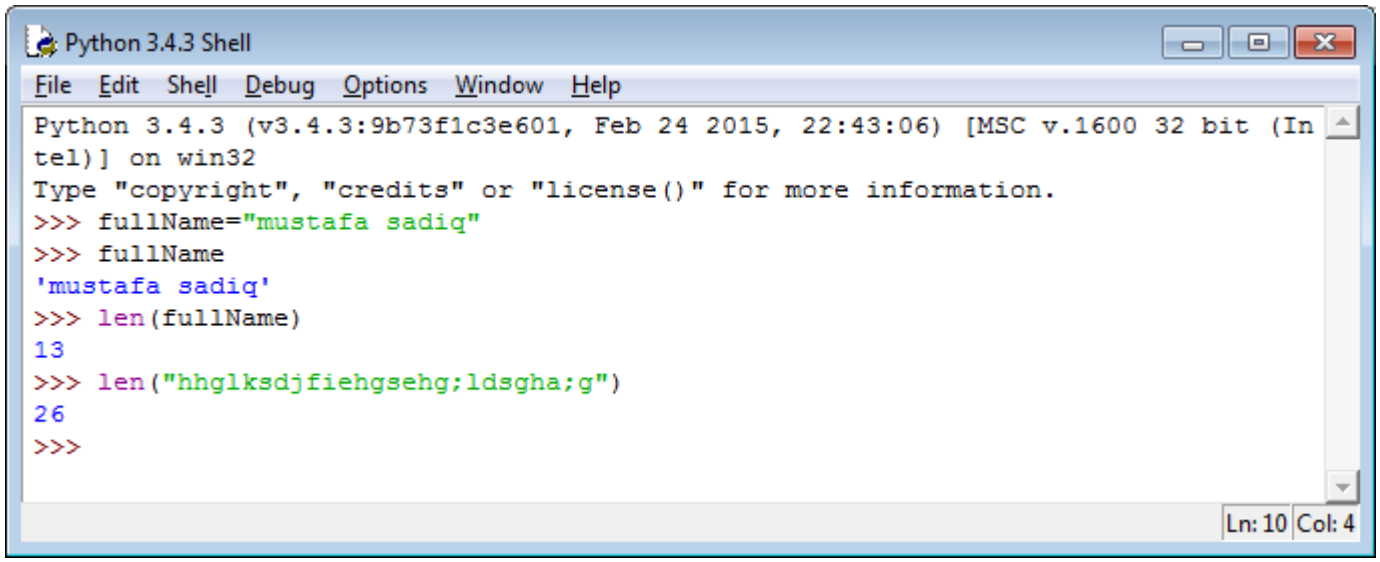


واخيراً نذكر اننا حين نكتب قيم فهارس سالبة فأننا بذلك نخبر المفسر بأن يبدأ الحساب من اليمين الى اليسار أي ان الفهرس سالب واحد يعني اخر رمز في السلسلة والفهرس سالب ٢ هو الرمز قبل الأخير وهكذا وكما في ادناه:



واخيراً يجب الحذر عند التعامل مع فهارس السلاسل الرمزية من اليمين واليسار وملاحظة الترقيم الصحيح لكل اتجاه.

كما في كل لغات البرمجة الأخرى، فأن لغة البايثون توفر دالة حساب طول السلسلة الرمزية وهي len(string) وكما في ادناه:



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 قمتم 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 بالتدريج 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 وأؤكد 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:

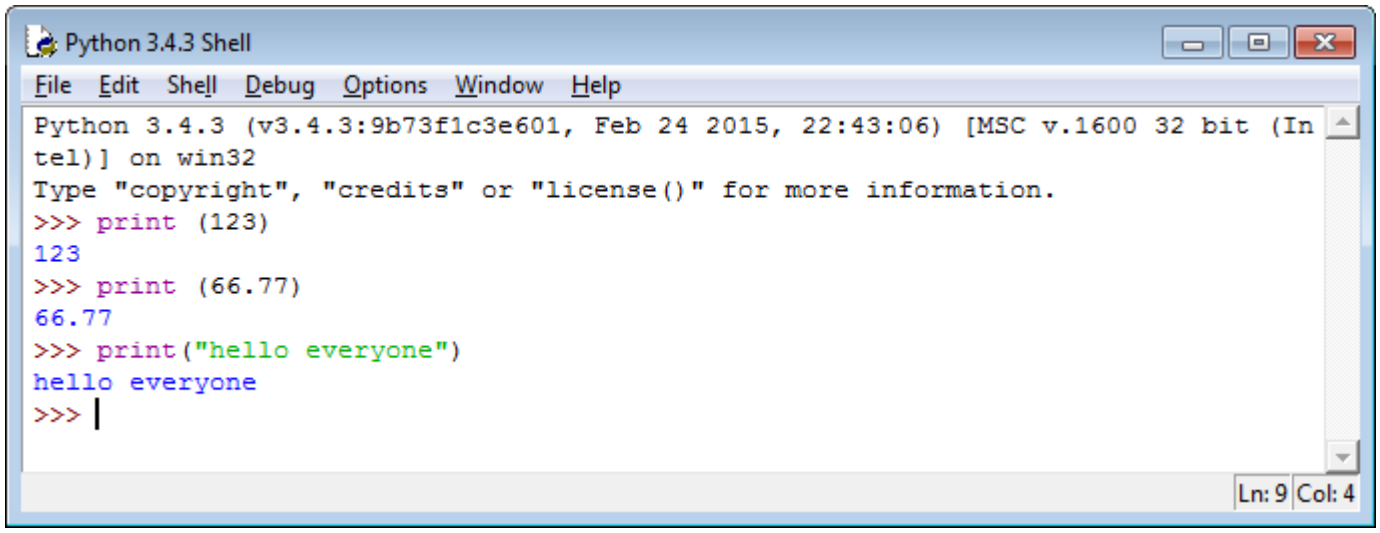
الثوابت في لغة بايثون:

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.

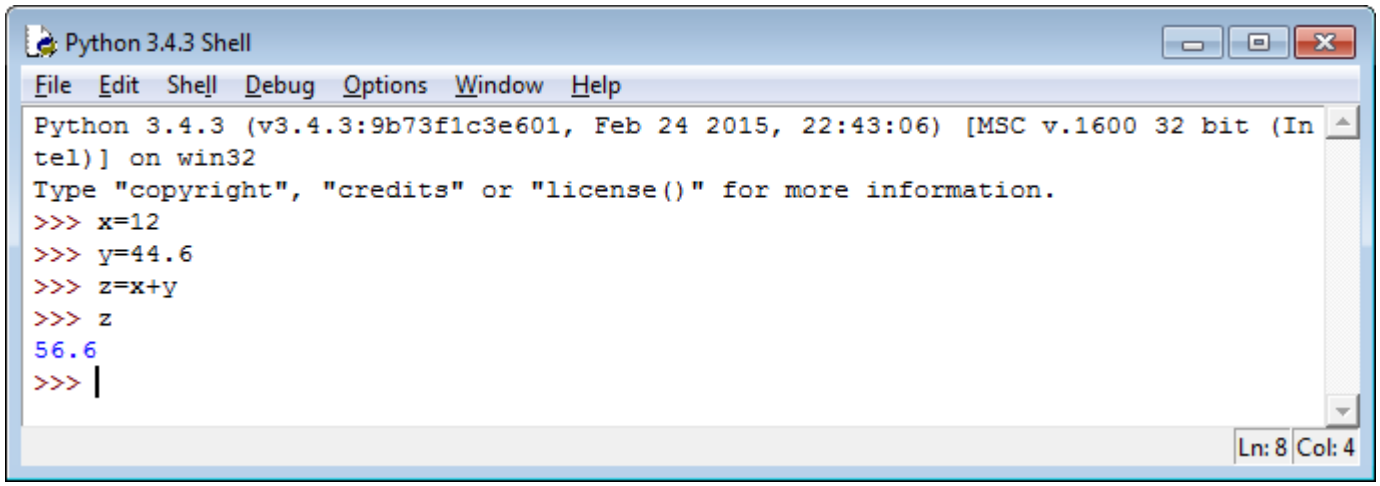
بعض الأمثلة على ادخال الثوابت في بايثون:



المتغيرات في لغة البايثون:

وهي الأسماء التي تطلق على مواقع خزن البيانات في الذاكرة حيث يستطيع المبرمج خزن 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:



شروط تسمية المتغيرات: كل لغات البرمجة فيها شروط معينة لتسمية المتغيرات وهي تقريباً متشابهة في كل لغات البرمجة وتسمل 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.**

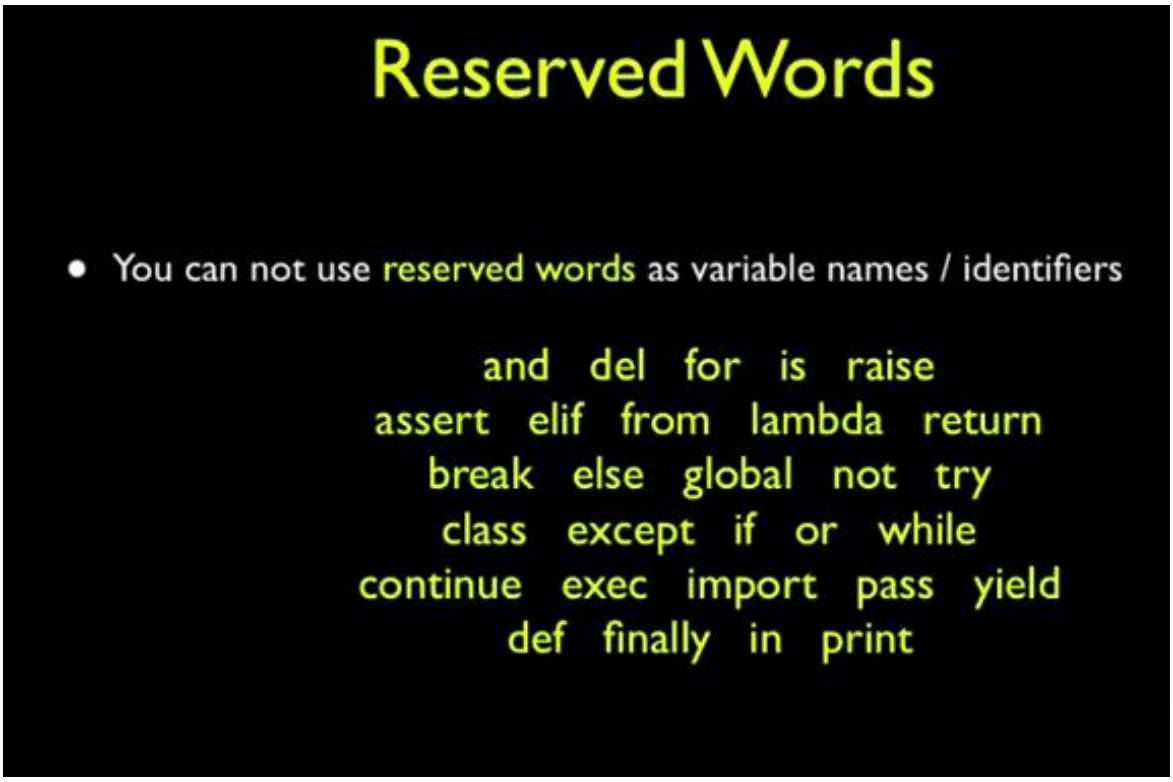
امثلة على أسماء المتغيرات المقبولة وغير المقبولة:

مقبول: _saad Qasim22 ali o1k2j3

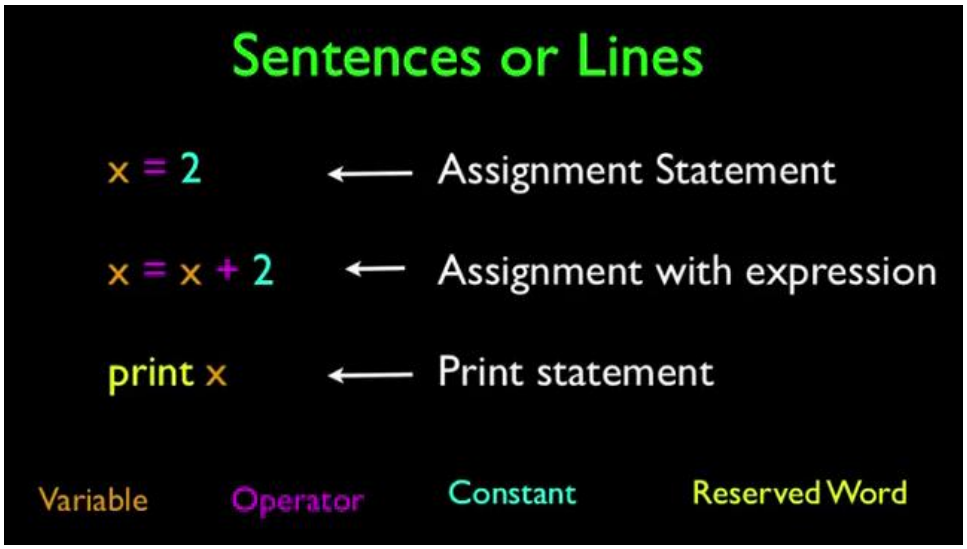
سيء: Alaa ALAA alaa لأنها متشابهة وتسبب مشاكل للمبرمجين

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:**



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

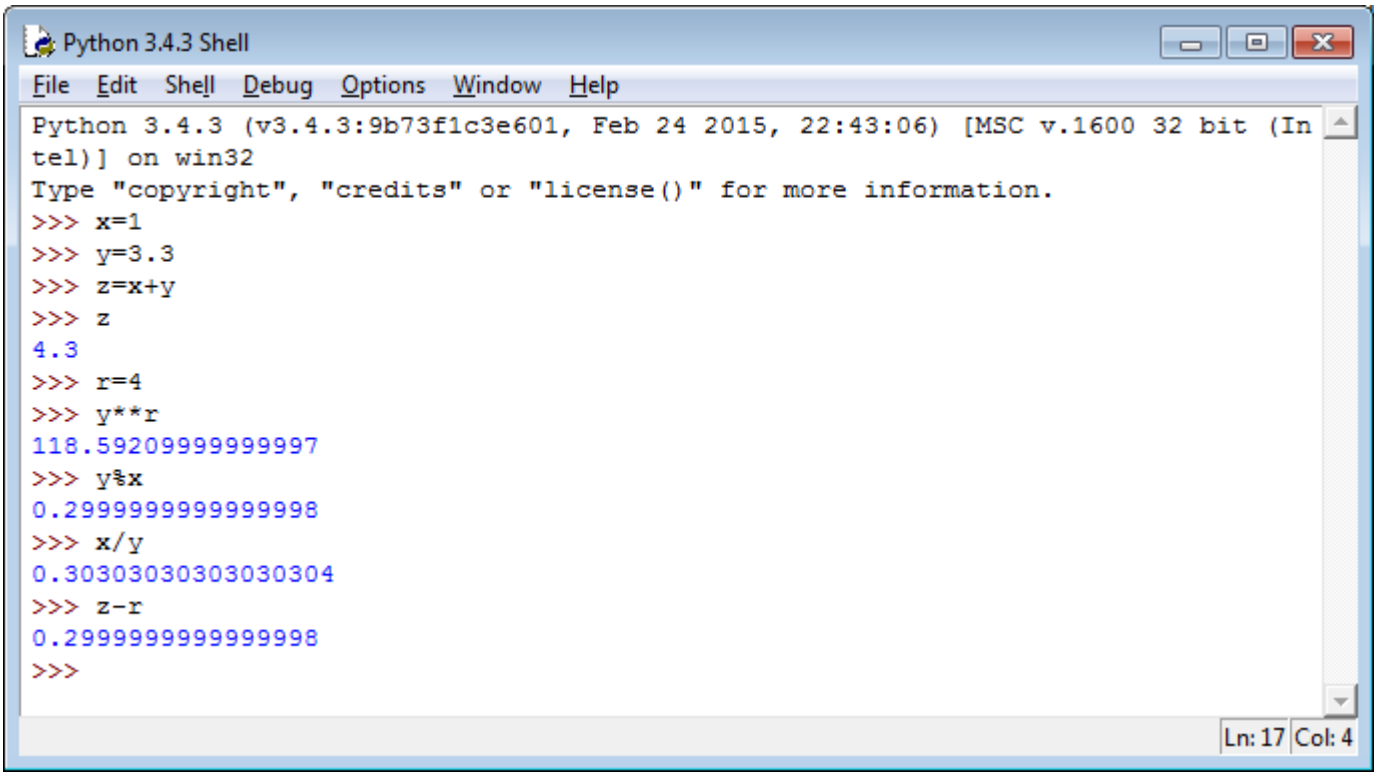


حيث تكون بعض الجمل البرمجية هي عبارات اسناد فقط مثل (x=2) وبعضها تكون جمل اسناد وتعابير رياضية مثل (y=x+2) وبعضها عبارات ادخال او طباعة او شروط او تكرار وكما سنرى في الدروس القادمة ان شاء الله.

العمليات الرياضية الأساسية للغة البايثون:

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

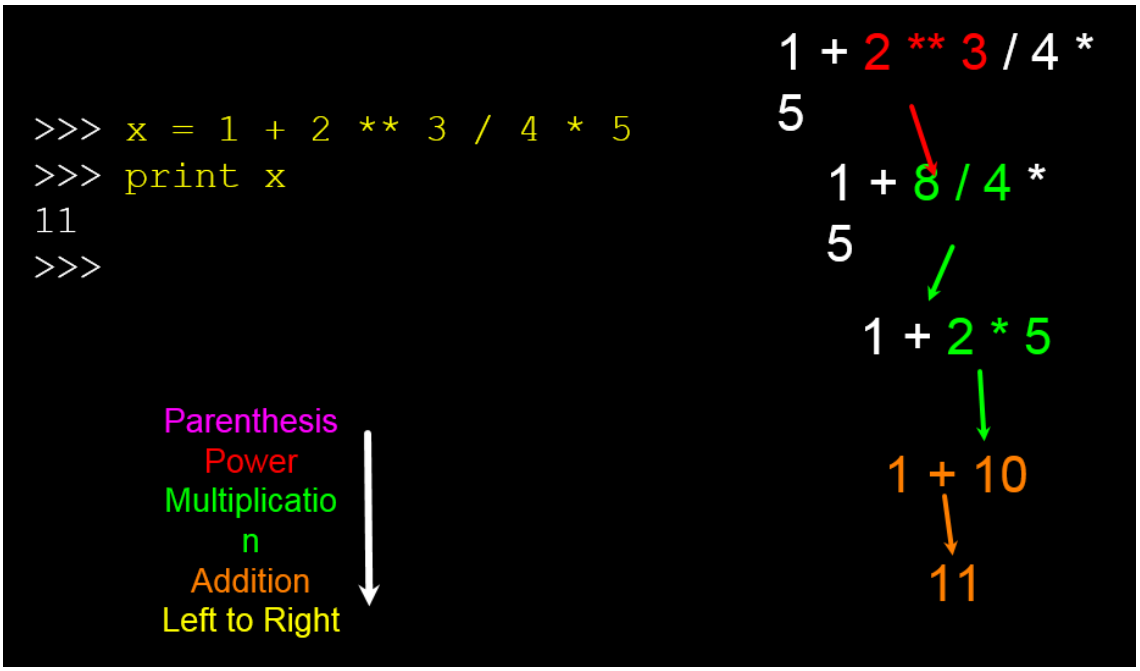
امثلة على تطبيق العمليات الرياضية في بايثون:



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

- **الاقواس**
- **الأسس**
- **الظرب والقسمة**
- **الجمع والطرح**
- **من اليسار الى اليمين**

مثال:



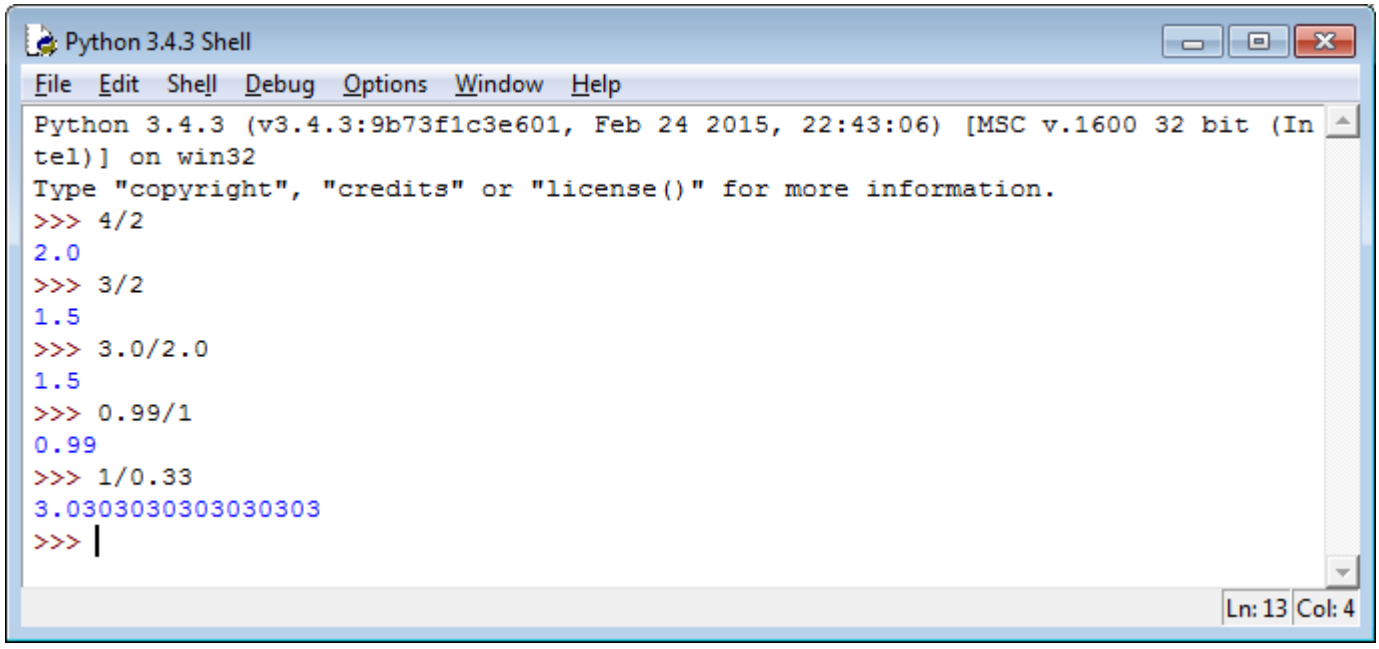
ملاحظة: لتبسيط الأمور على المفسر والقاري فيما بعد يفضل فصل كل العمليات ذات الأولوية المهمة بأقواس عن بقية أجزاء العمليات الرياضية.

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
```

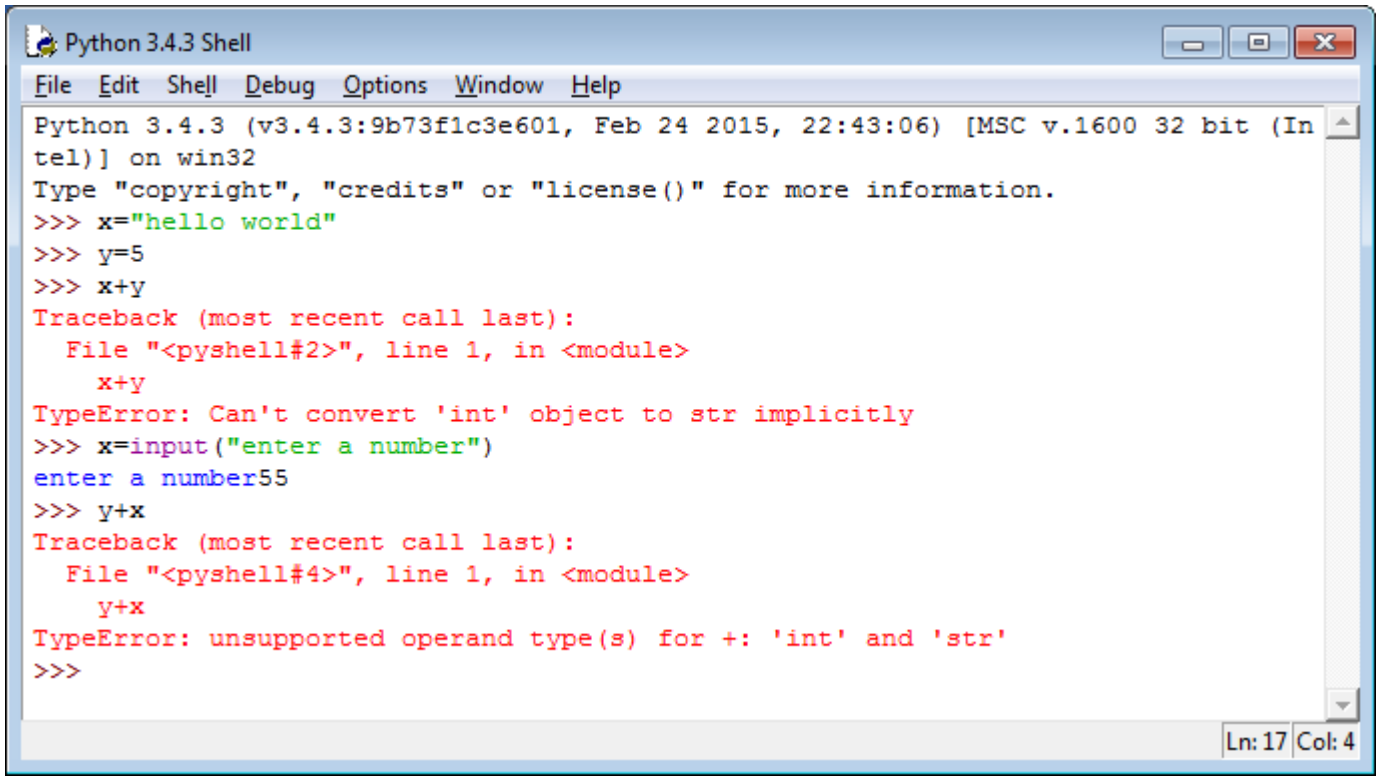
اما في البايثون ٣



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 اجراء بعض العمليات الحسابية البسيطة بين الثوابت والمتغيرات من (أنواع) مختلفة وكما في ادناه:

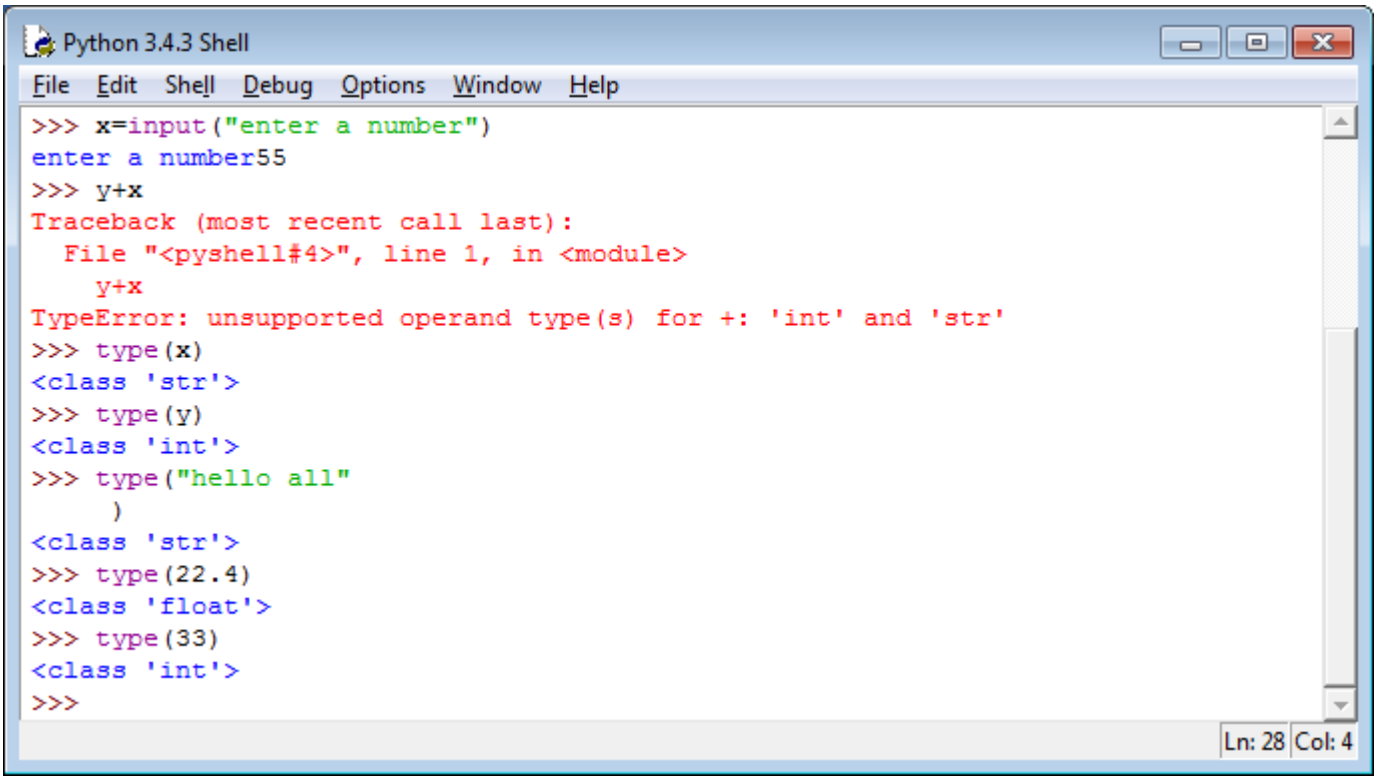


والسبب في الخطأ الأول هو محاولتنا جمع عدد صحيح مع سلسلة رمزية وهو خطأ بديهي طبعاً.

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:



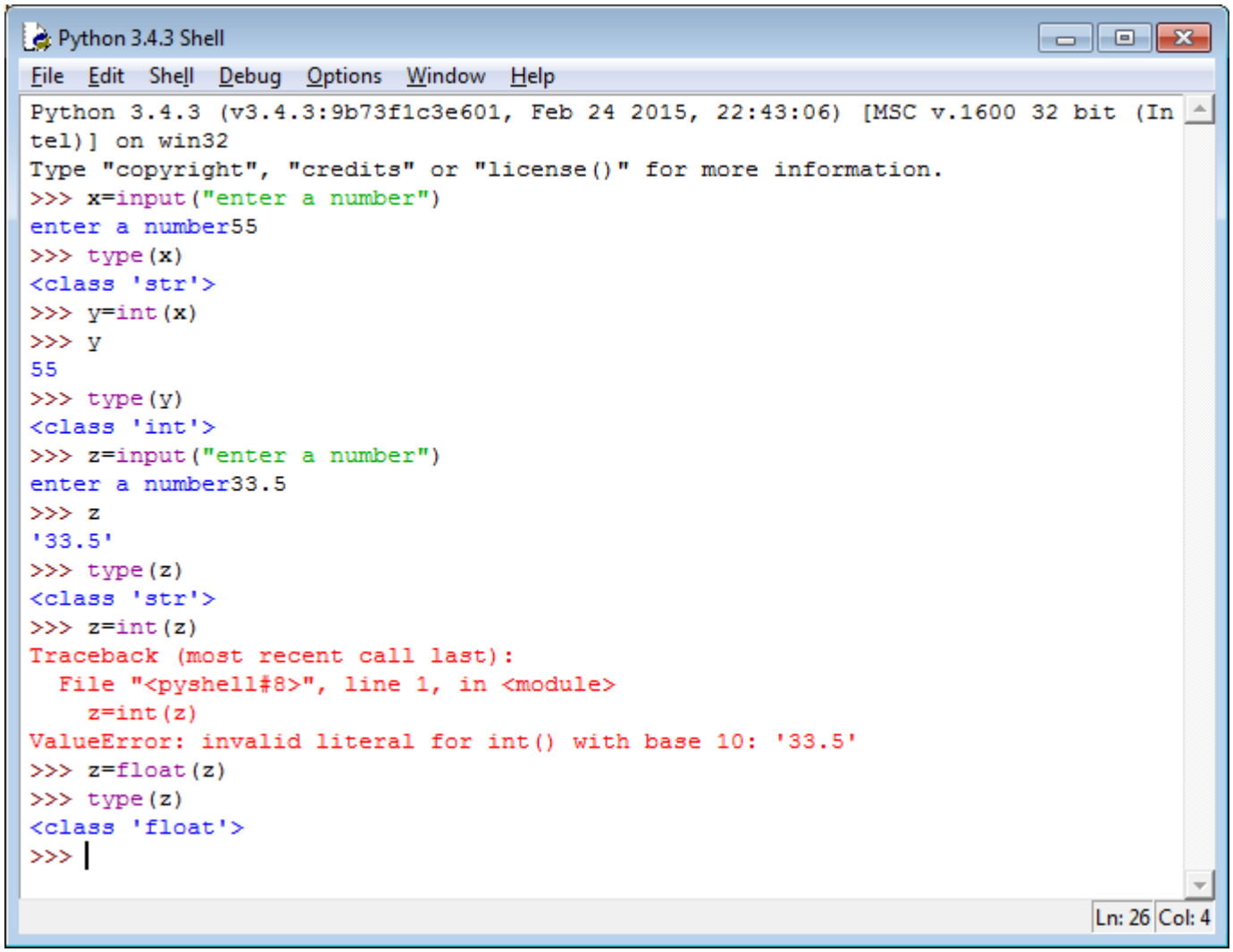
ومن هنا عرفنا ما هو نوع كل متغير او ثابت بوضعه بين قوسين وسبق ذلك بكلمة type والتي تساعد في حل مشاكل الأنواع واما التحويل بين الأنواع فقد تحدثنا عن بعضه سابقاً وذلك بتحويل السلسلة الرمزية الى عدد صحيح باستخدام دالة:

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، فأن لغة بايثون تتعامل مع الشروط بشكل ممتاز وباستخدام عبارة (if) وبالصيغة التالية:

If condition:

Statements

Or:

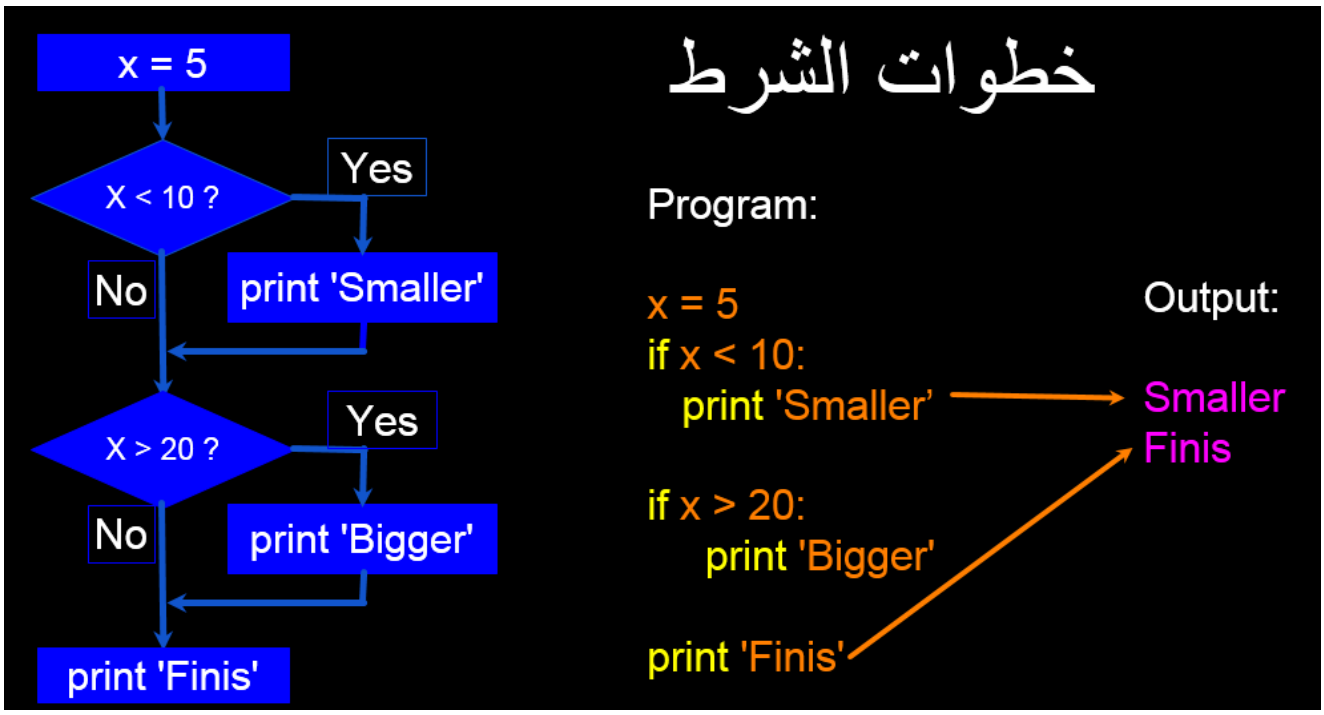
If condition:

Statements

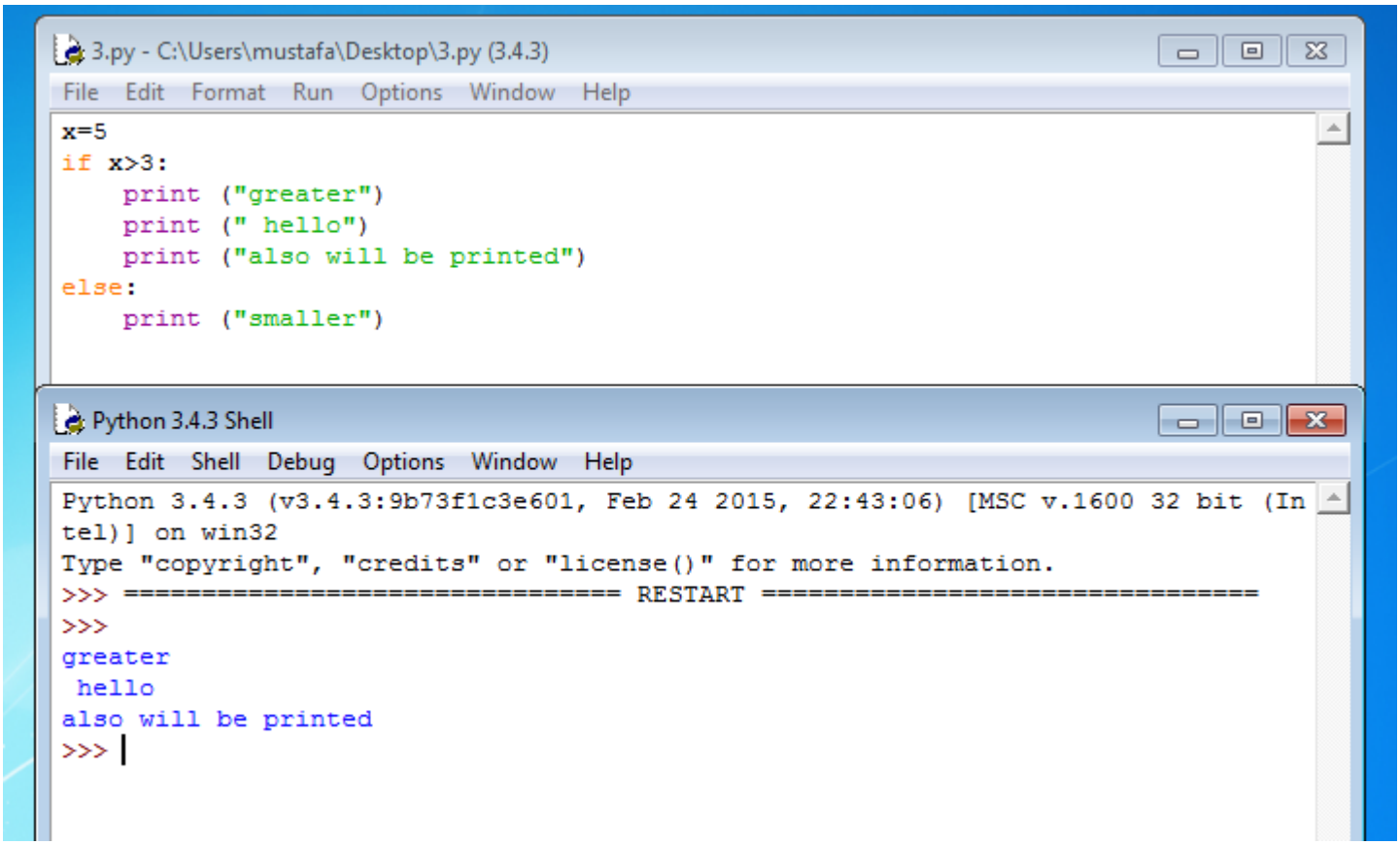
Else:

Statements

وكما يعرف الجميع فأن 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:



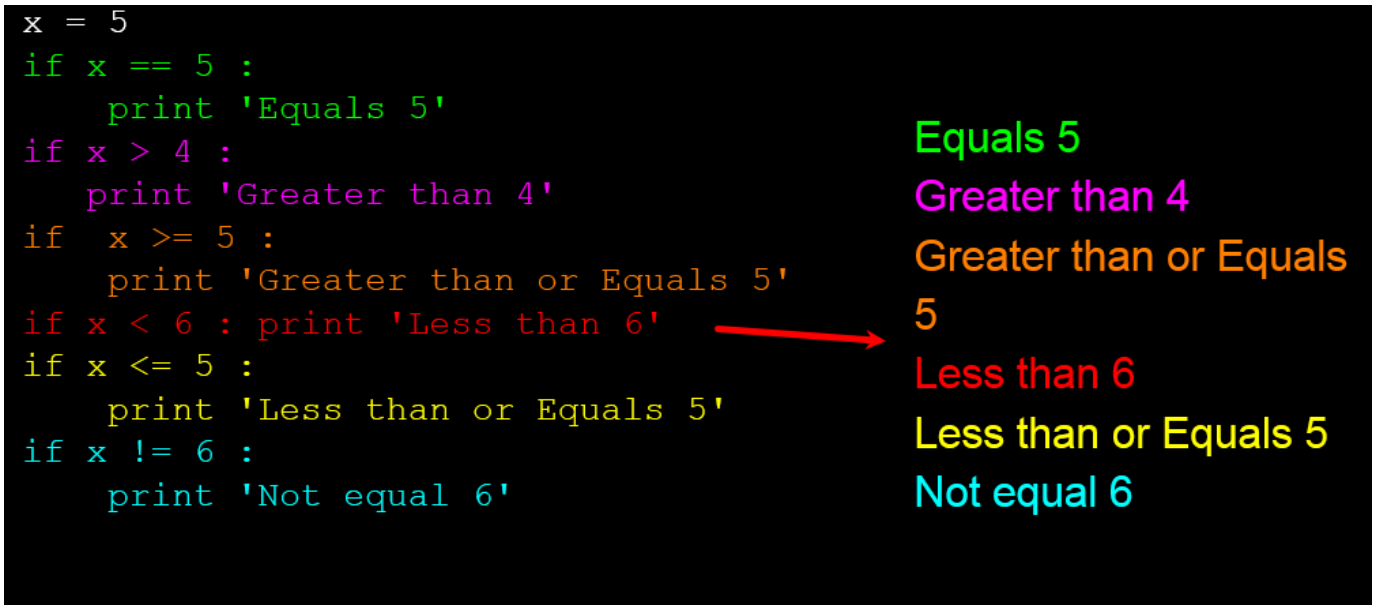
وكمثال على ذلك لاحظوا البرنامج التالي:



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 لأنهاء 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:

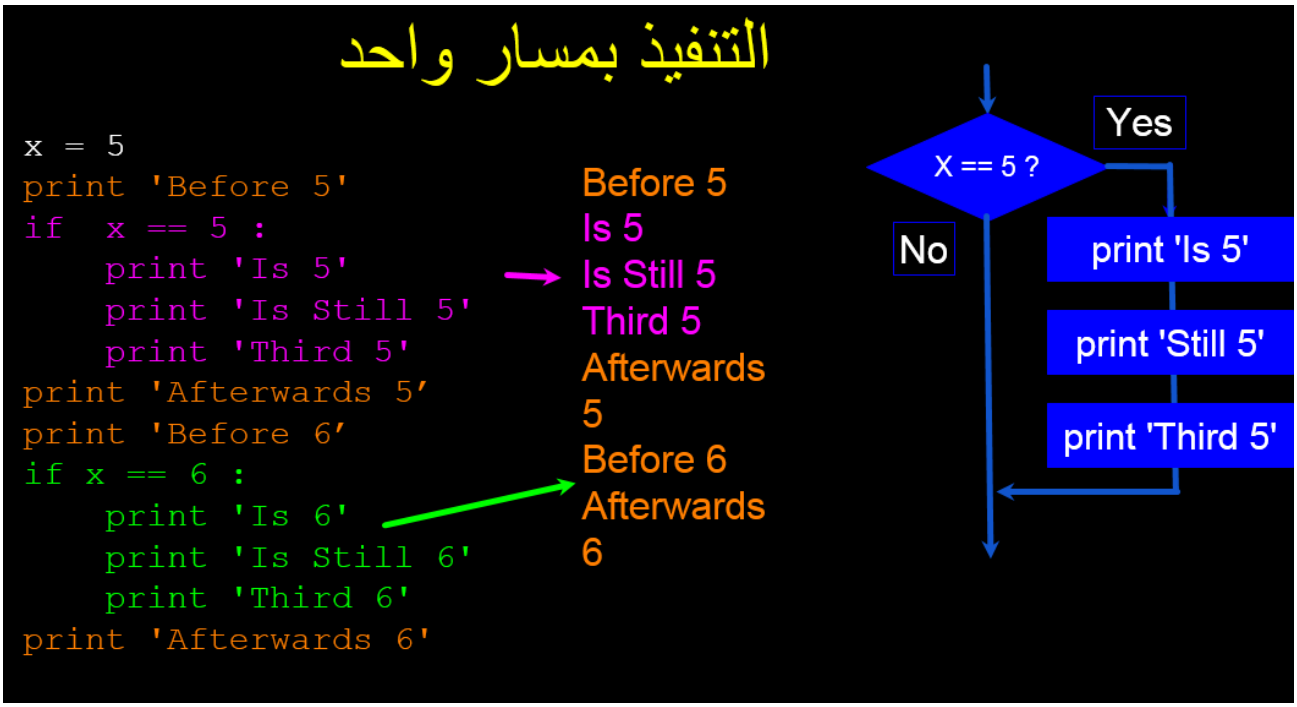
معانيها	أدوات المقارنة المنطقية
اقل من	<
اقل من او يساوي	<=
هل يساوي؟	==
اكبر من او يساوي	>=
اكبر من	>
لا يساوي	!=

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



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

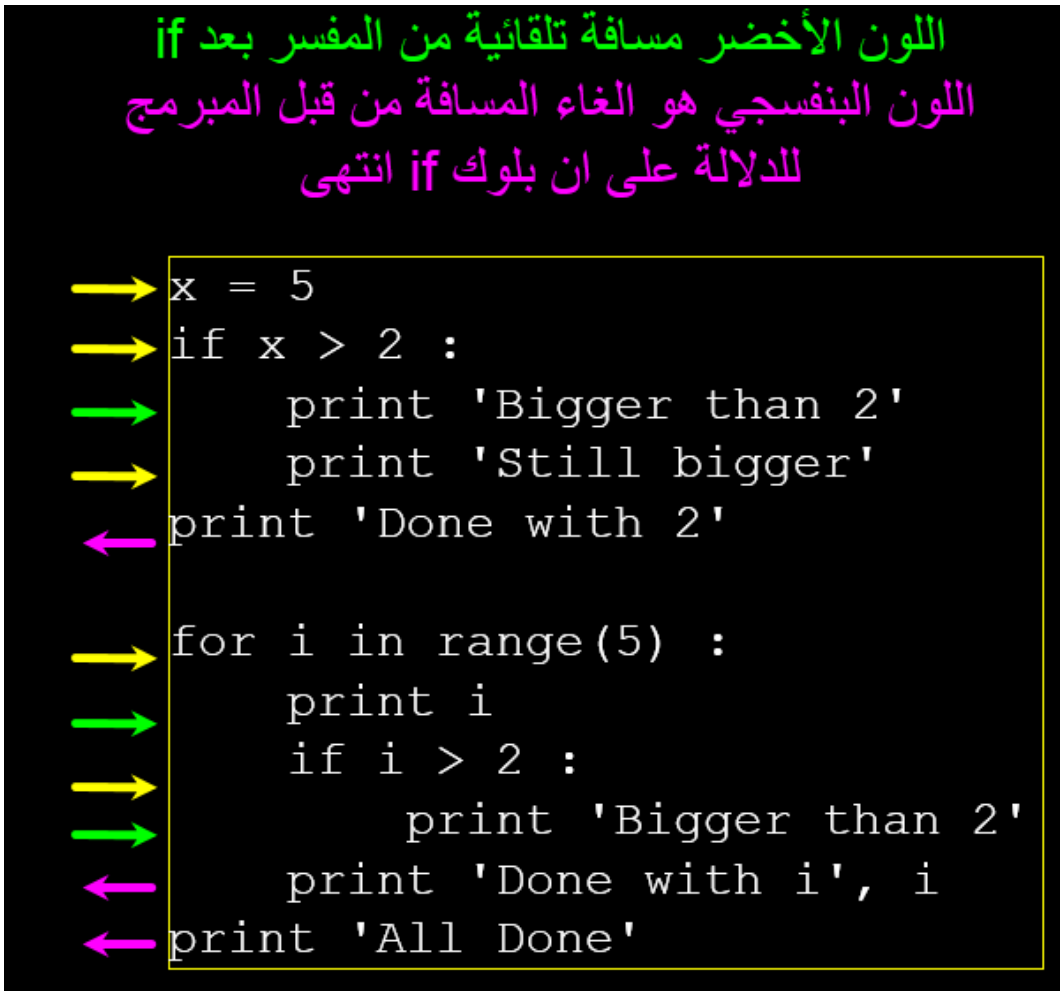
كما يعرف المبرمجون ببقية لغات البرمجة، فأن عبارة (if) تحتوي الكثير من الخيارات والمميزات ومنها انها يمكن ان تستخدم في التنفيذ بمسار واحد او بعدة مسارات وبحسب نوعية استخدامها وكما في التفصيل ادناه:



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 فأن 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:

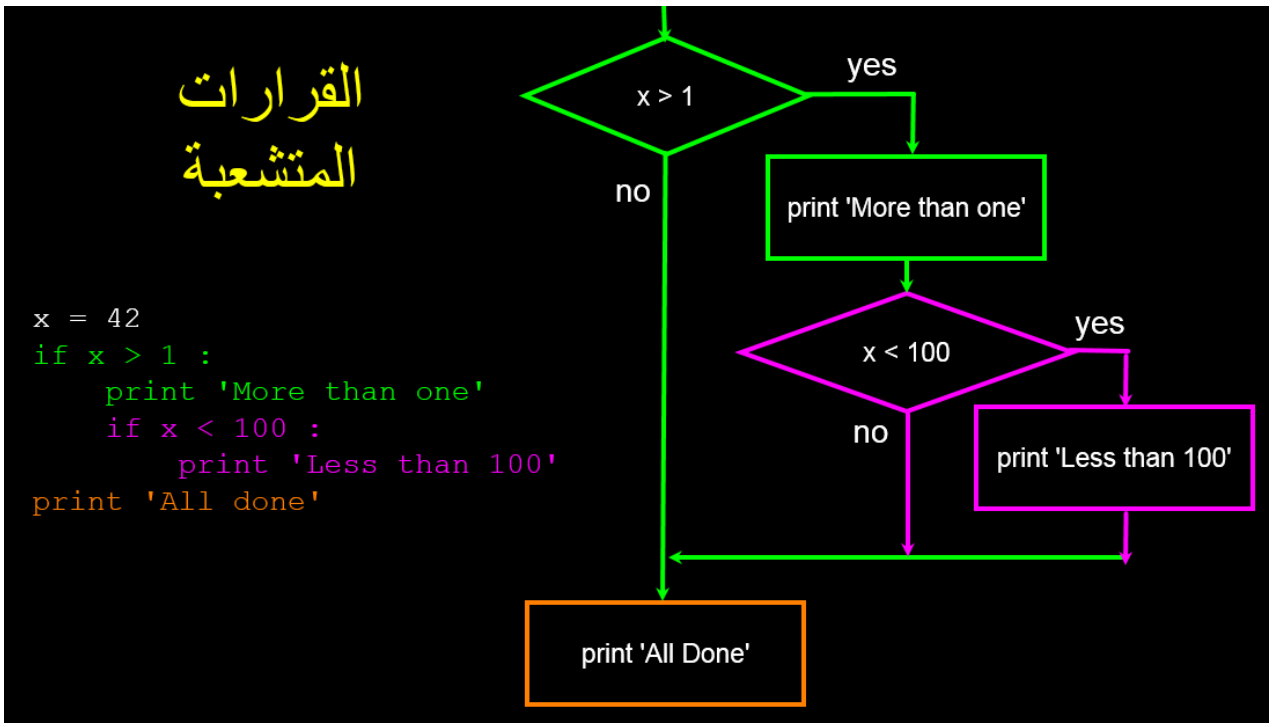


ولتوضيح البلوكات المتداخلة نضعها بألوان مختلفة هنا:

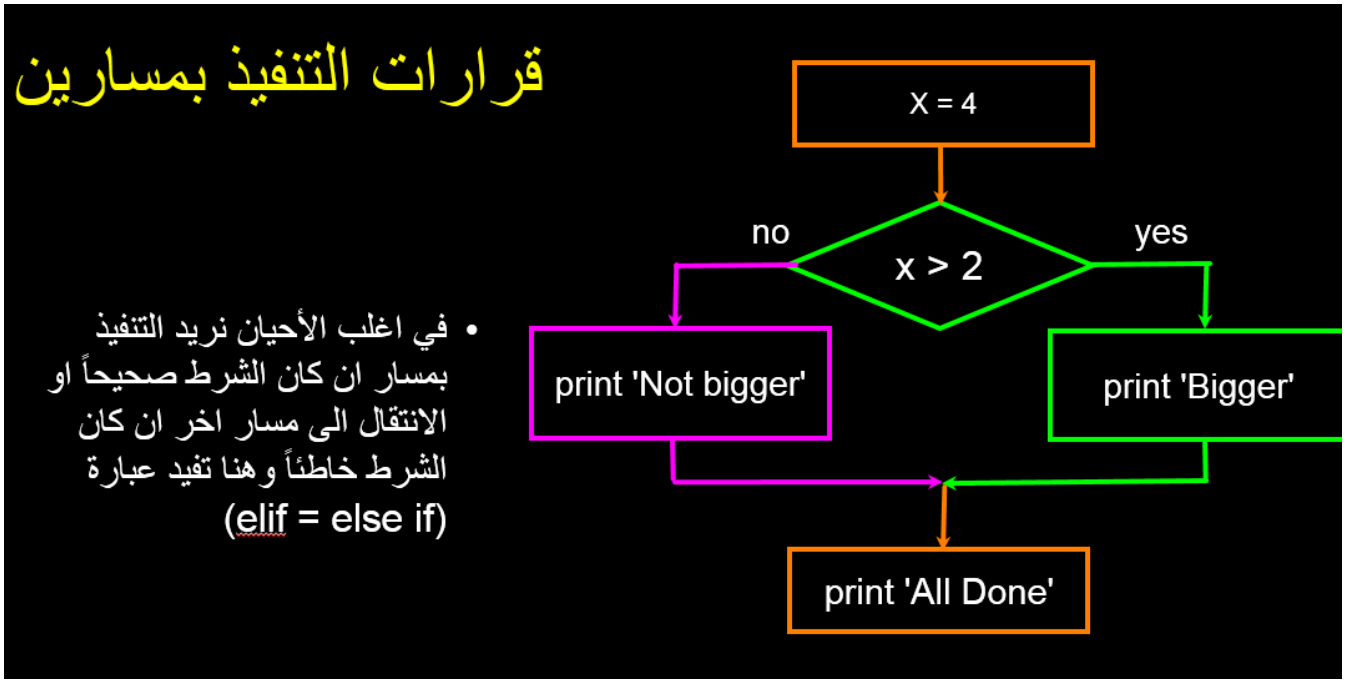
```
x = 5
if x > 2 :
    print 'Bigger than 2'
    print 'Still bigger'
print 'Done with 2'

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

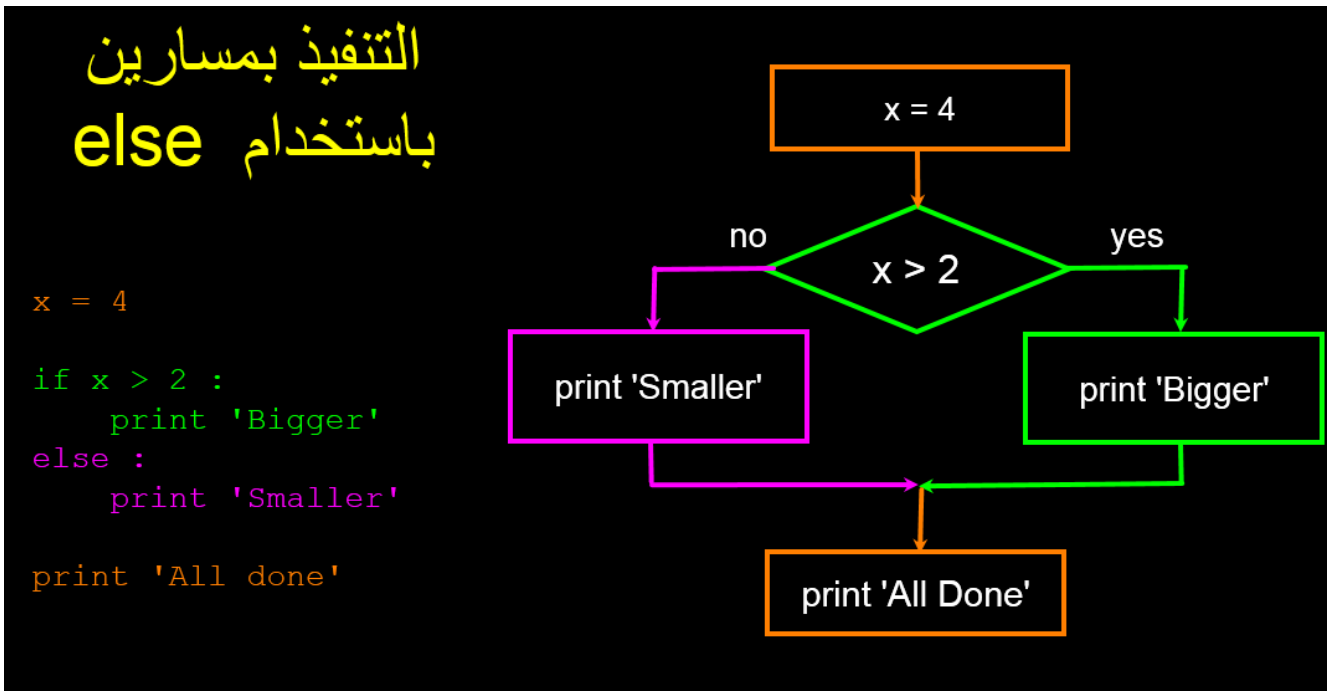
لاحظ عدم وجود اقواس وانما مسافات فقط.



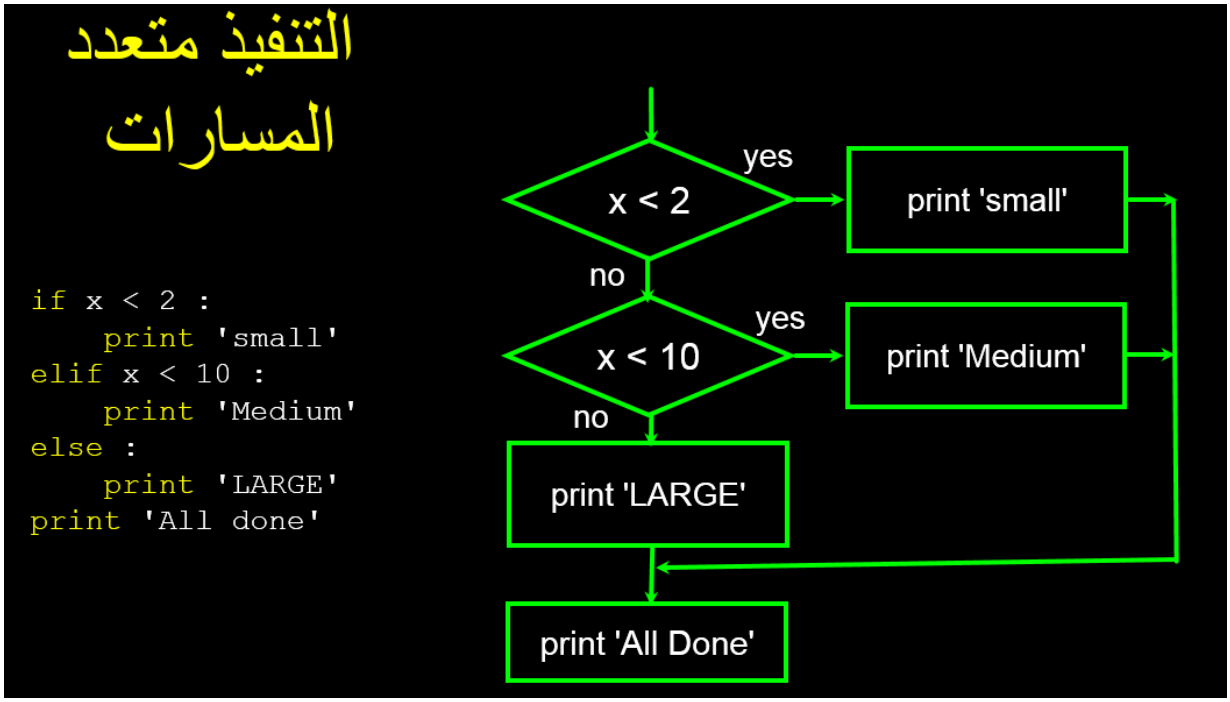
كما في الصورة السابقة نرى ان عبارة (if) تستخدم بشكل متشعب (if بداخل if) وتسمى باصطلاح البرمجة (nested if) وتعني انه ان كان الشرط الأول صحيح فقم بالدخول الى البلوك الخاص به لتجد شرطاً اخر، فأن كان هذا الشرط صحيح ايضاً يقوم المفسر بالدخول الى داخل البلوك الخاص به والا فلا. وهكذا؟



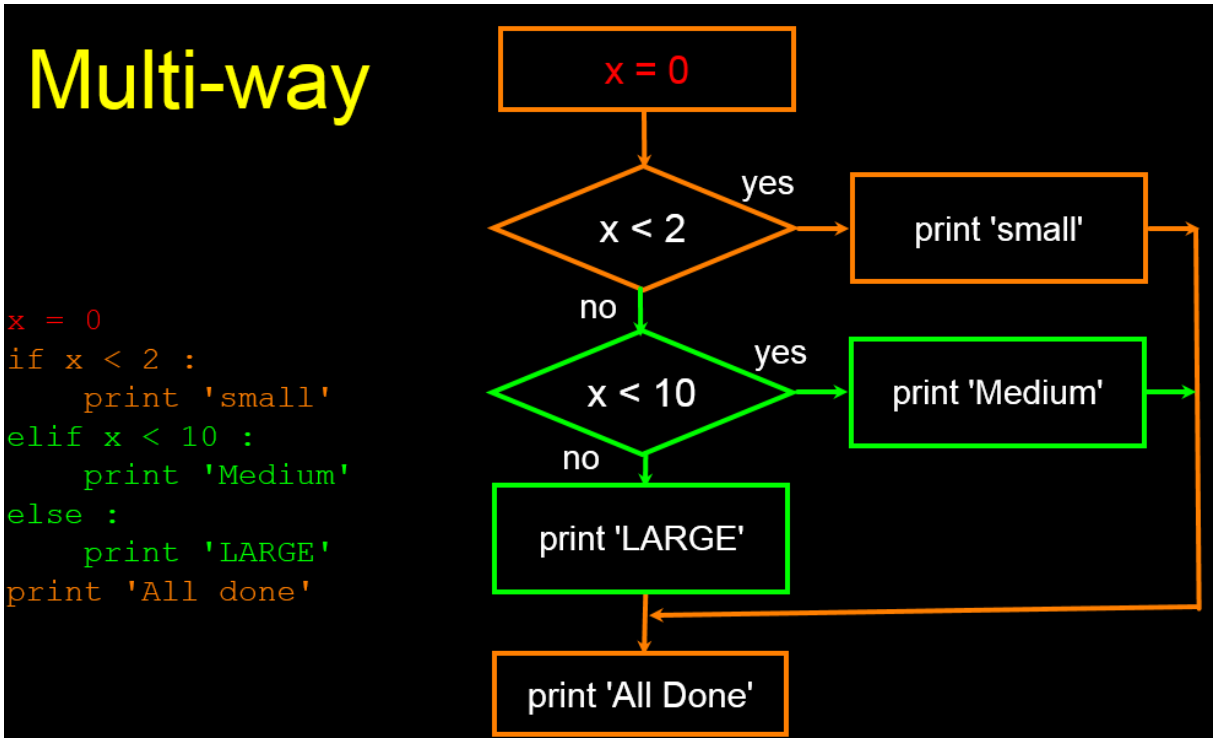
يمكن في هذه الحالات استخدام (else) او (elif = else if) وبحسب الحاجة وكما في ادناه:



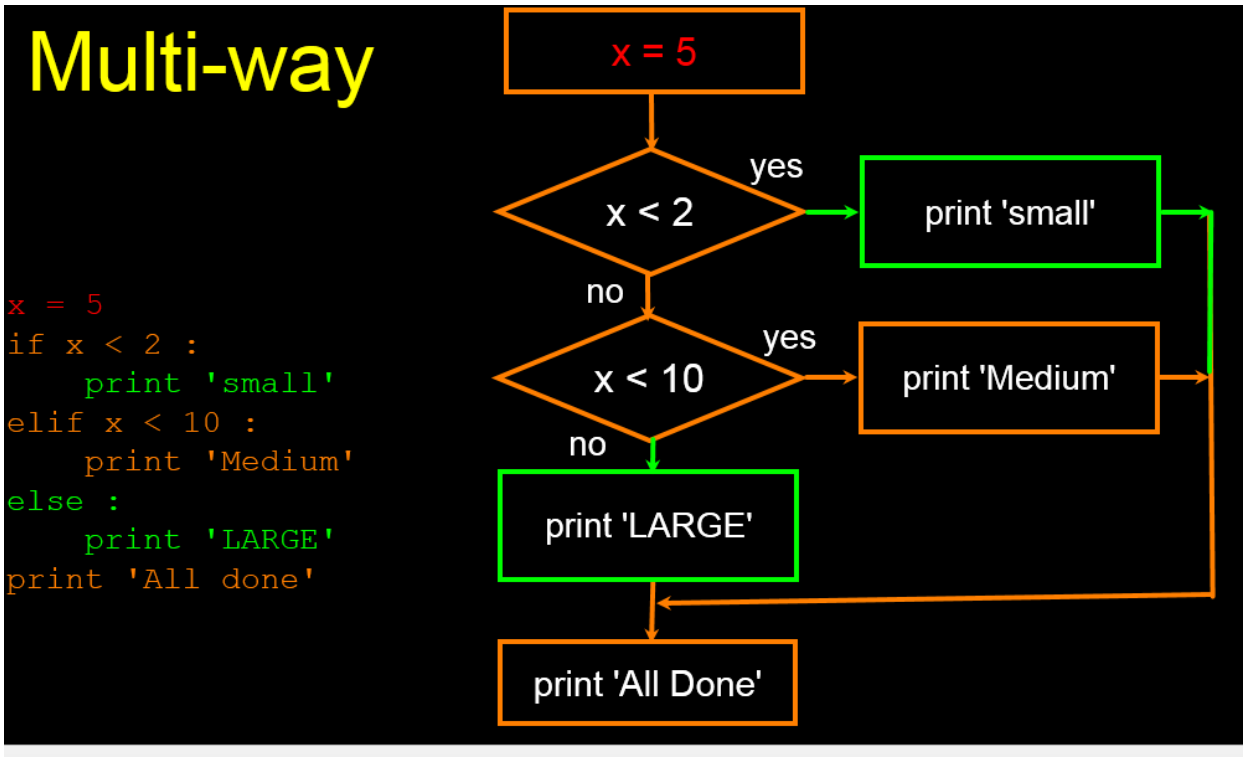
ويمكن ان يكون التنفيذ متعدد المسارات وكما في ادناه:



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

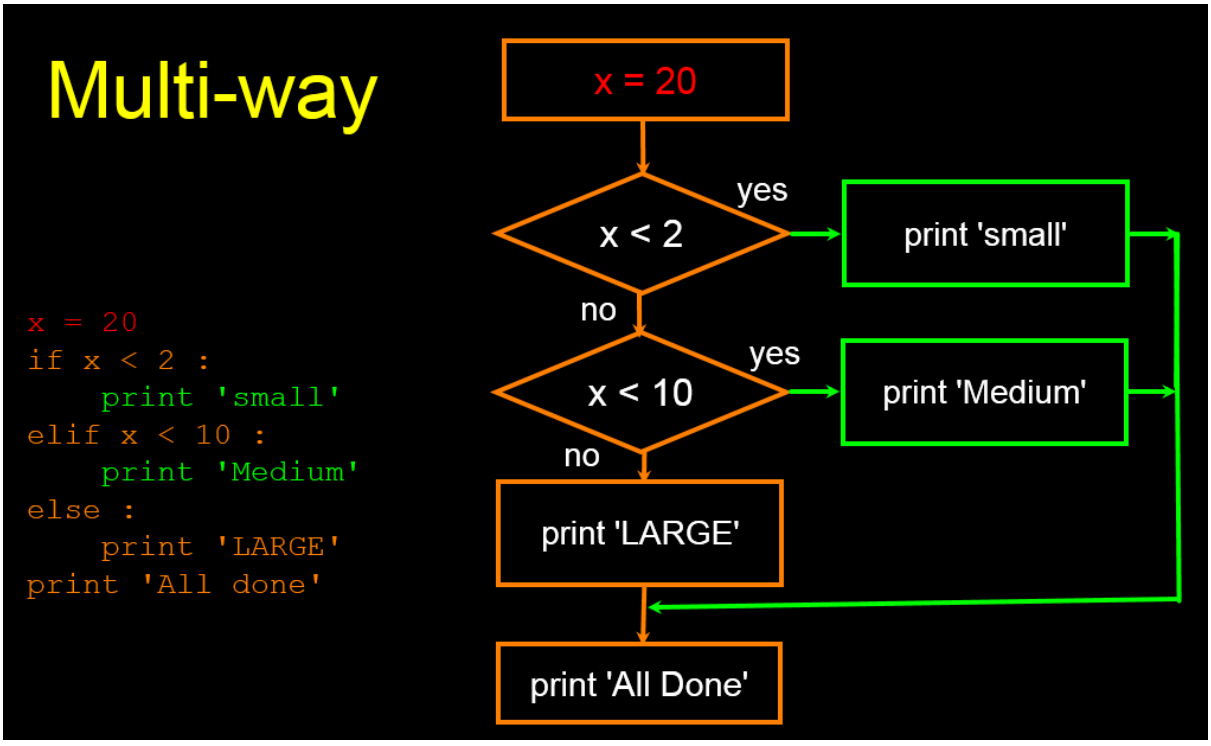


حيث ينفذ الجزء البرتقالي فقط لأن قيمة (x) تساوي صفر.



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:

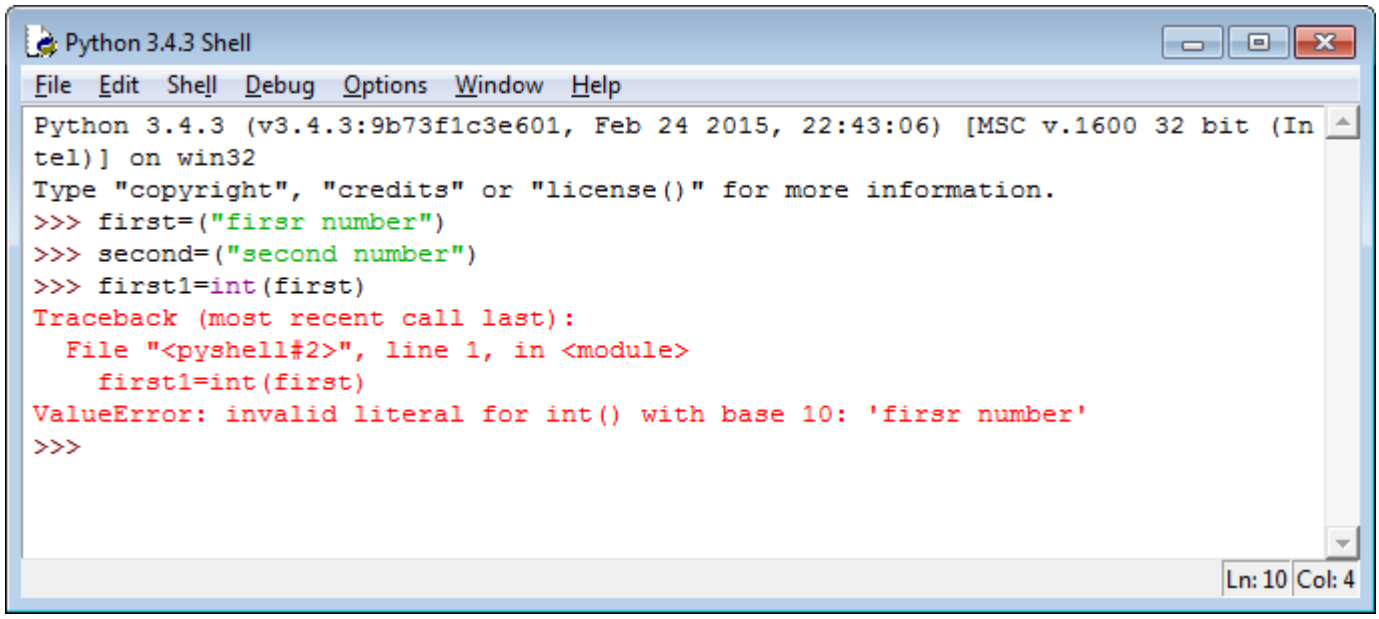
Try

هنا نضع الكود الخطر او المحتمل ان يسبب فشل تنفيذ البرنامج

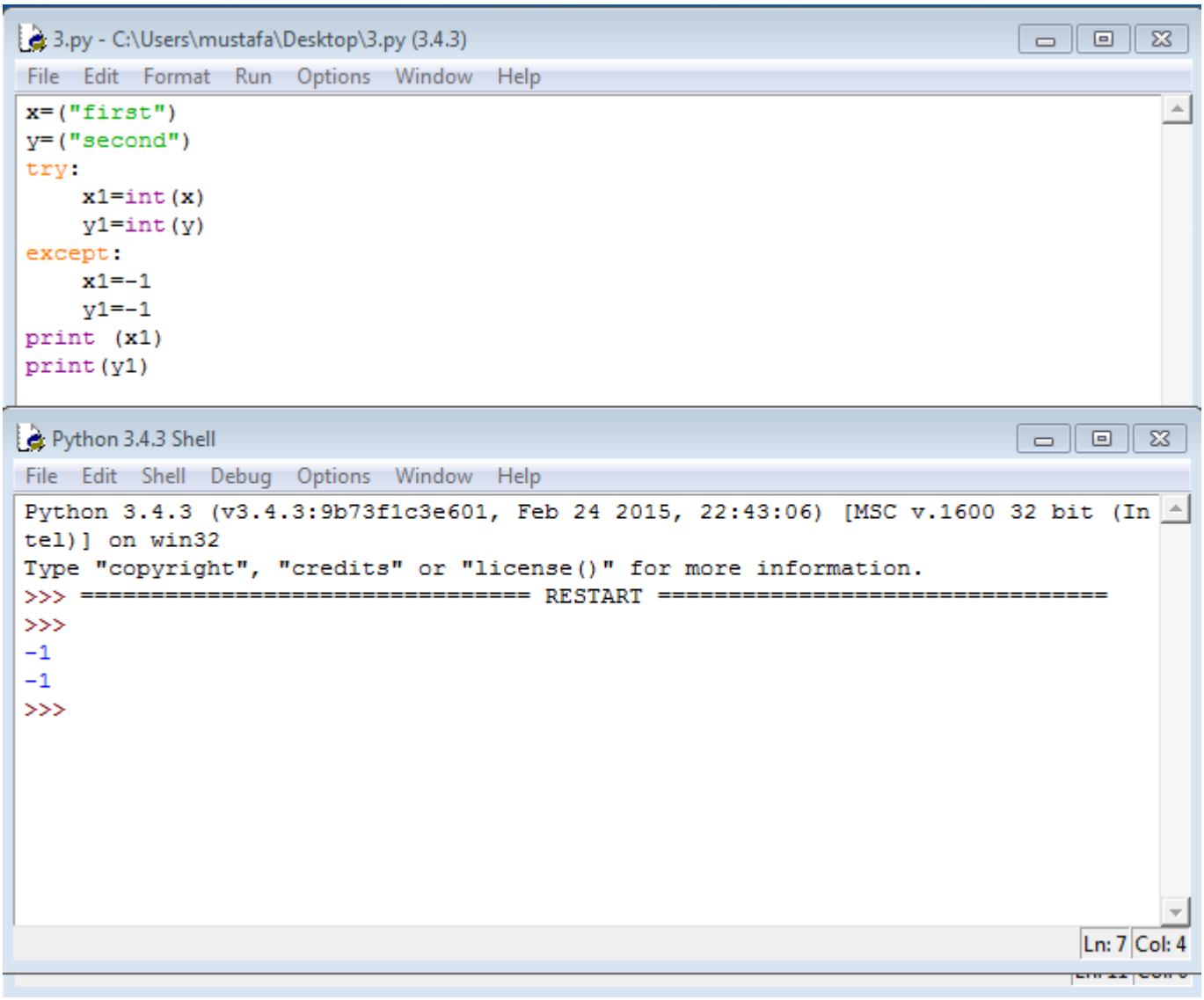
Except

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:



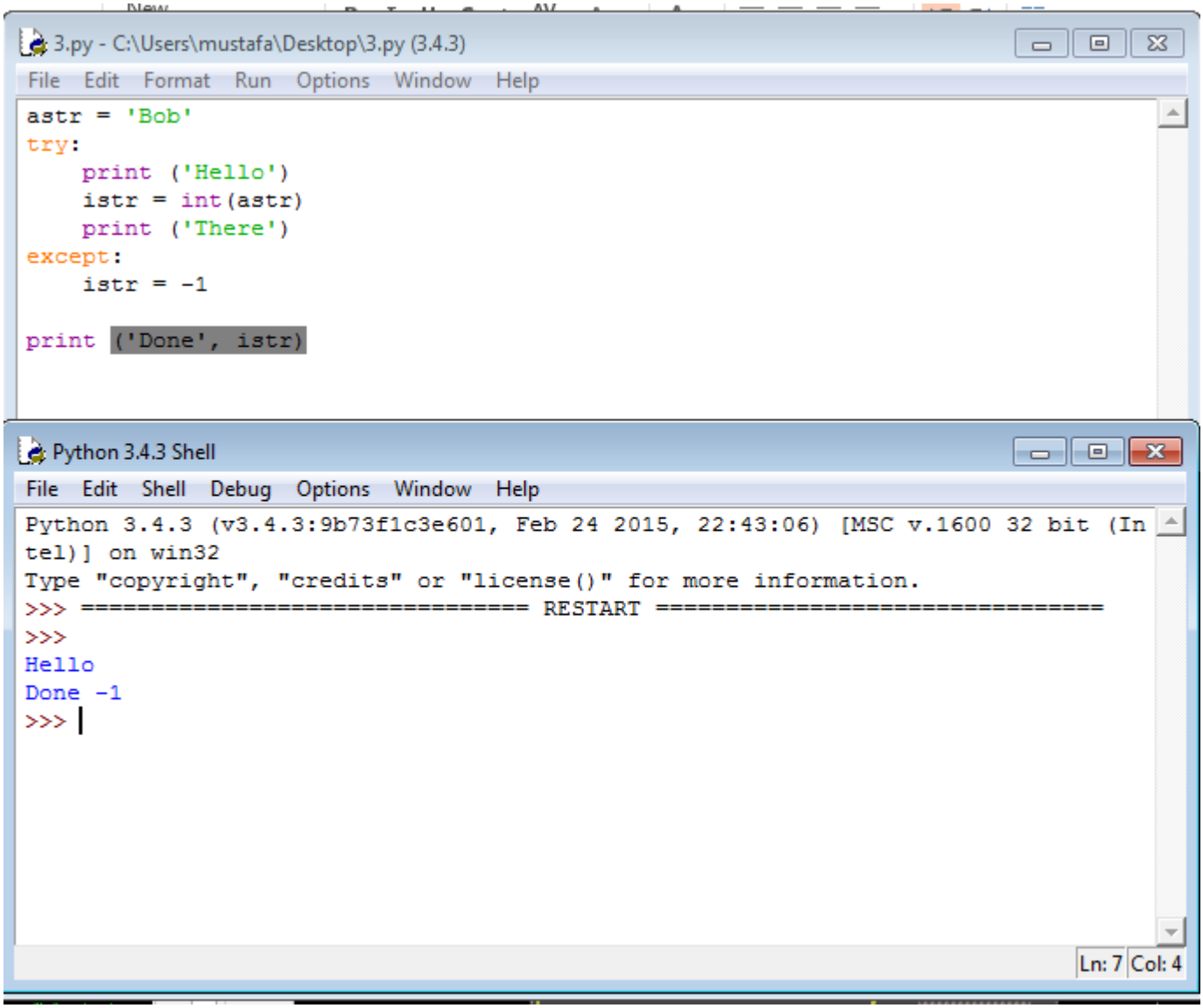
وهنا نرى انه من الطبيعي ان تظهر هذه الرسالة لأننا حاولنا تحويل قيمة رمزية الى عدد صحيح وهو شيء غير مقبول برمجياً. ولتجنب هذه المشكلة نقوم بالتالي:



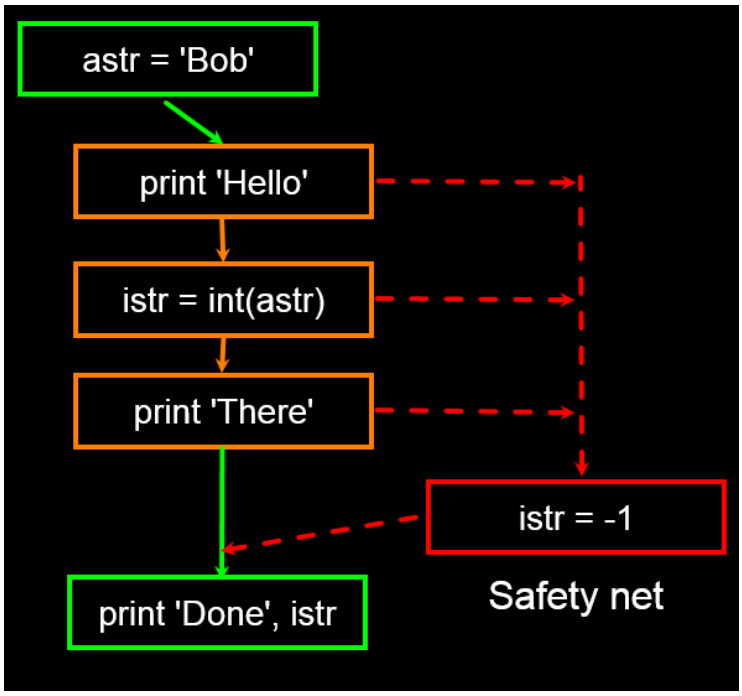
وهنا قمنا بأخبار المفسر بالقيام بالتالي:

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.

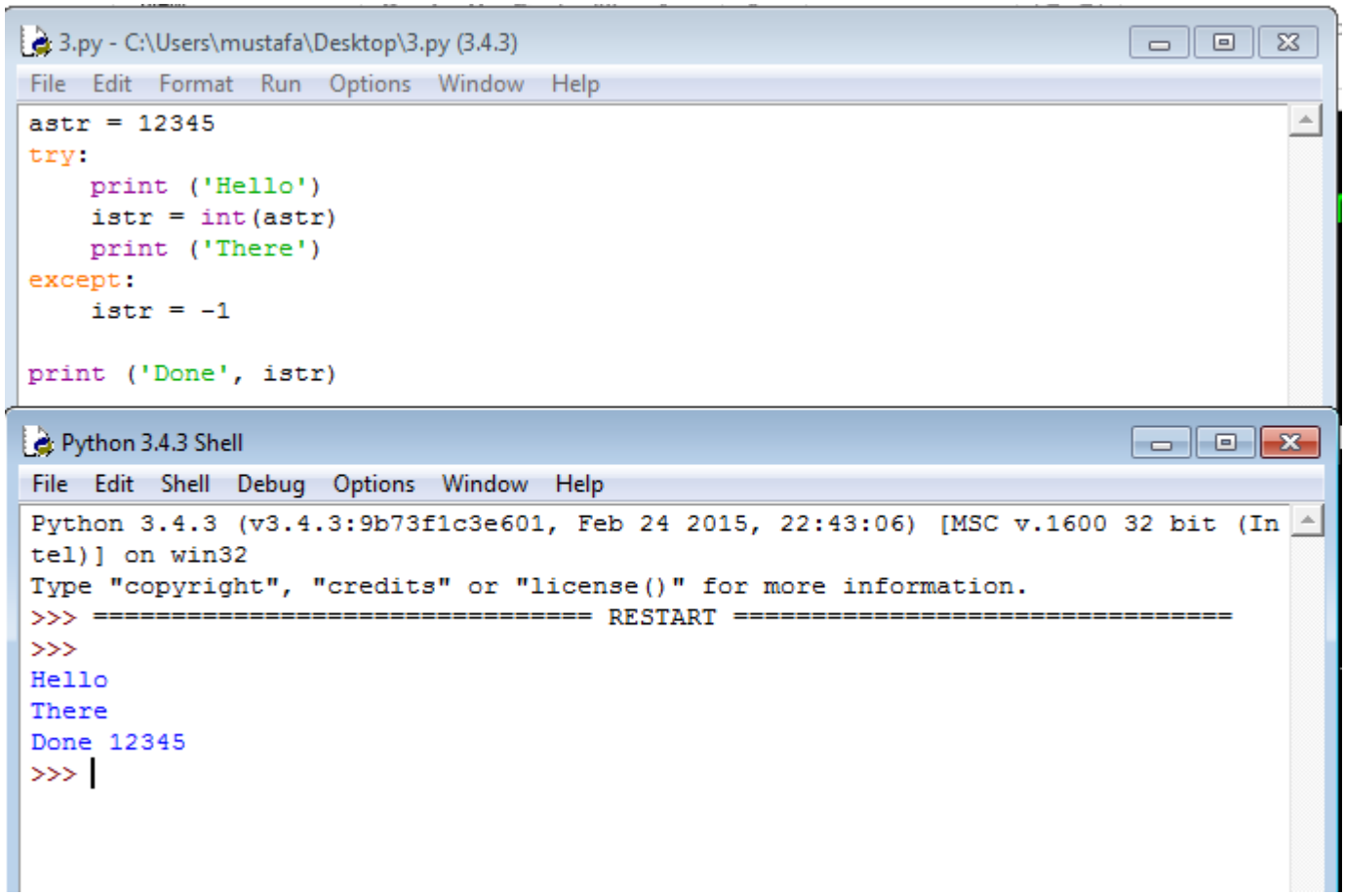
مثال اخر على التعامل مع عبارة المحاولة والاستثناء:



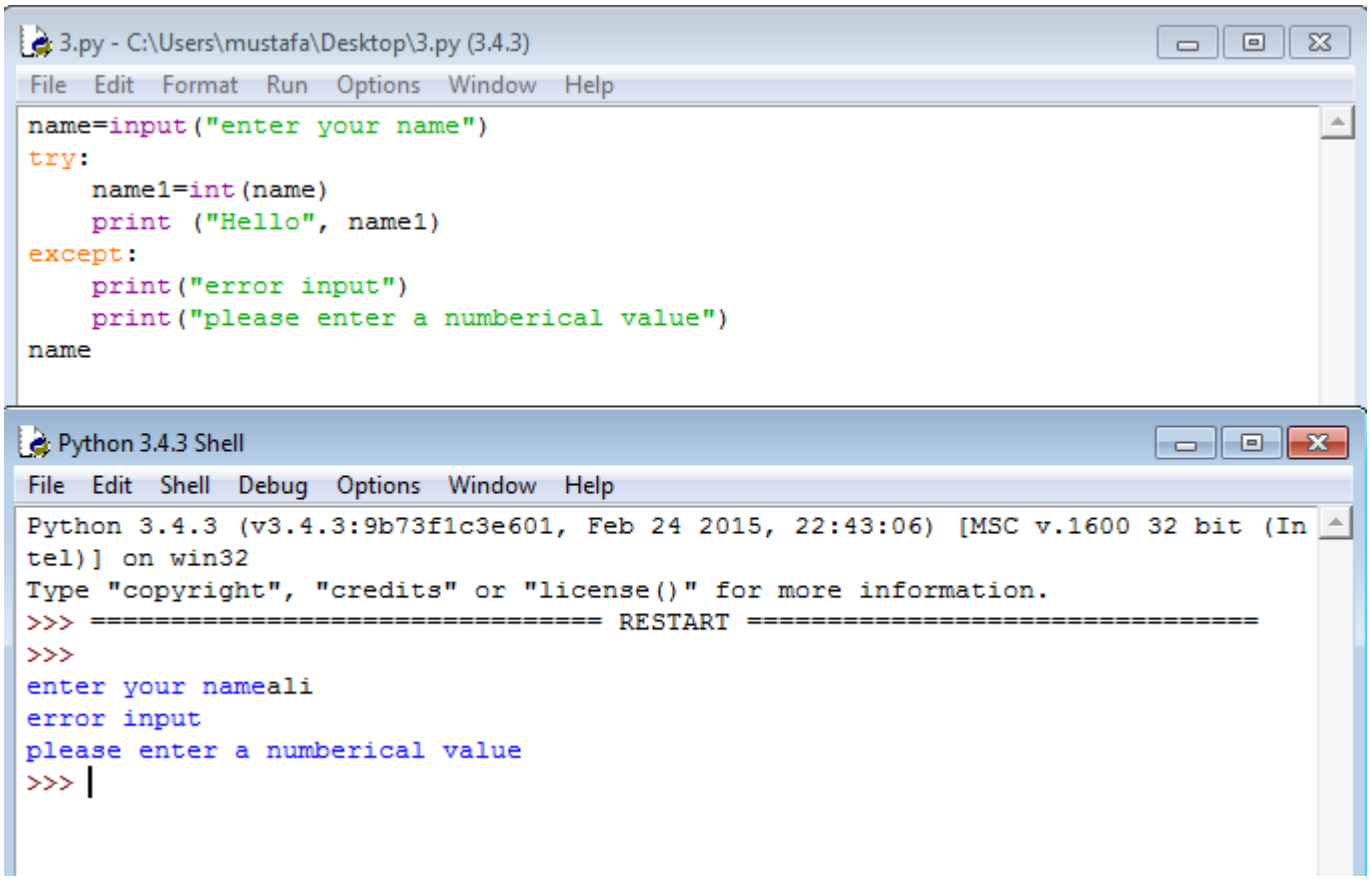
والتوضيح لتسلسل التنفيذ كما في ادناه:



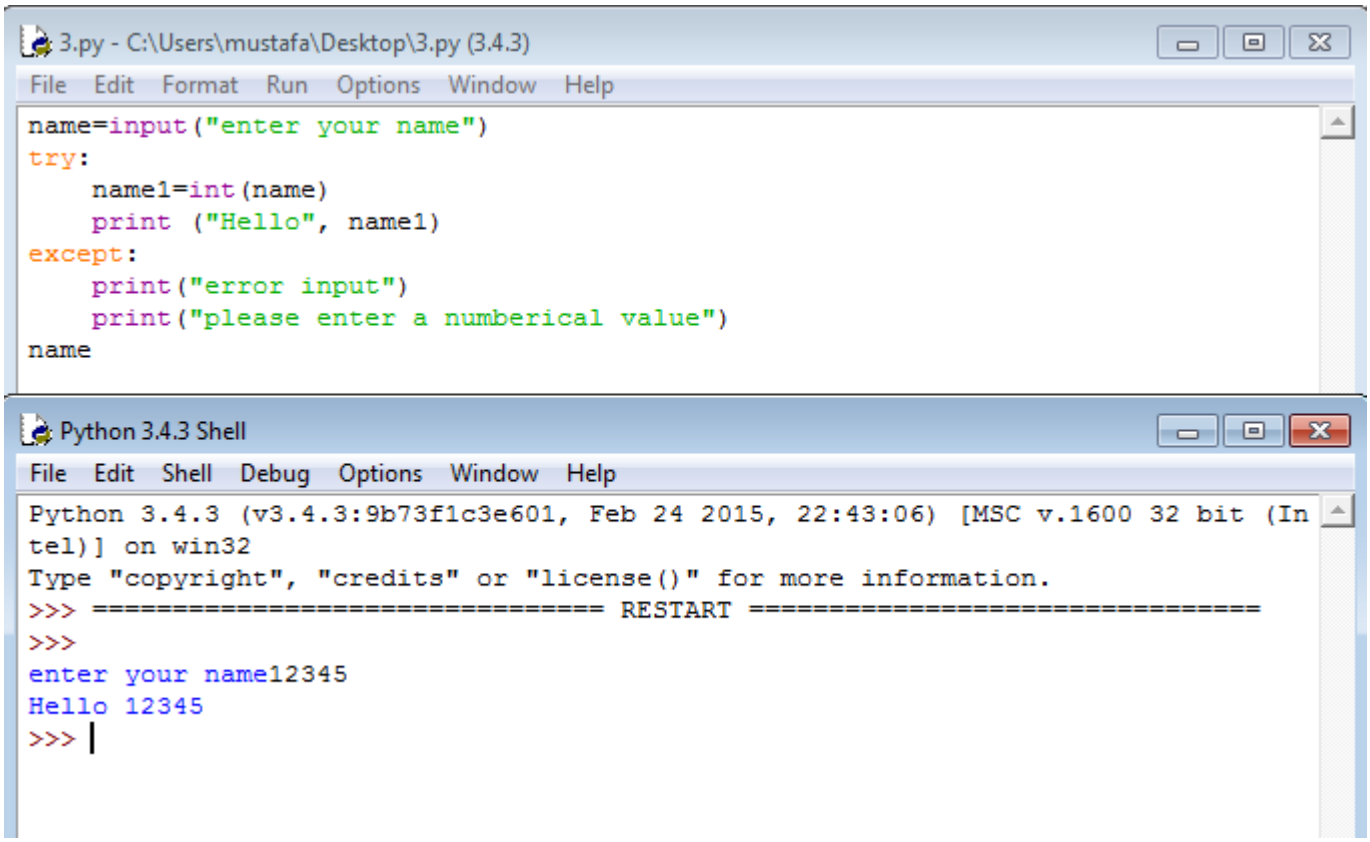
وهذه نتيجة التنفيذ في حالة تبديل قيمة المتغير (astr) وجعلها قيمة عددية حيث يتم تنفيذ ما بداخل عبارة (try) وكما في ادناه:



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:



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



وفي نهاية درس اليوم ارفق لكم احبتي بعض الأمثلة 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.

```
hrs = raw_input("Enter Hours:")
h = float(hrs)
rate=raw_input("enter the rate")
r=float(rate)
if h > 40:
    pay=(40*r) + (h-40)*(r*1.5)
    print pay
else:
    pay=h*r
    print pay

```

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.

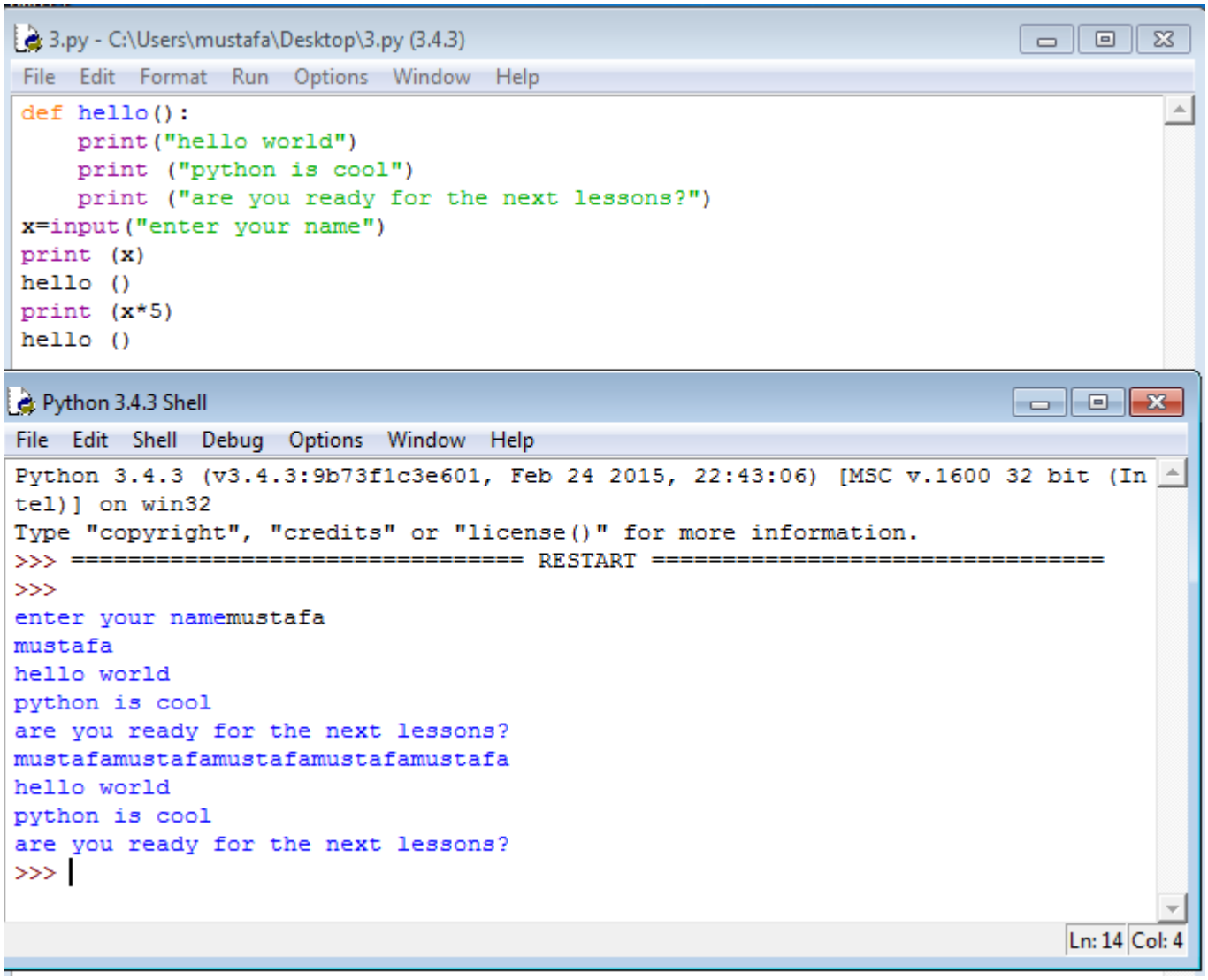
```
score=raw_input("enter a number between 0.0 and 1.0")
try:
    s=float(score)
except:
    print "not a number"
if s>1.0:
    print "error"
    quit()
if s>=0.9:
    print "A"
elif s>=0.8:
    print "B"
elif s>=0.7:
    print "C"
elif s>=0.6:
    print "D"
else:
    print "F"

```

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:



هنا نلاحظ الأمور التالية:

١- لتعريف دالة معينة نسبق اسمها بالعبارة المفتاحية (def) وهي مختصر كلمة (Define) بمعني 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) ليدل على اننا الان نكتب بداخل جسم الدالة.

٤- يمكن ان تحتوي الدالة على أي عدد من العبارات البرمجية الحسابية او المنطقية او التكرارية او الشرطية.

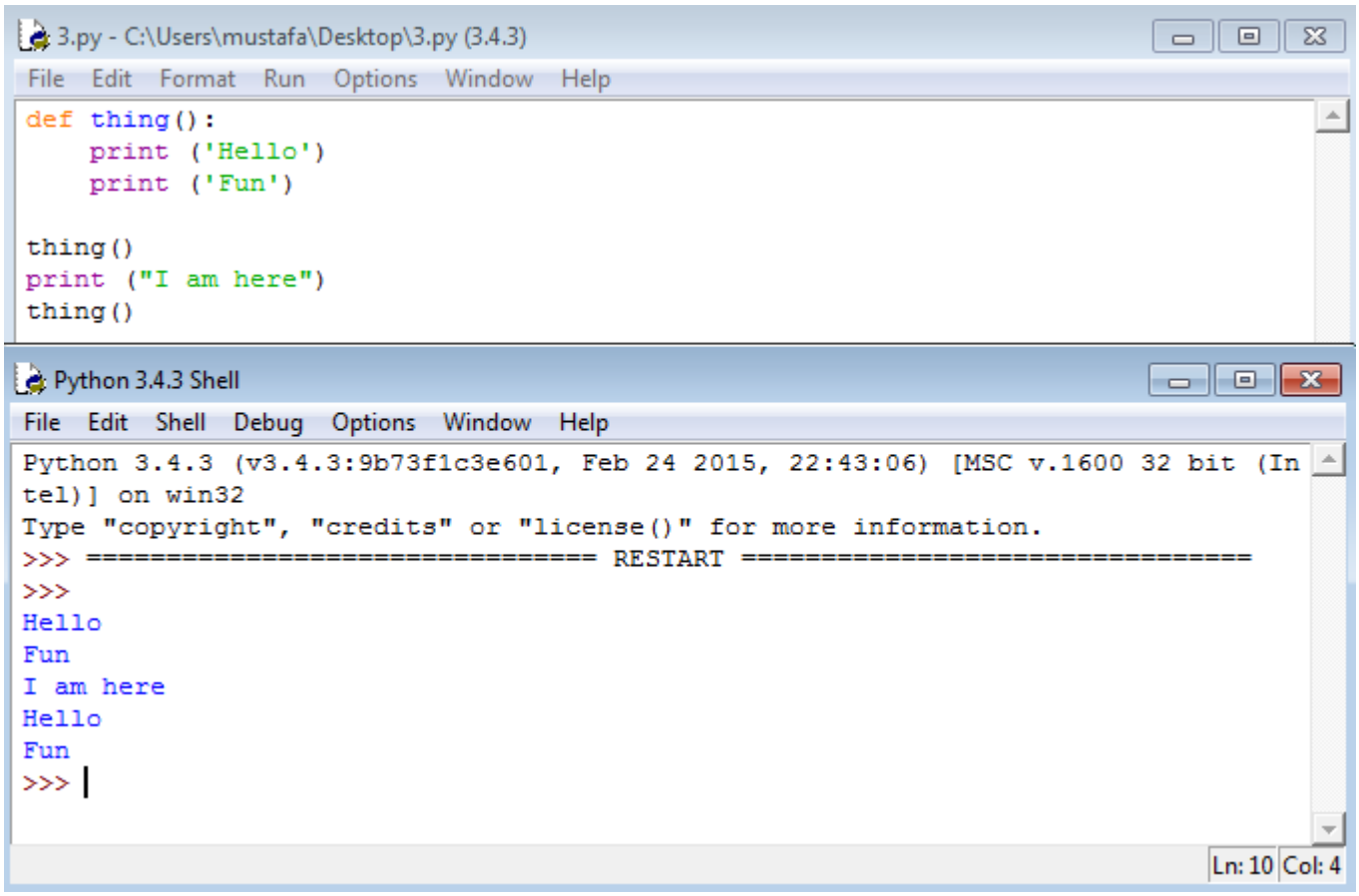
٥- لتعريف المفسر ان جسم الدالة قط انتهى نقوم فقط بالنزول الى سطر جديد والرجوع الى بدايته أي الغاء المسافات التلقائية ونحن بذلك نقوم بأخبار المفسر ان جسم الدالة قد انتهى.

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).

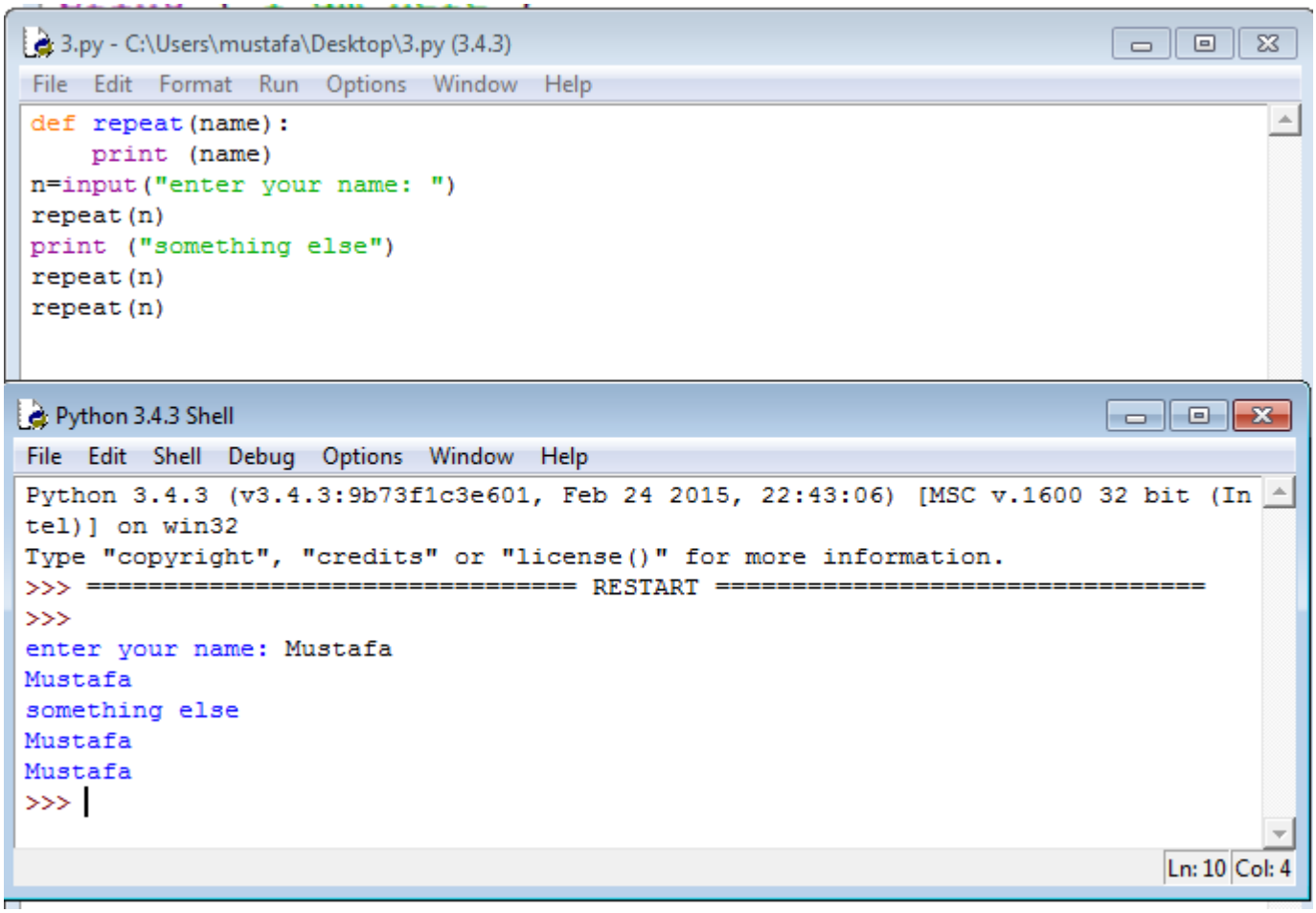
٧- يتم تنفيذ ما بداخل الدالة عند استدعائها وبأي عدد من المرات داخل البرنامج وهذه هي الميزة الرئيسية والفائدة الكبرى للدوال: تجنب تكرار كتابة نفس الكود أكثر من مرة.

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:



وهذا المثال ايضاً:



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 كالاتي:

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).

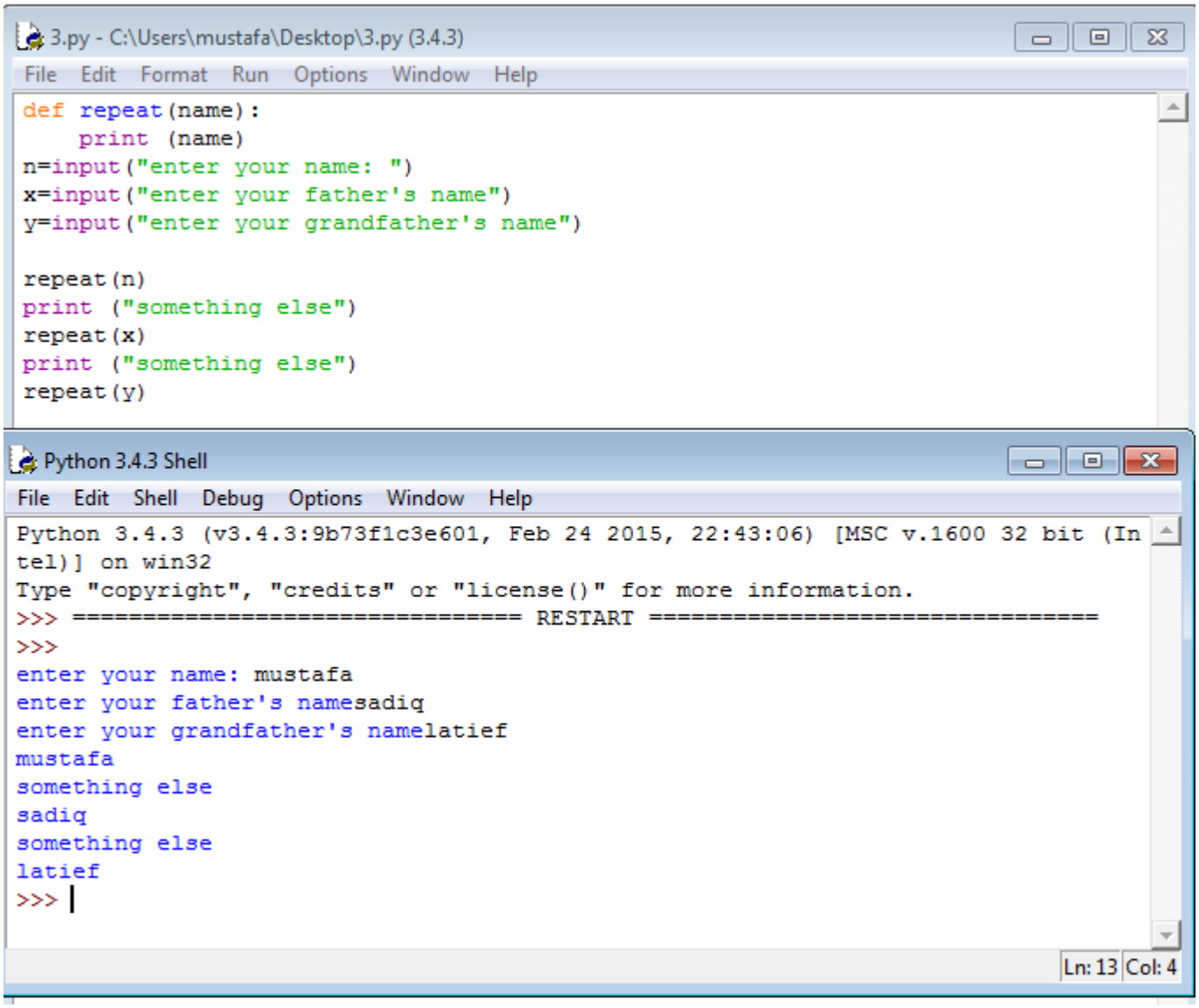
٤- والان نقوم باستدعاء دالة (repeat) للمتغير (n) أي اننا نقول للمفسر: خذ قيمة المتغير (n) وضعها بدل متغير الدالة المسمى (name) ثم طبق ما بداخل الدالة لهذا المتغير (n).

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.

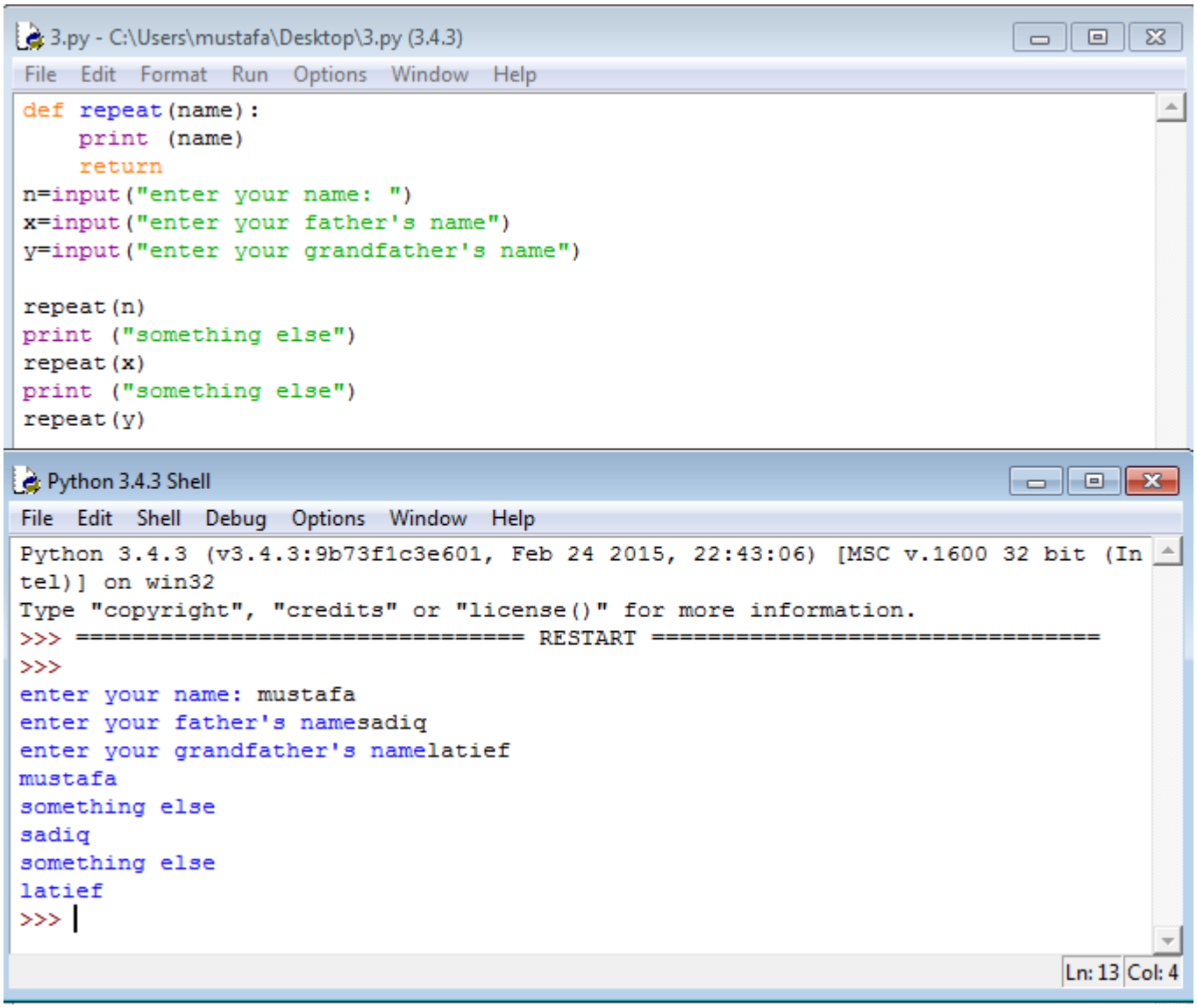
ولكن ماذا يحصل لو أردنا تكرار تنفيذ الدالة لمتغيرات مختلفة؟

دعونا نرى الان:

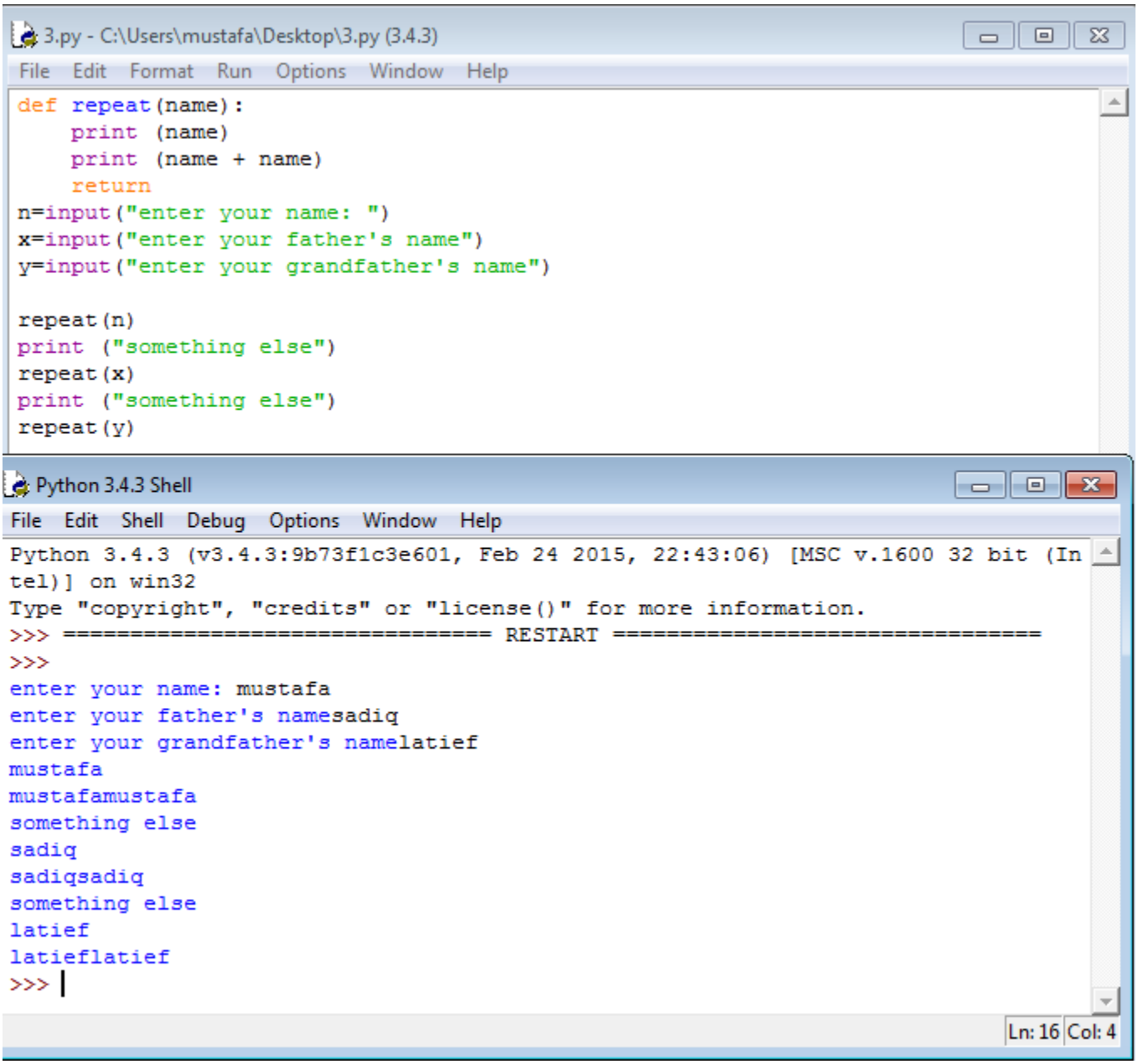


هنا نرى ميزة أخرى وفائدة أخرى من فوائد الدوال وهي اننا قمنا باستدعاء الدالة لأكثر من متغير وبعدة أسماء وقيم وهي (n,x,y) وفي كل مرة يتم استدعاء الدالة فيها يتم طباعة او تنفيذ الدالة بشكل مختلف.

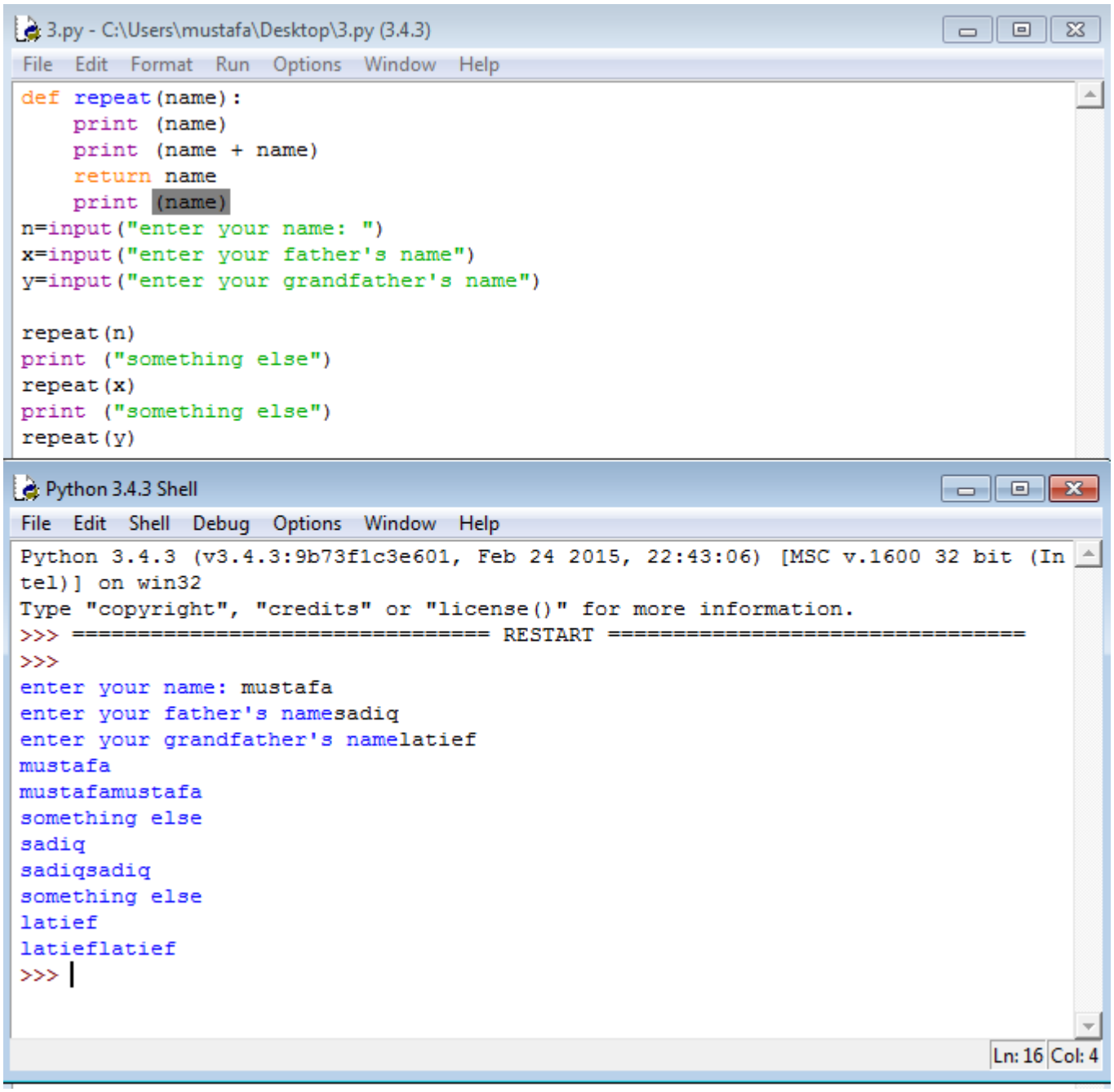
والان نأتي الى توضيح أحد مميزات الدوال الأخرى وهي استخدام دالة (Return) لإرجاع قيمة من الدالة الى البرنامج الرئيسي وكما في المثال ادناه:



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:

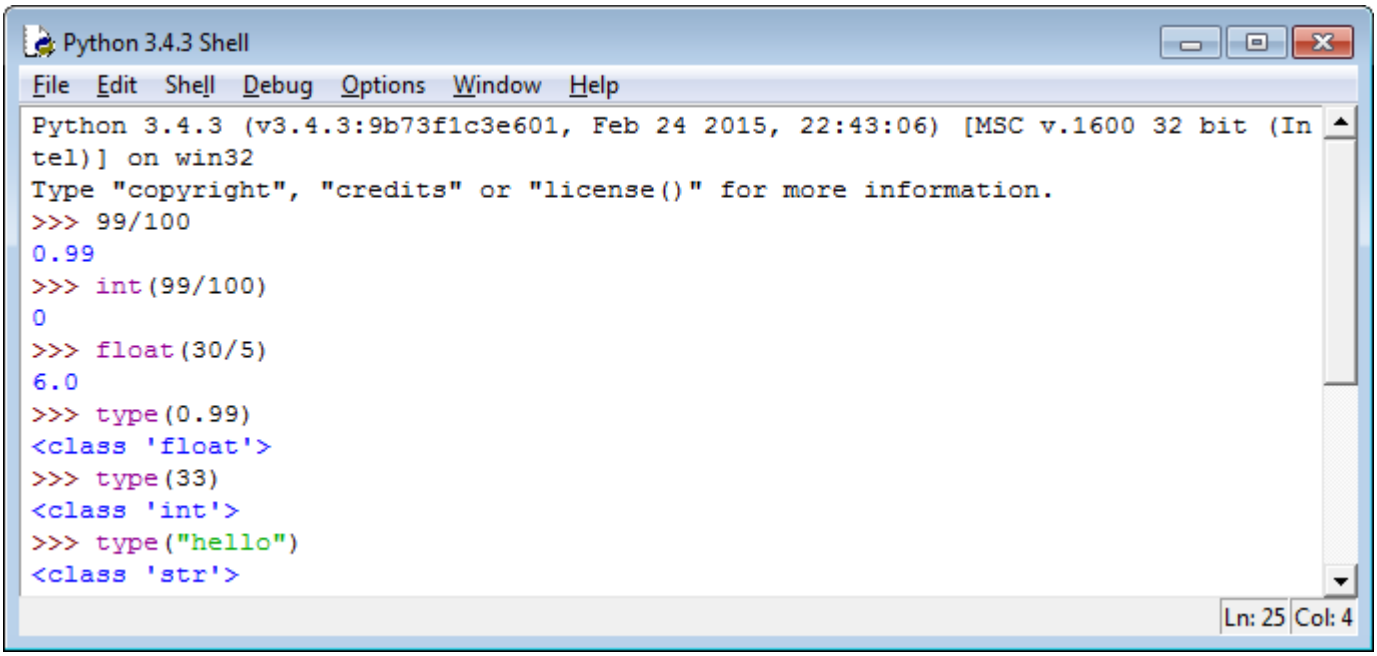


ولاحظ الفرق في التنفيذ الان:



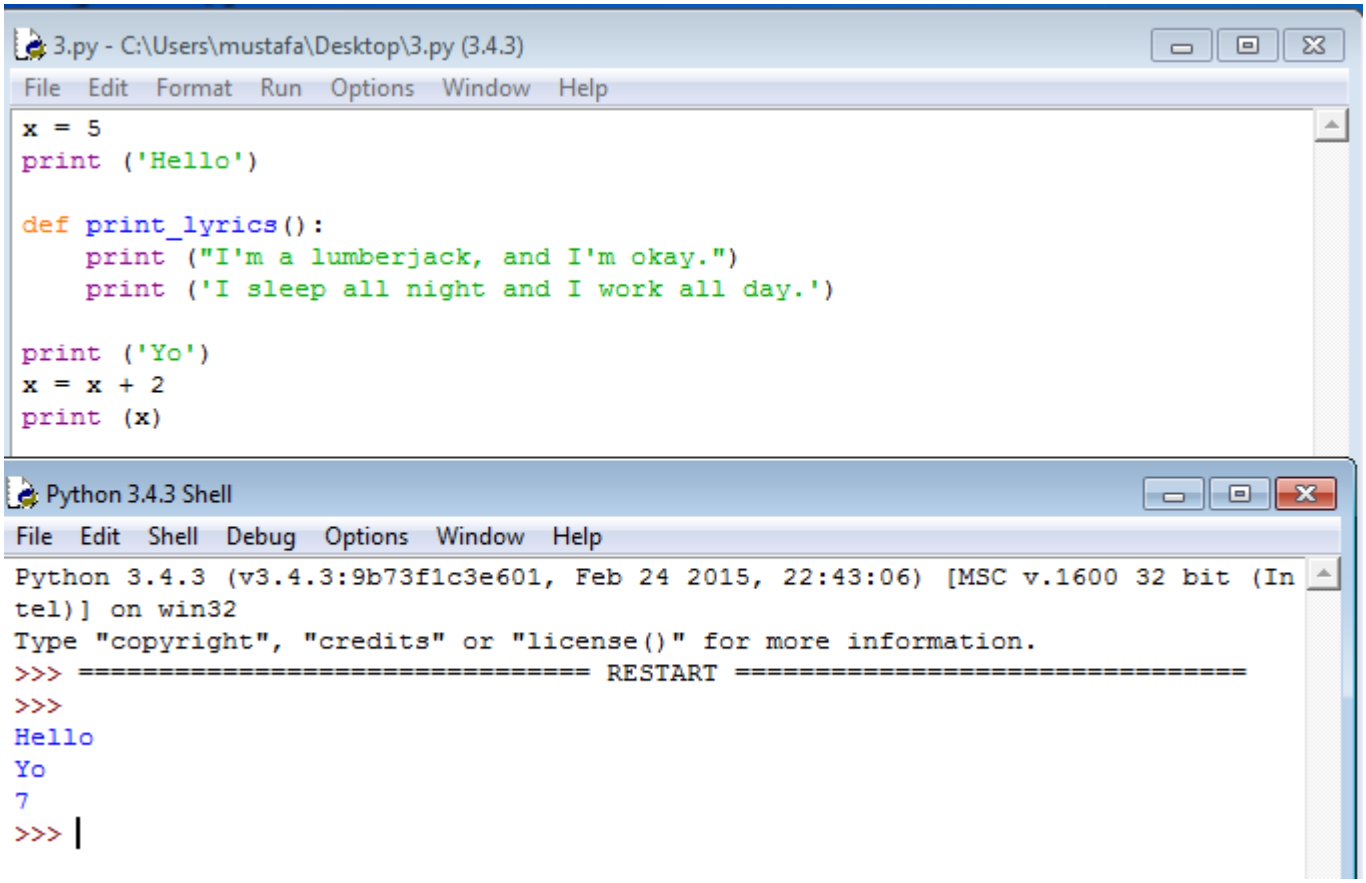
هنا تم التنفيذ بنفس الطريقة تماماً كما في المثال السابق رغم وجود سطر إضافي للطباعة بعد عبارة (Return) داخل الدالة وهذه من فوائد عبارة الارجاع (return) حيث انها تؤشر نهاية الدالة وتخبر المفسر ان الدالة قد انتهت وان كل ما ورائها من العبارات المزحفة (indented) ليست ذات أهمية بل ان المفسر يهملها مباشرة.

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:

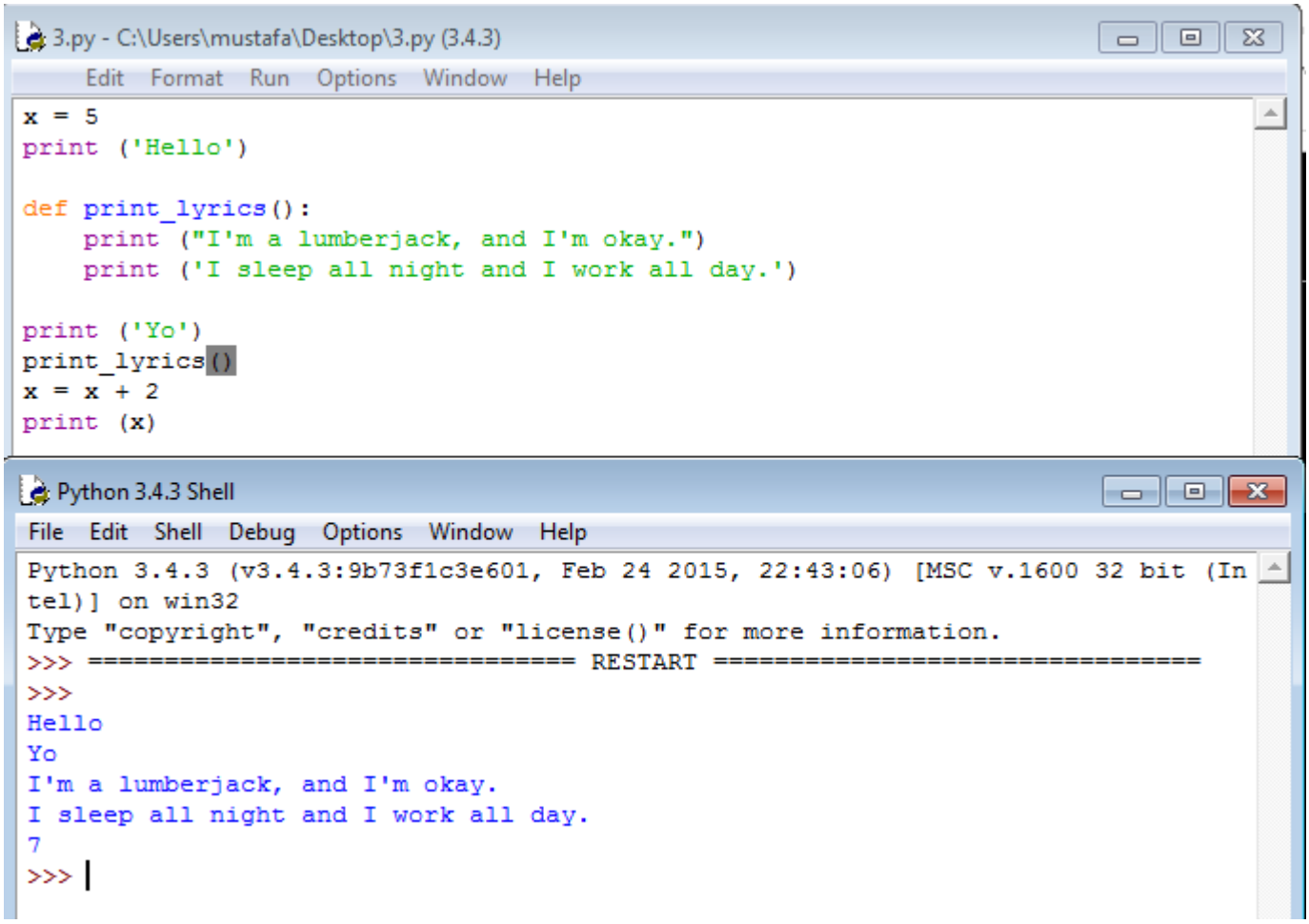


حيث تعتبر هذه الدوال هي من الدوال الجاهزة المخزونة بداخل لغة البرمجة لتحديد نوع المتغيرات والثوابت والتحويل بينها.

المزيد من الأمثلة عن الدوال:

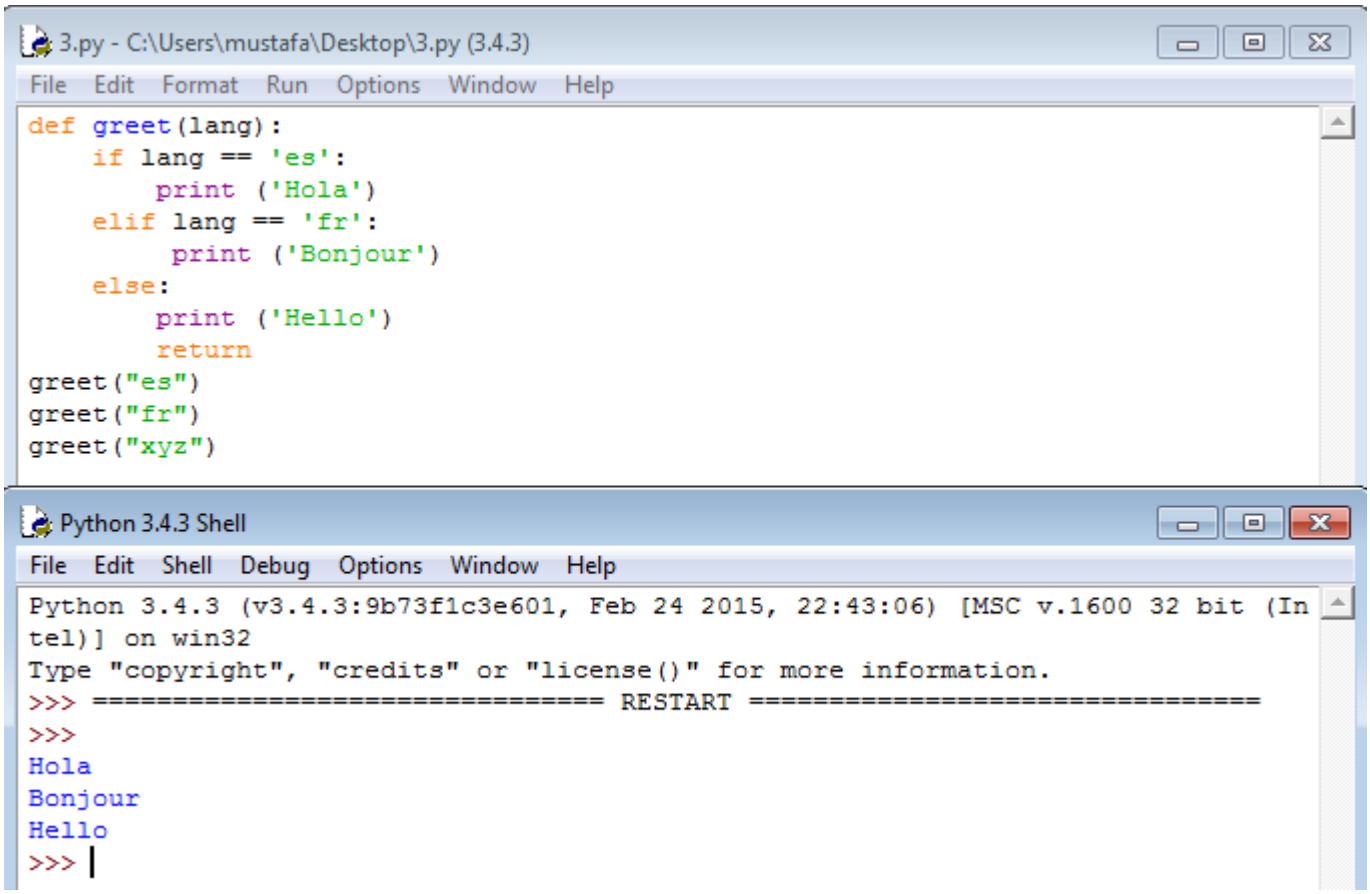


المثال أعلاه لدالة بدون استدعاء.

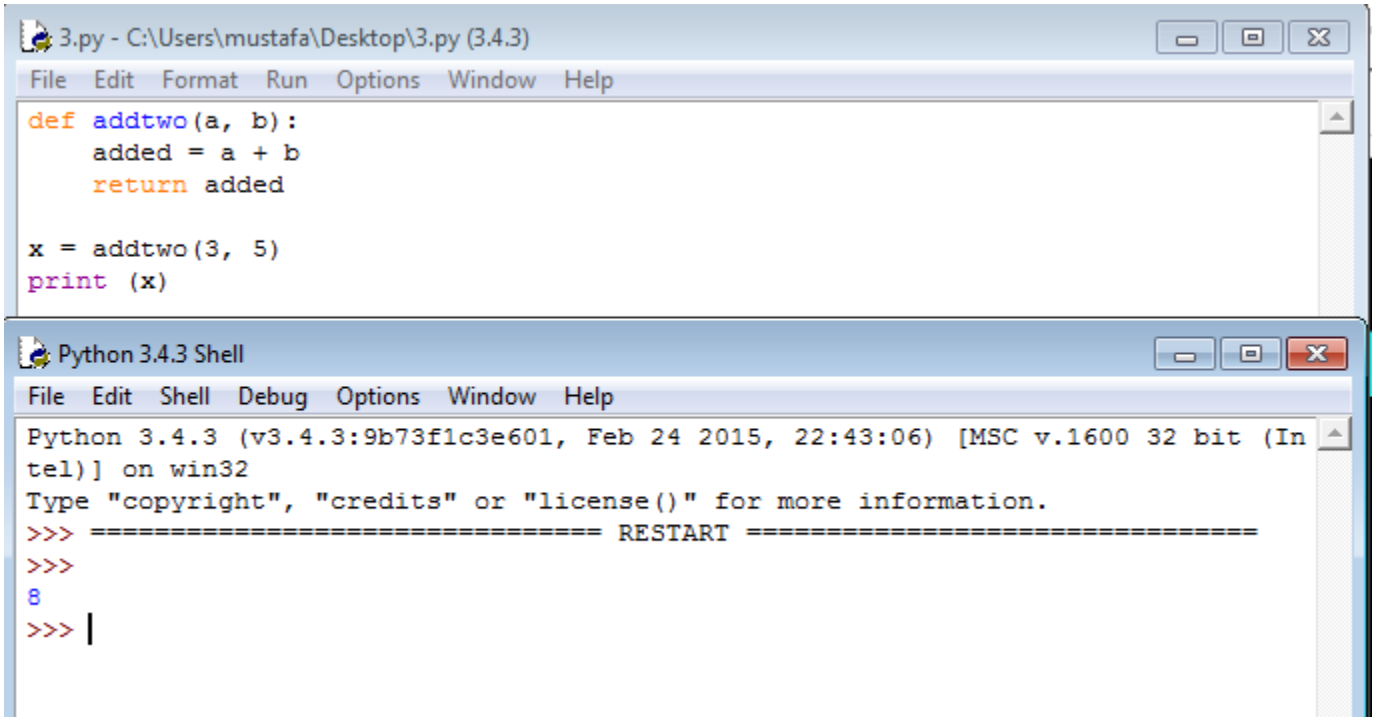


هنا تم استدعاء الدالة فظهرت نتيجة تنفيذها.

والان مثال عن كيفية توظيف الشروط بداخل الدوال:



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



- **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 لتنظم 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 **computepay()** 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.

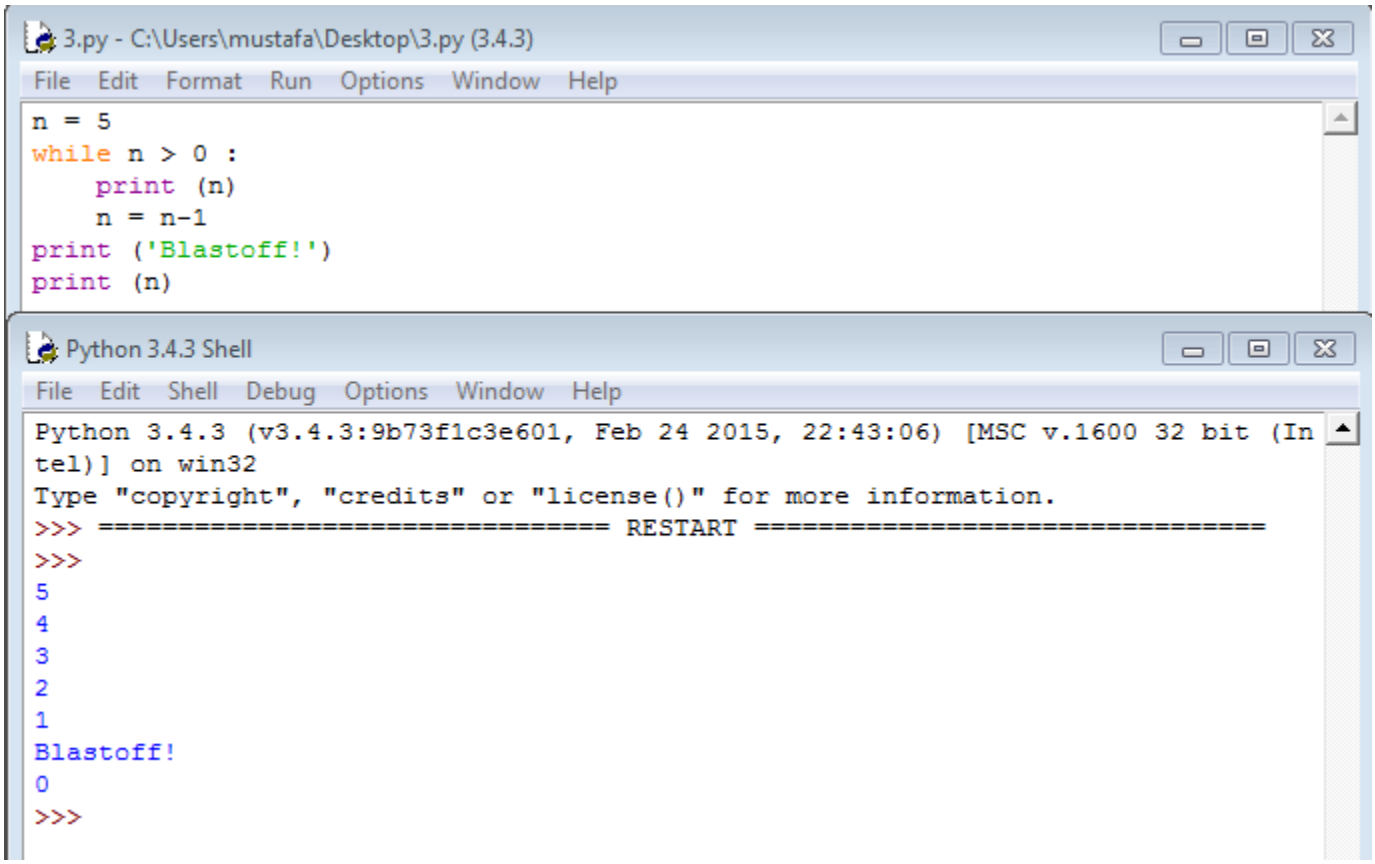
Check Code | Reset Code | Exit | Grade updated on server.

```
def computepay(h,r):
    if h>40:
        pay=(1.5*r*(h-40))+(40*r)
        return pay
    else:
        pay=r*h
        return pay

hrs = raw_input("Enter Hours:")
hours=float(hrs)
rat=raw_input("enter rate:")
rate=float(rat)
p = computepay(hours,rate)
print p
```

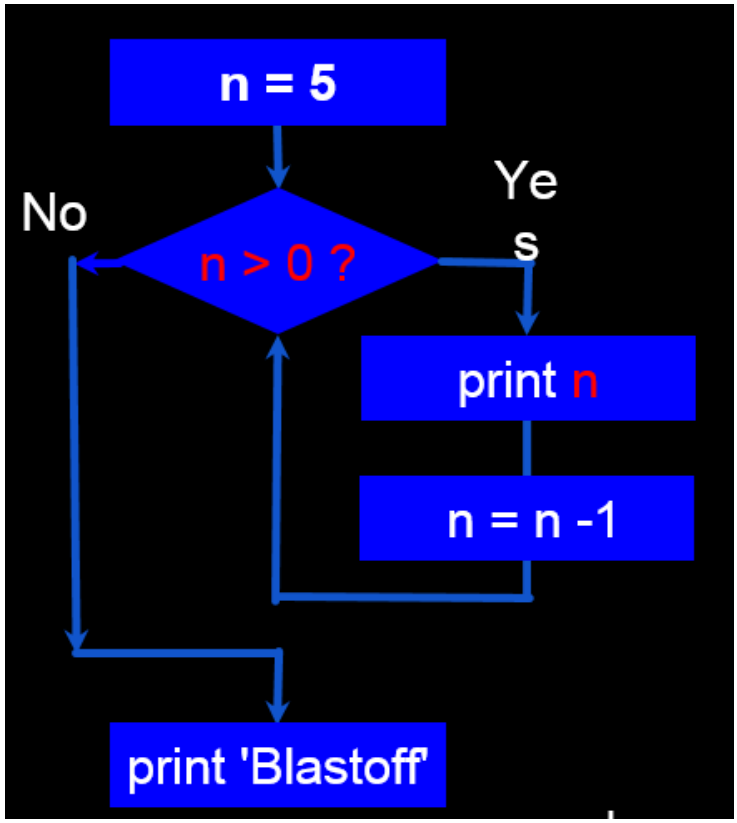
الدرس العاشر: الحلقات التكرارية (While)

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:



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:



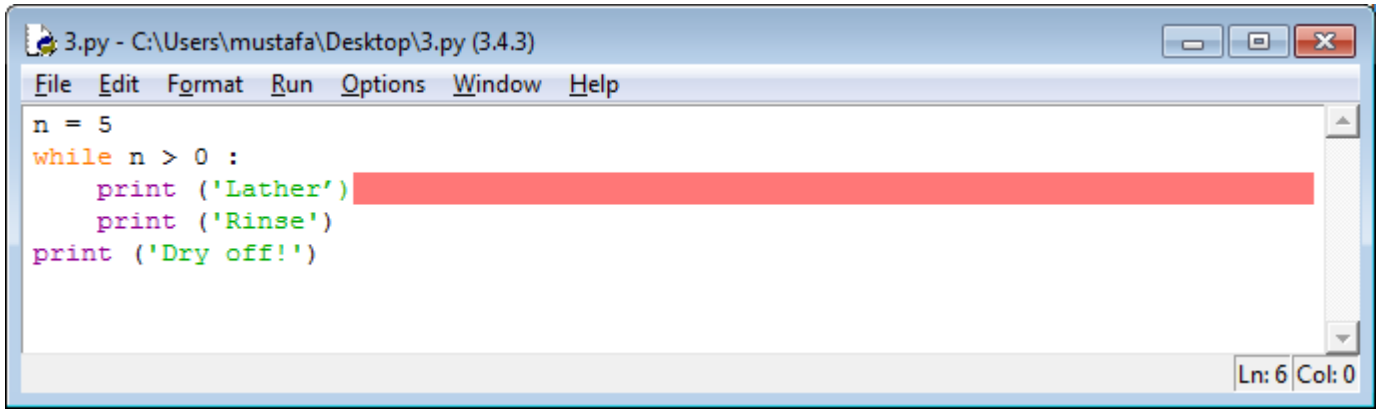
وهنا قمنا بتوضيح الية التنفيذ بشكل رسومي للتوضيح أكثر.

مشاكل التعامل مع عبارة (while):

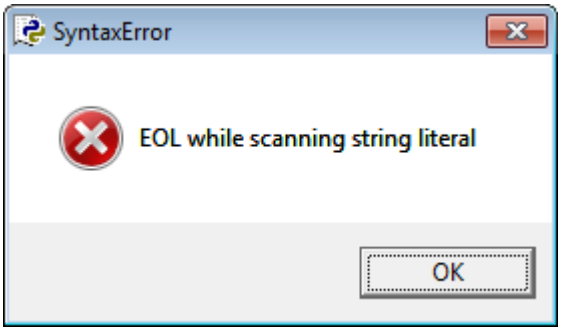
لكيلا يتم فهم العنوان هنا بشكل خاطئ، فأن عبارة while ممتازة في الكثير من الحالات ولا اشكال في استخدامها في البرمجة ولكن بشروط:

- **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:

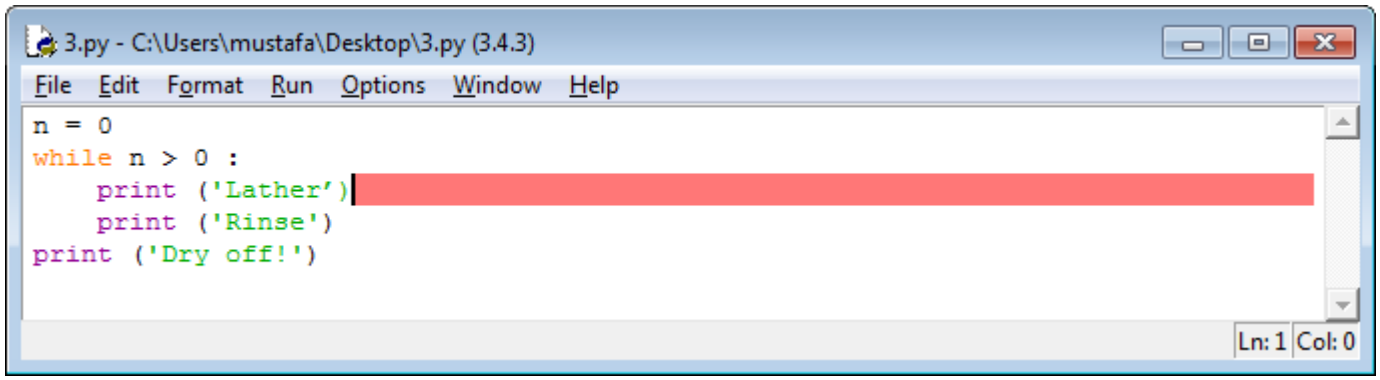


هنا نلاحظ ان شرط while سيبقى صحيحاً الى ما لا نهاية ولذا تظهر الرسالة التالية عند محاولة التنفيذ:



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.

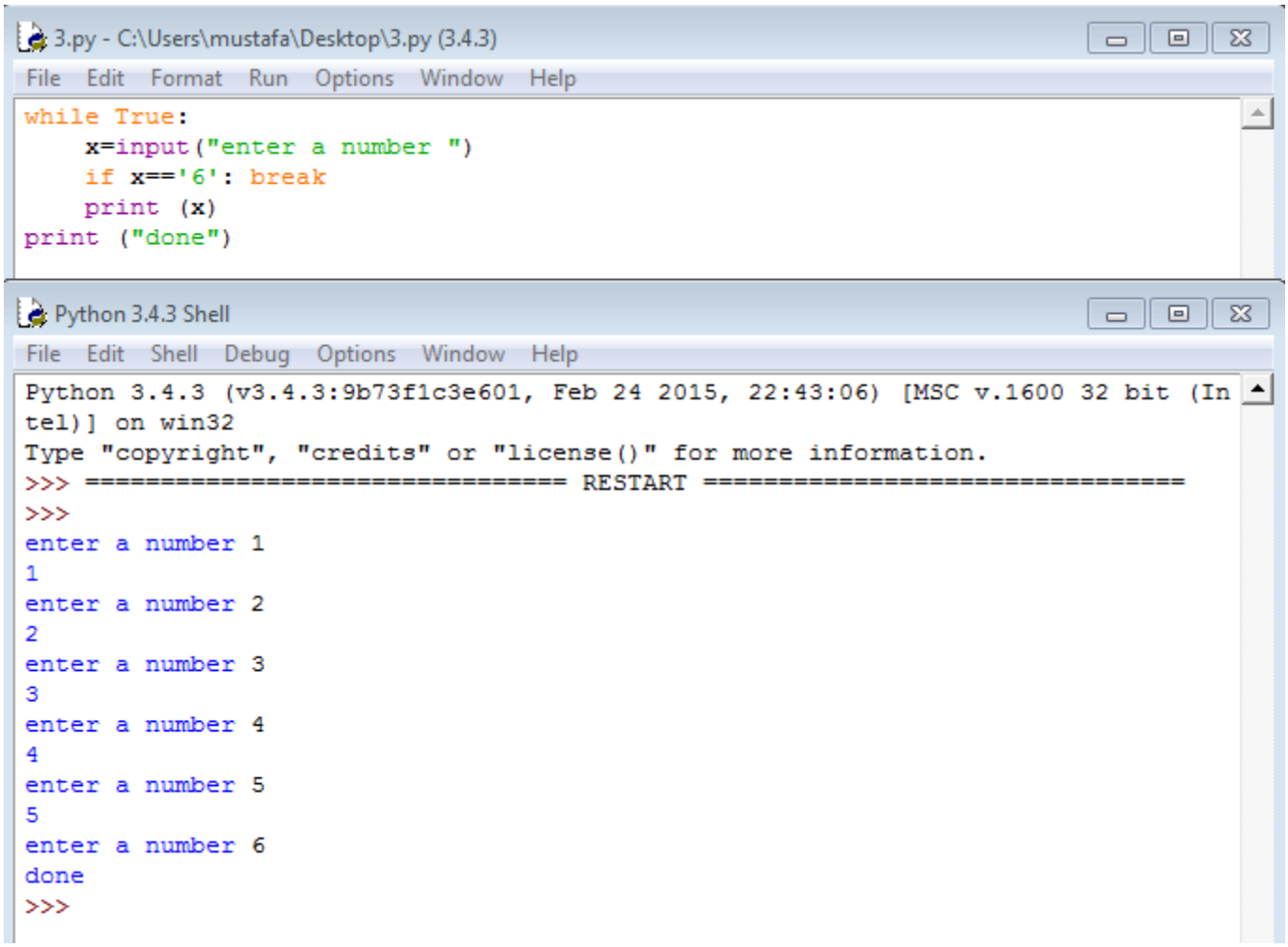
عبارة (while) بلا فائدة:



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:

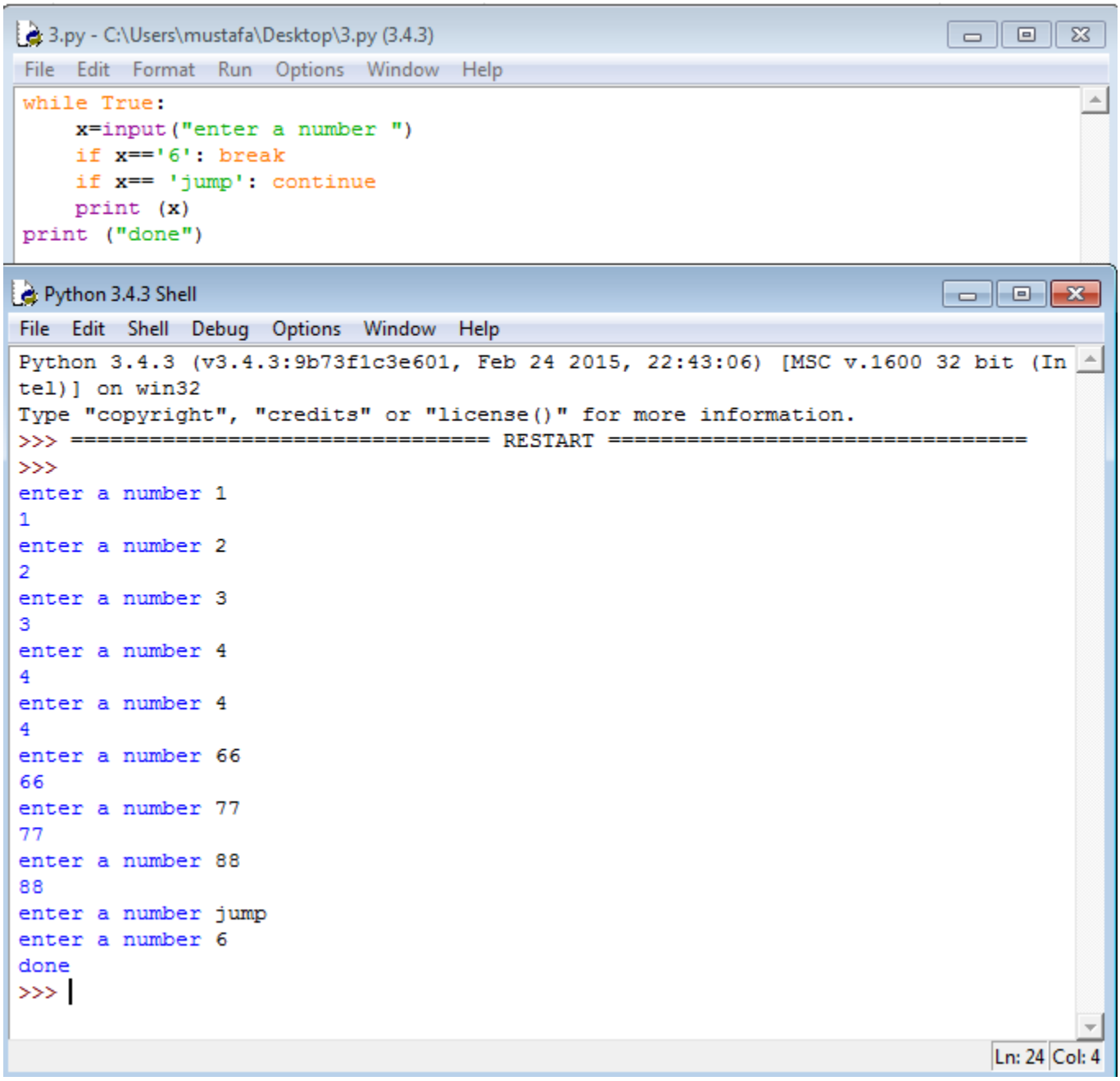
نحتاج بعض الأحيان الى إيقاف الحلقة التكرارية حين يتحقق شرط معين مثل وجود رقم معين كنا نبحث عنه او ادخال المستخدم لقيمة معينة كنا نبحث عنها وهنا نستخدم عبارة (break) والتي تقول للمفسر: قم بأنهاء الحلقة التكرارية واقفز الى ما بعدها. وكما في المثال التالي:



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:

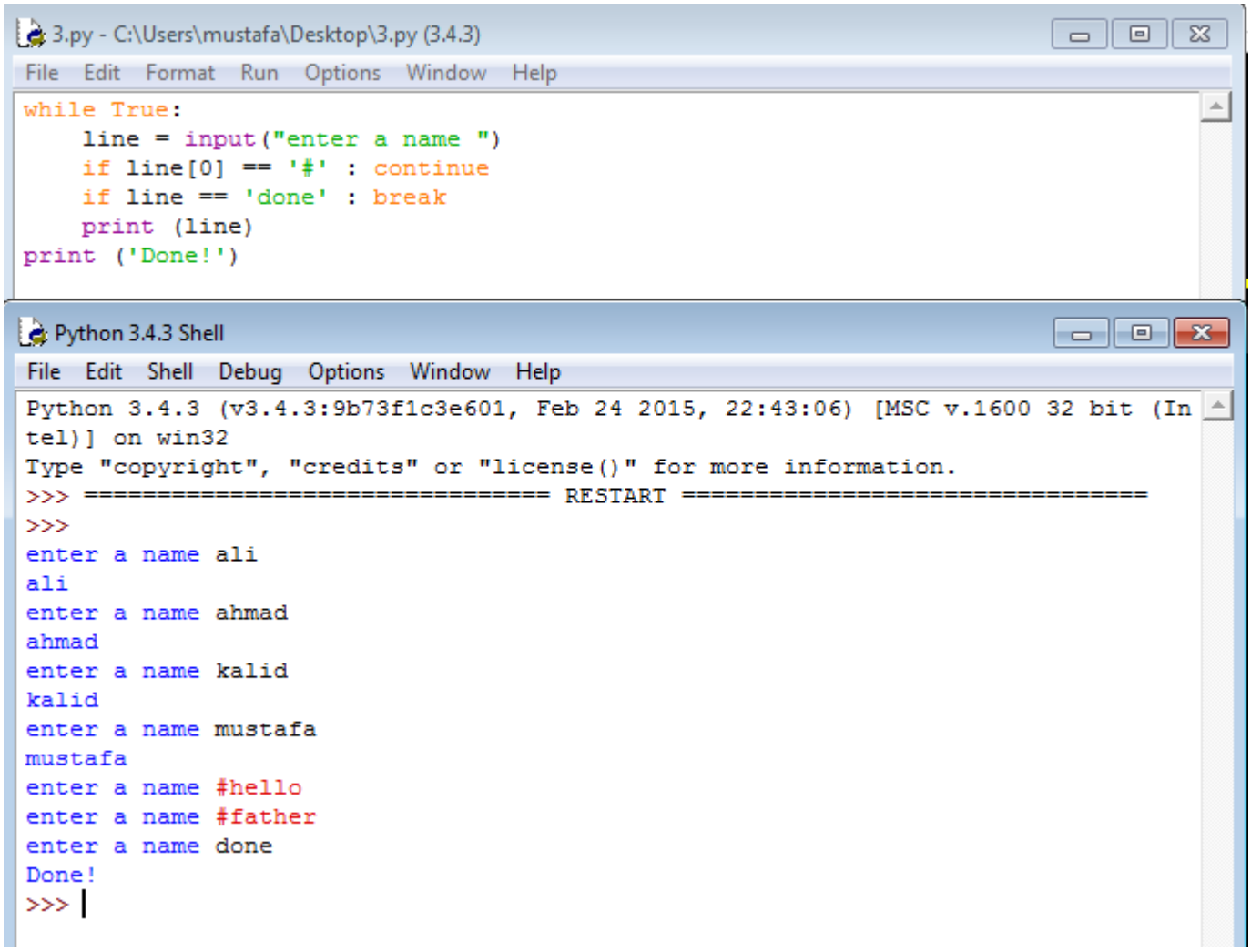
في بعض الأحيان نحتاج الى ترك الحلقة التكرارية الحالية في المنتصف والقفز الى بداية الحلقة من جديد لاستئناف التنفيذ حين يتحقق شرط معين ونستخدم عبارة (continue) لهذا الغرض وكما في المثال ادناه:



هنا استخدمنا نفس المثال السابق مع إضافة شرط يقول (إذا كانت قيمة الادخال 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.

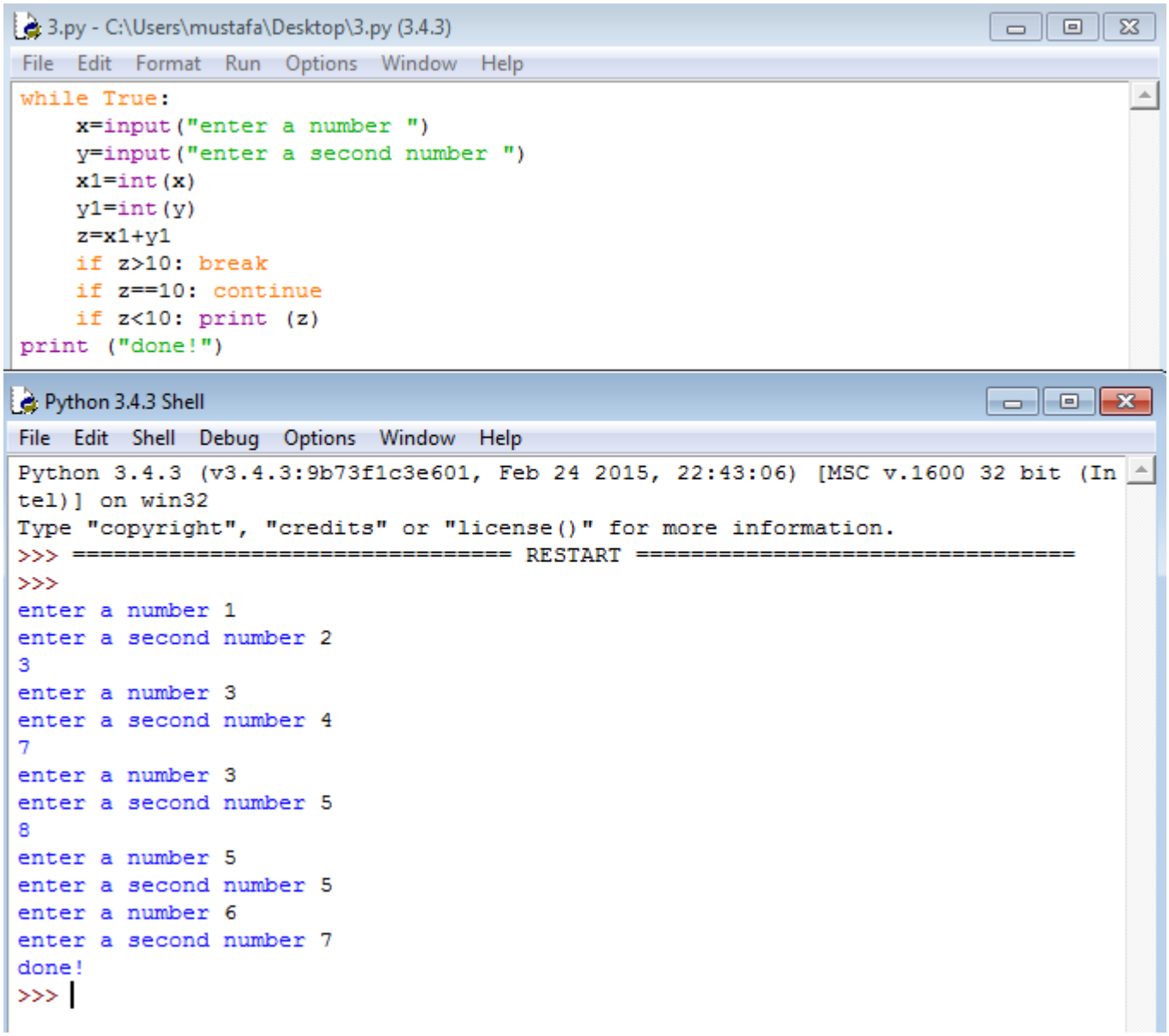
مثال اخر على كيفية التعامل مع (continue) و (break):



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 ادخالات 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:



هنا كان شرط الاستمرار (continue) ان يكون ناتج الجمع ١٠ وشرط 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 بأستخدام (for)

بعد ان شرحنا كيفية برمجة الحلقات التكرارية الغير محددة المدى (indefinite loops) بأستخدام 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 كالاتي:

```
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).

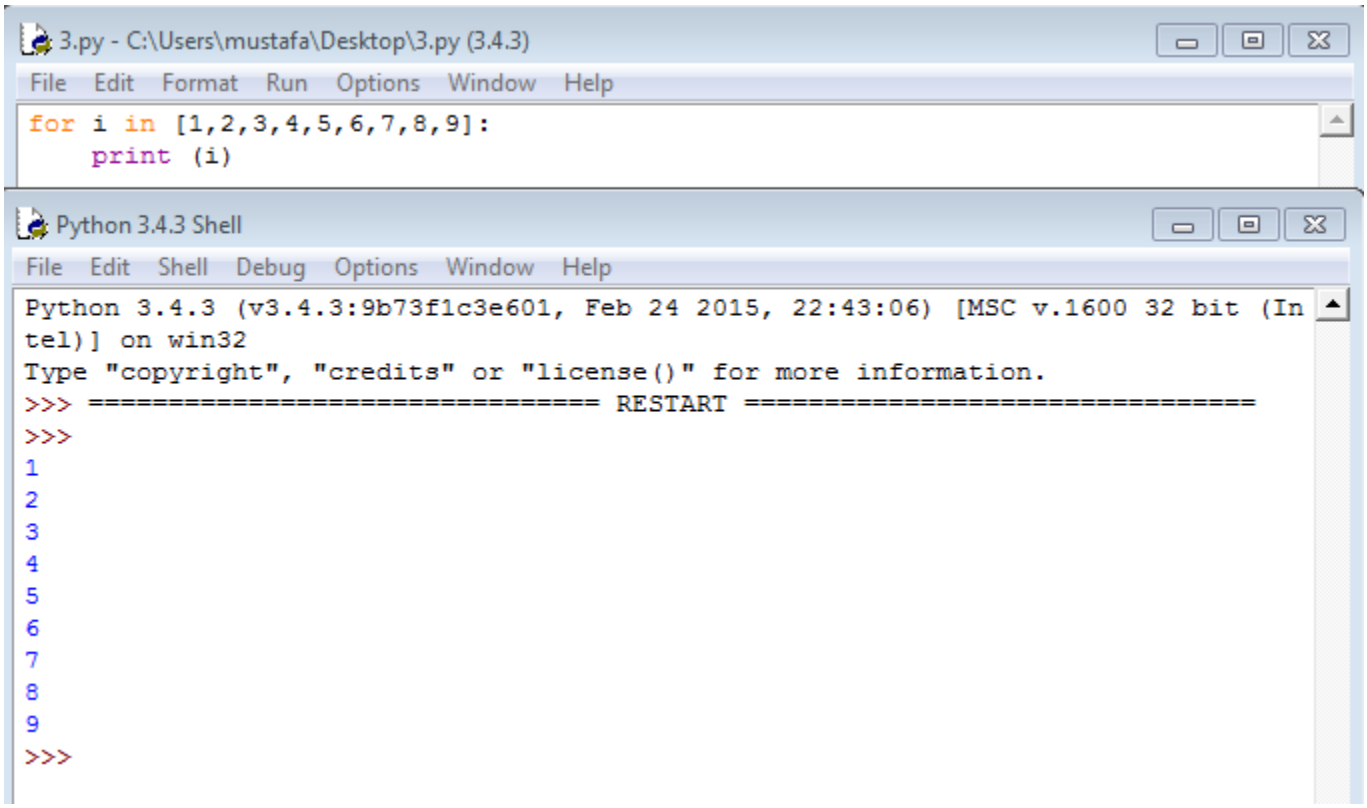
واما (list of numbers, names, …etc.) فهو المدى او قائمة الأرقام او الأسماء او المتغيرات التي سيتحرك ضمنها متغير التكرار ليأخذ قيمها كل مرة.

واخيراً (statement) هي العبارات داخل عبارة (for) والتي تتكرر اعتماداً على متغير التكرار ونلاحظ انها مزحفة الى اليمين بمقدار ٤ 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).

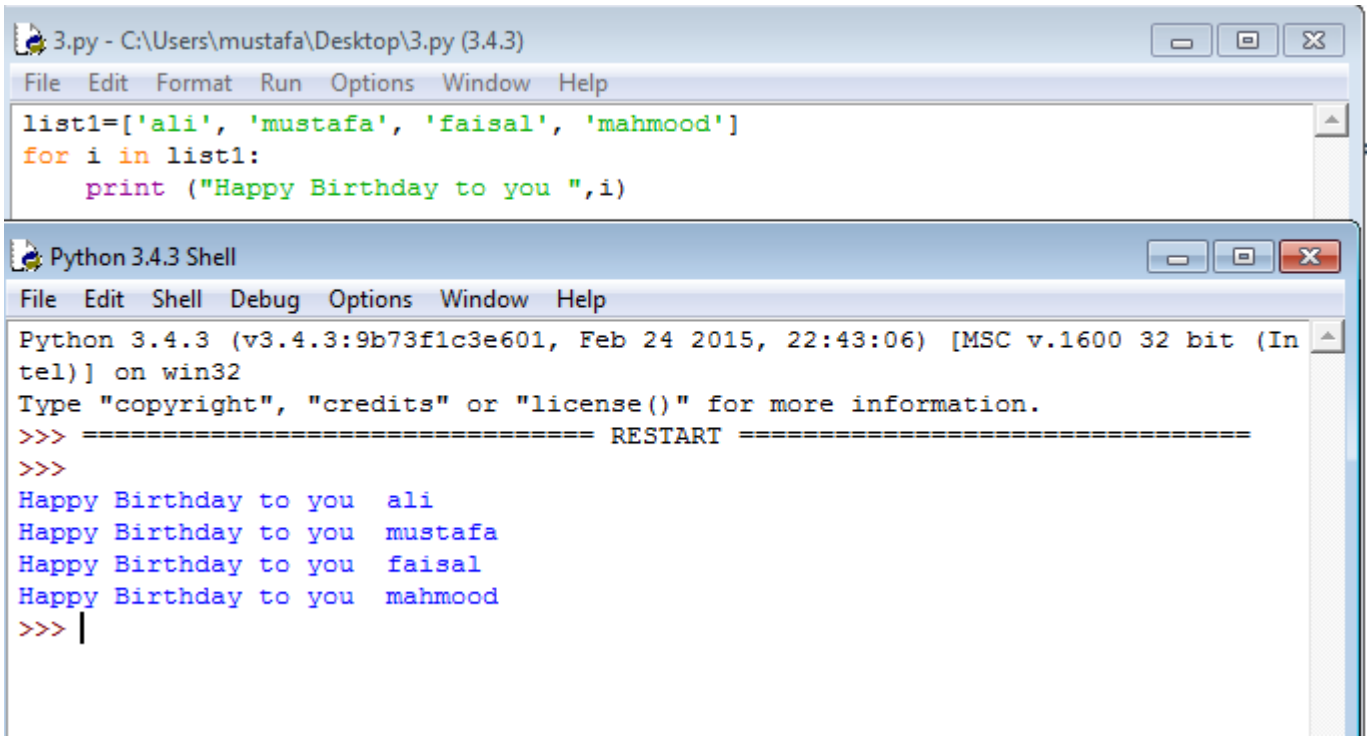
ملاحظة مهمة جداً: يجب مراعاة عدم نسيان (:) في نهاية كل من عبارات الشروط والتكرار فهي مهمة جداً ولا ينفذ البرنامج بدونها.

وفي ادناه مثال بسيط يوضح الفكرة:

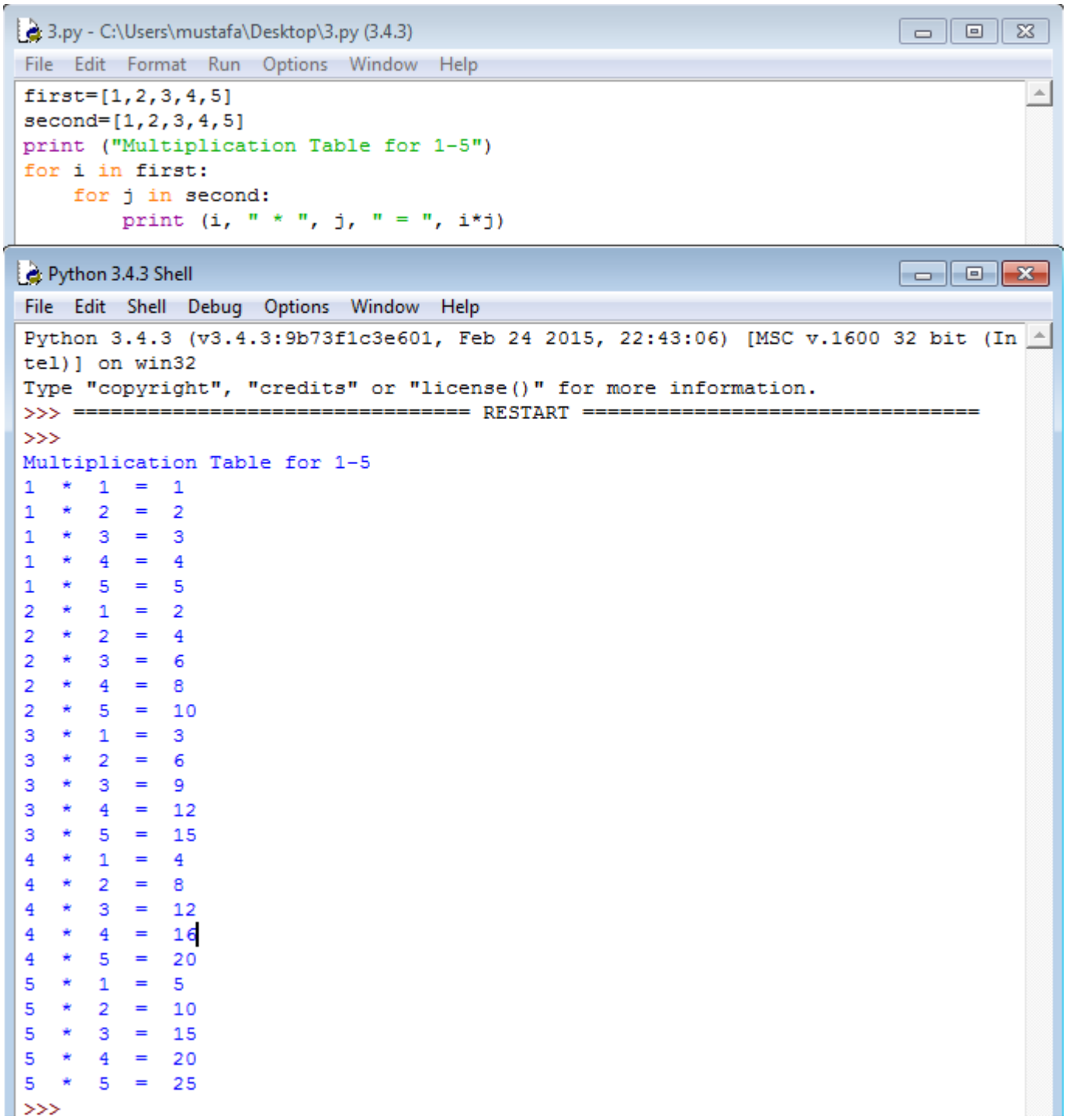


وهذا ابسط مثال على كيفية استخدام عبارة (for) حيث عرفنا متغير التكرار على ان اسمه (i) وهو يأخذ قيمه من القائمة التي تبدأ من ١ وتنتهي بال ٩ واخيراً وبداخل ال(For) قلنا للمفسر فقط اطبع لنا قيم (i).

امثلة أخرى اكثر تفصيلاً:

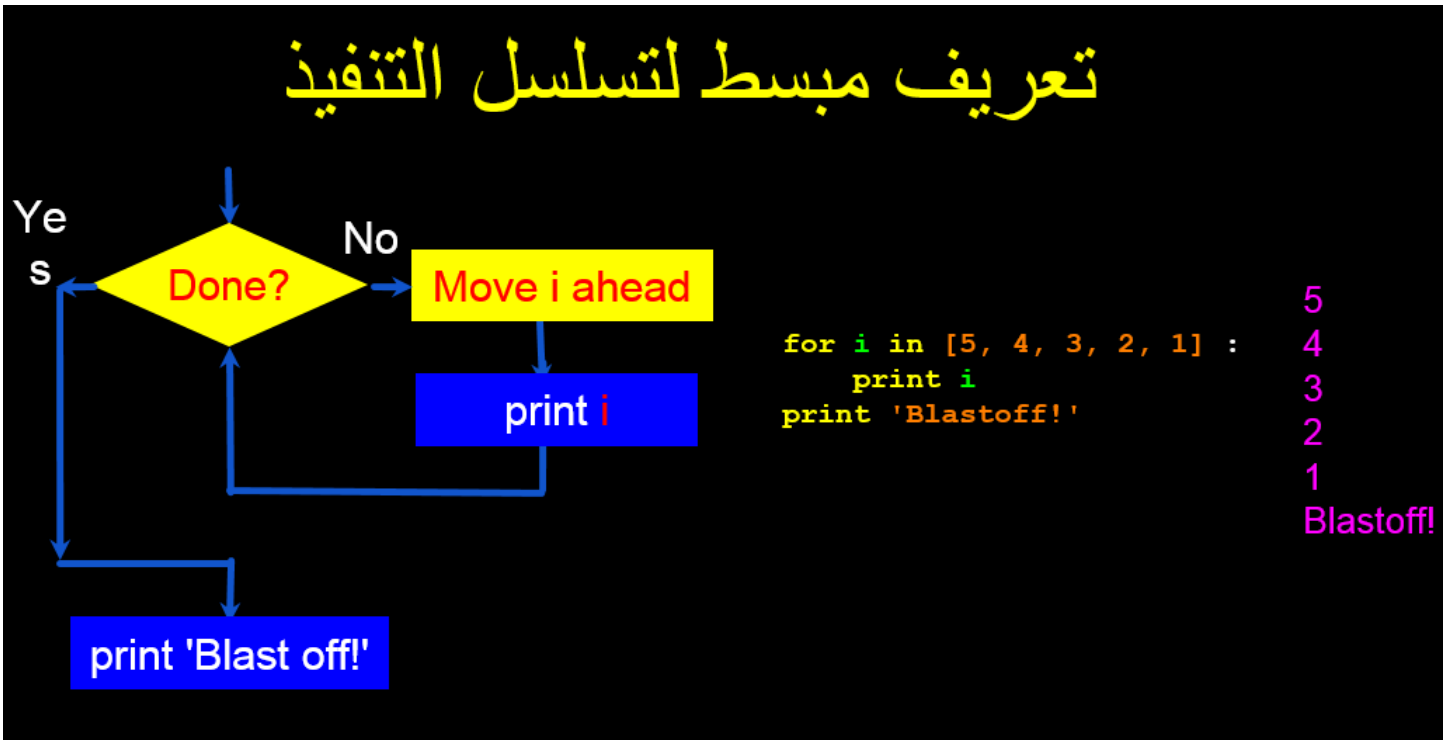


هنا كانت قيم متغير التكرار (i) هي أسماء من قائمة أسماء وليست ارقام.

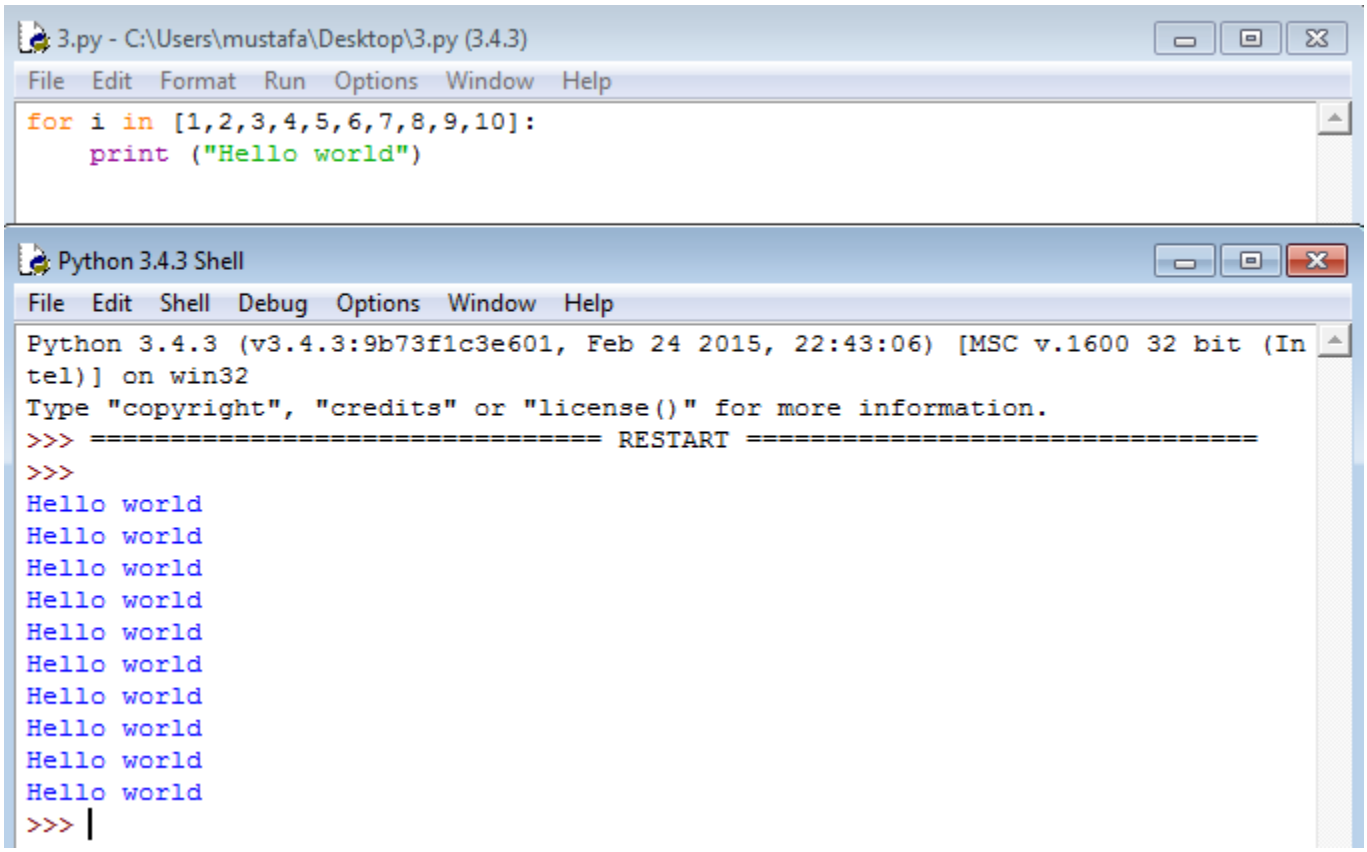


جدول الضرب للأعداد من ١ الى ٥ موضحاً في المثال أعلاه.

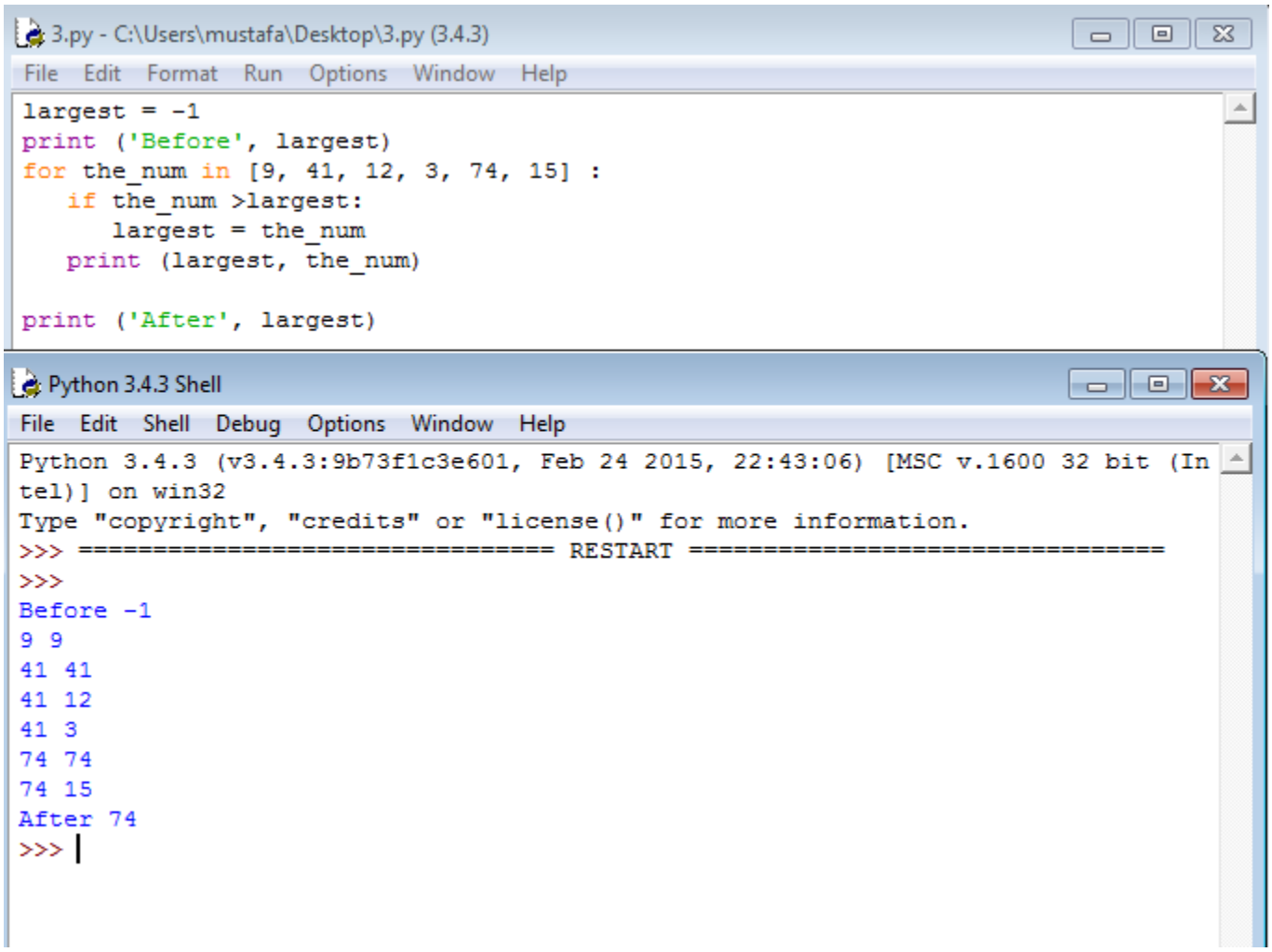
واما لمعرفة كيفية فهم المفسر لعمل عبارة (for) وكيفية تسلسل تنفيذ عباراتها الداخلية فالمخطط التالي يشرح ذلك ببساطة:



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:

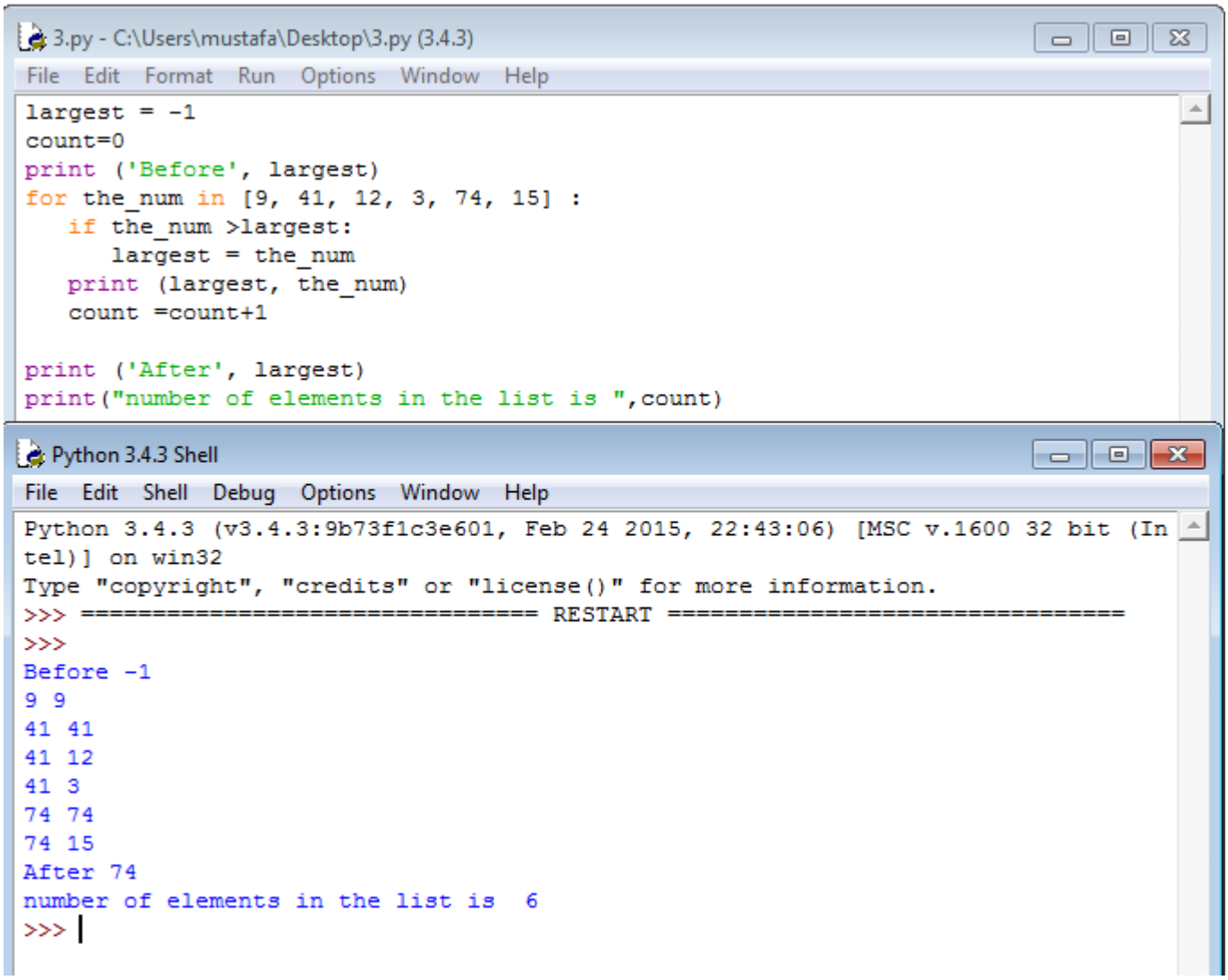


المثال أعلاه يبين ان متغير التكرار لا يشترط ان يكون موجوداً في عبارات داخل عبارة (For) بل انه يمكن ان يستخدم كمتغير تكرار للمفسر فقط وليس للمستخدمين.

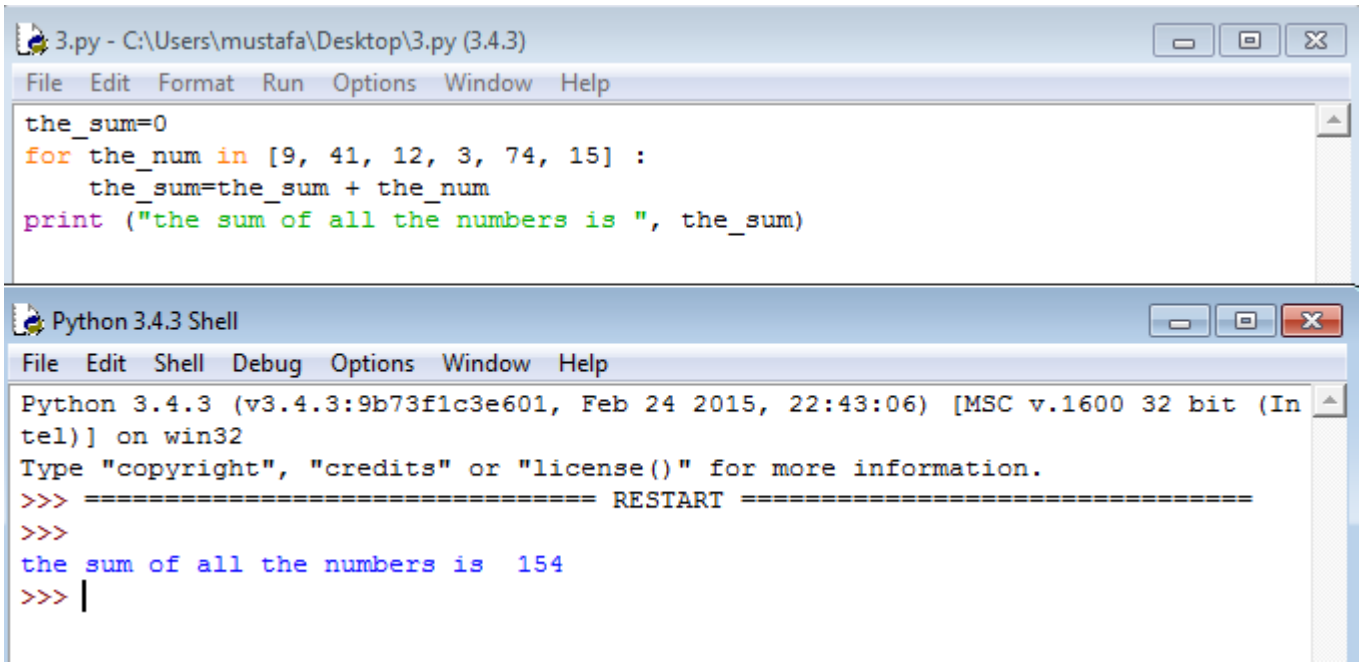


برنامج لأيجاد العدد الأكبر في قائمة اعداد في الصورة أعلاه.

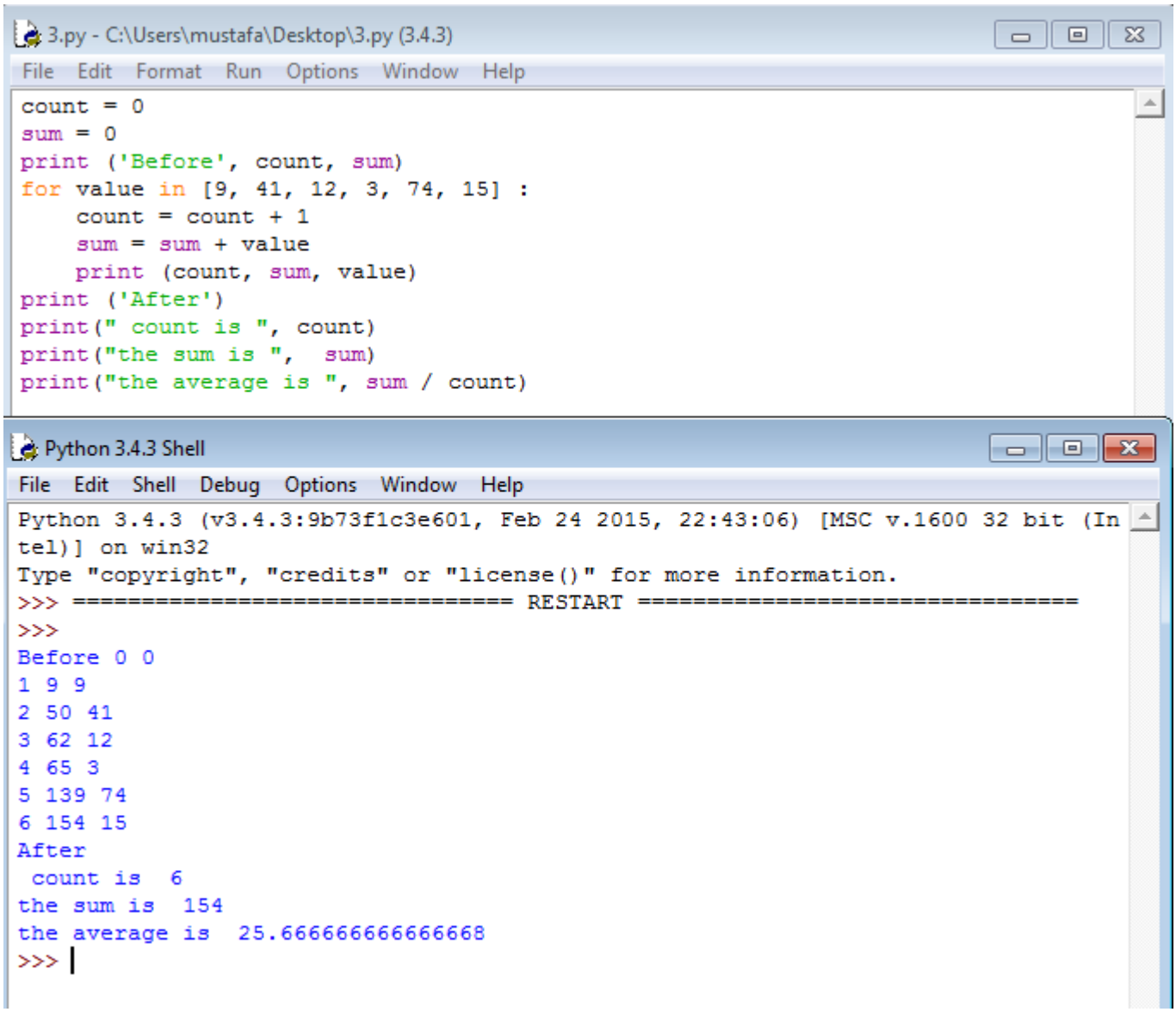
والان نفس البرنامج مع إضافة قابلية حساب عدد مرات التكرار للحلقة التكرارية وكما في ادناه:



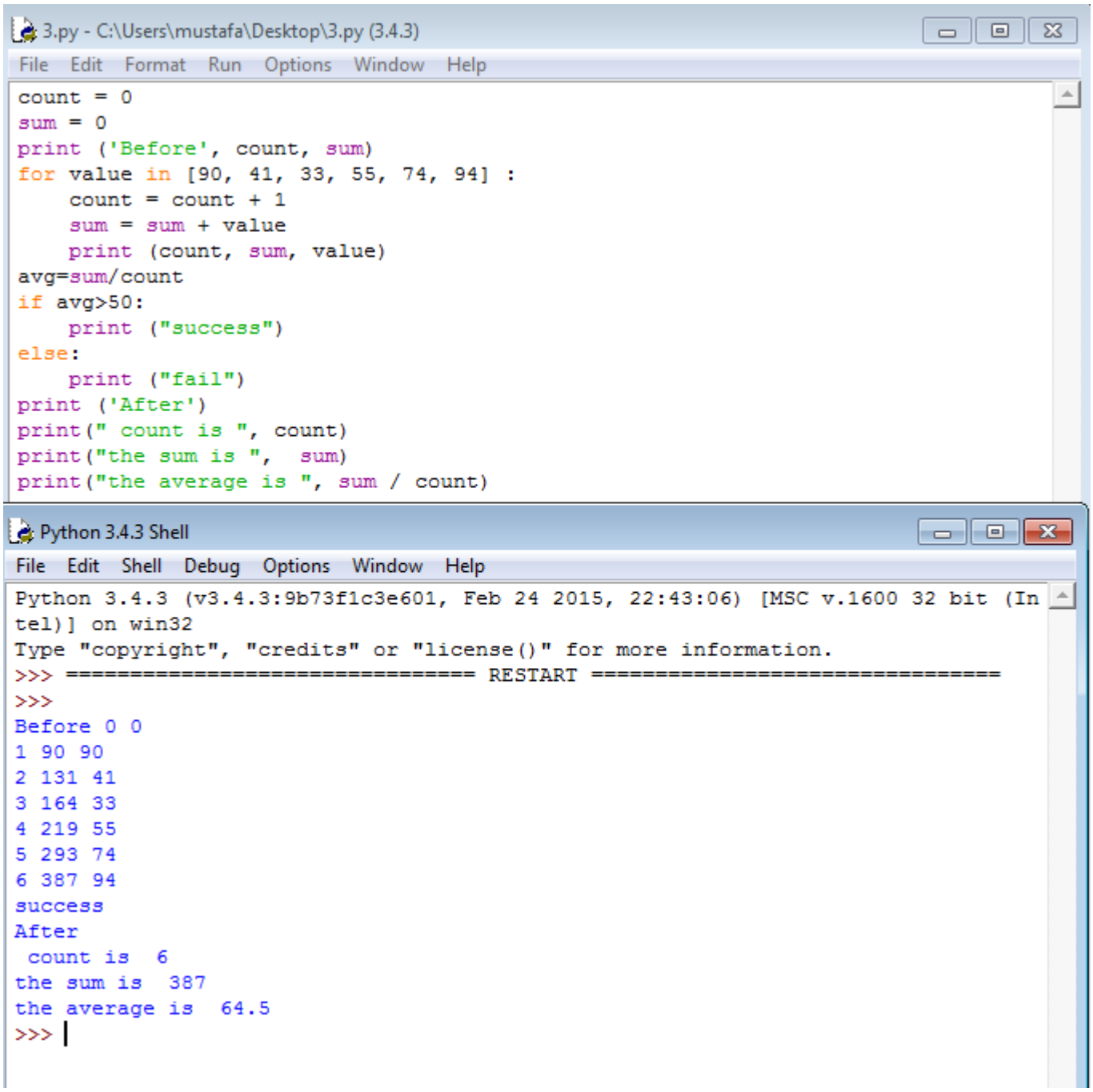
لأيجاد مجموع قيم متغير التكرار يمكن الاستعانة بالبرنامج التالي:



والان لحساب معدل مجموعة من الأرقام نلاحظ المثال التالي:



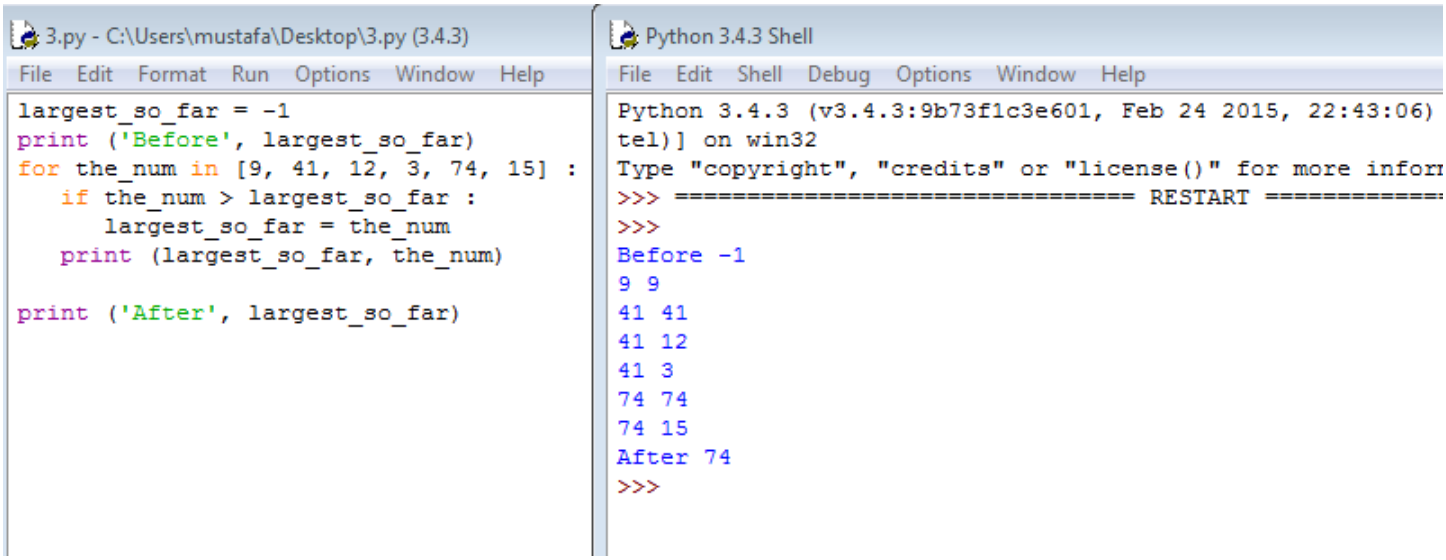
ولأستخدام كل من عبارات التكرار والشروط يمكن ملاحظة المثال التالي:


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

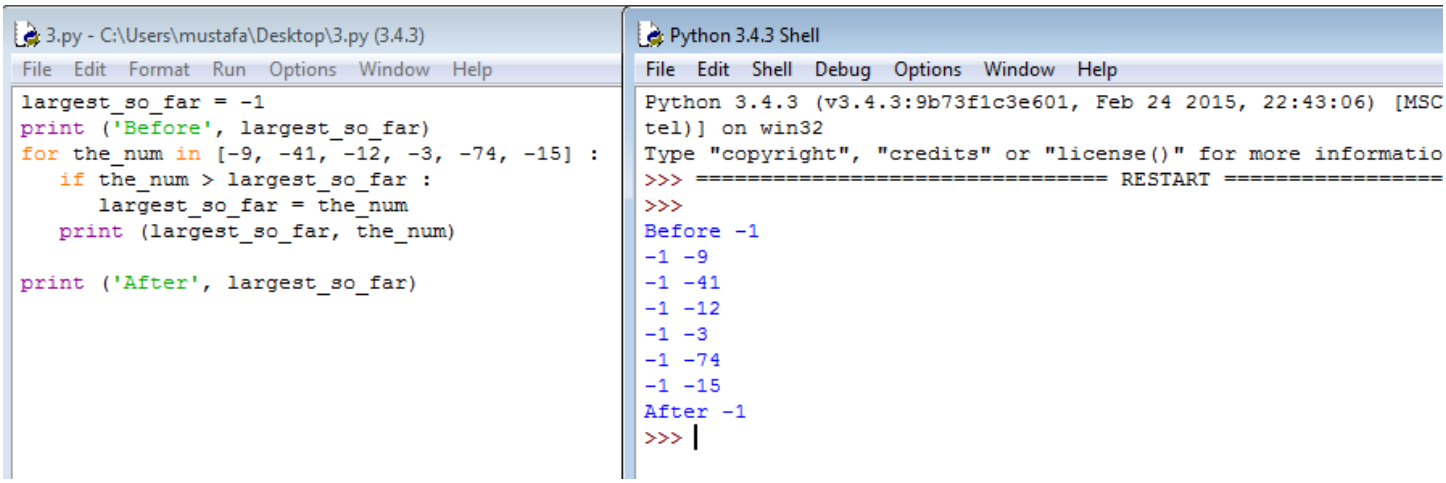
بعد ان شرحنا العبارات المستخدمة في الحلقات التكرارية مثل (while) و (for) نأتي اليوم الى مناقشة مشكلة يوضحها المثال التالي:

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



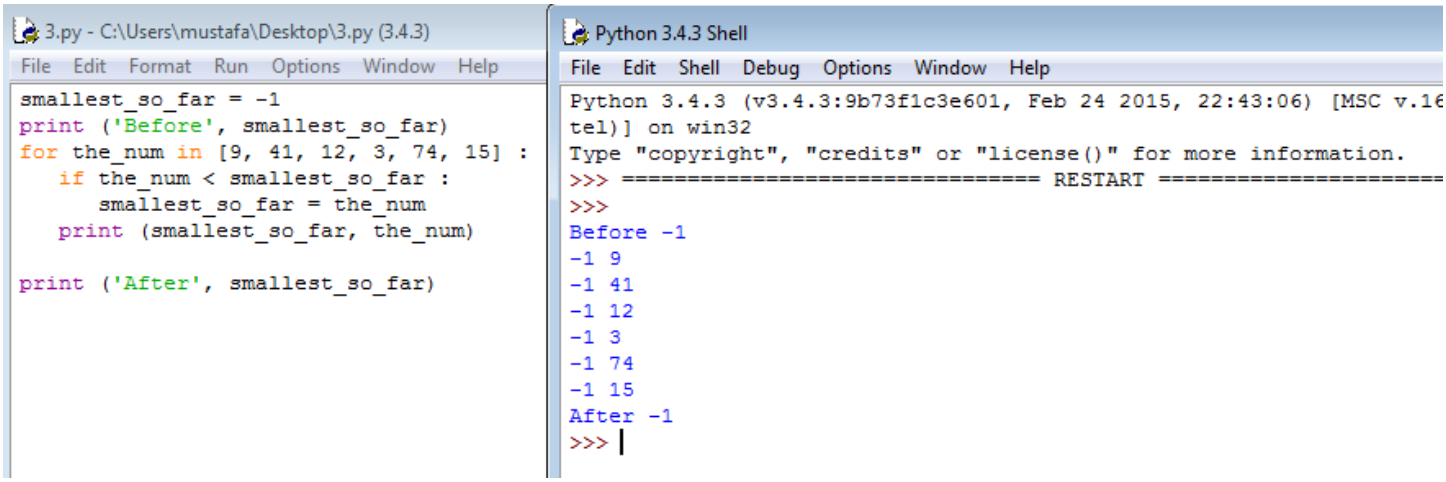
وهنا كما هو واضح اعتمدنا على إعطاء قيمة أولية للمتغير الأكبر (largest_so_far) وهي قيمة صغيرة مقدارها سالب واحد وقد حصلنا على نتائج صحيحة ولكن ماذا لو كانت كل الاعداد في القائمة سالبة واصغر من السالب واحد؟

دعونا نرى ماذا سيكون الإخراج:



كما هو متوقع تماماً، قام المفسر بمقارنة القيمة الأولية وهي سالب واحد بكل القيم الأخرى التي يصدف انها 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.

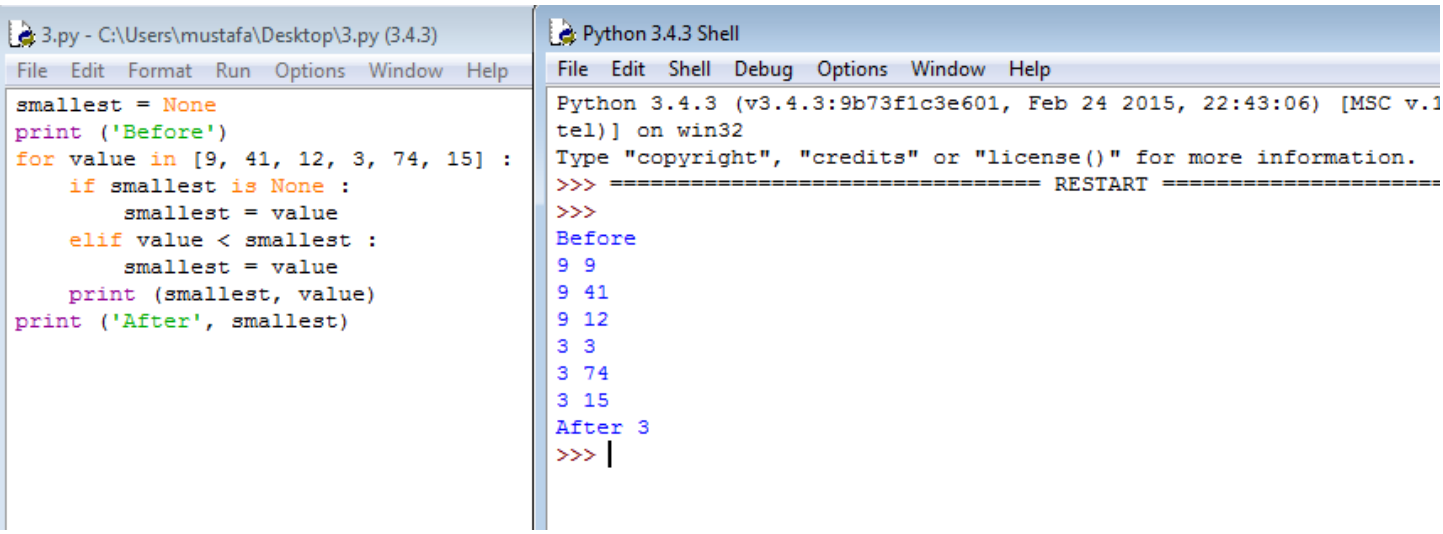
مثال اخر على نفس المشكلة: وهو البحث عن اصغر عدد في القائمة وكما في الصورة ادناه:



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!

اذا 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:



والان لنشرح خطوات البرنامج:

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

السطر الثاني: طباعة عبارة (before)

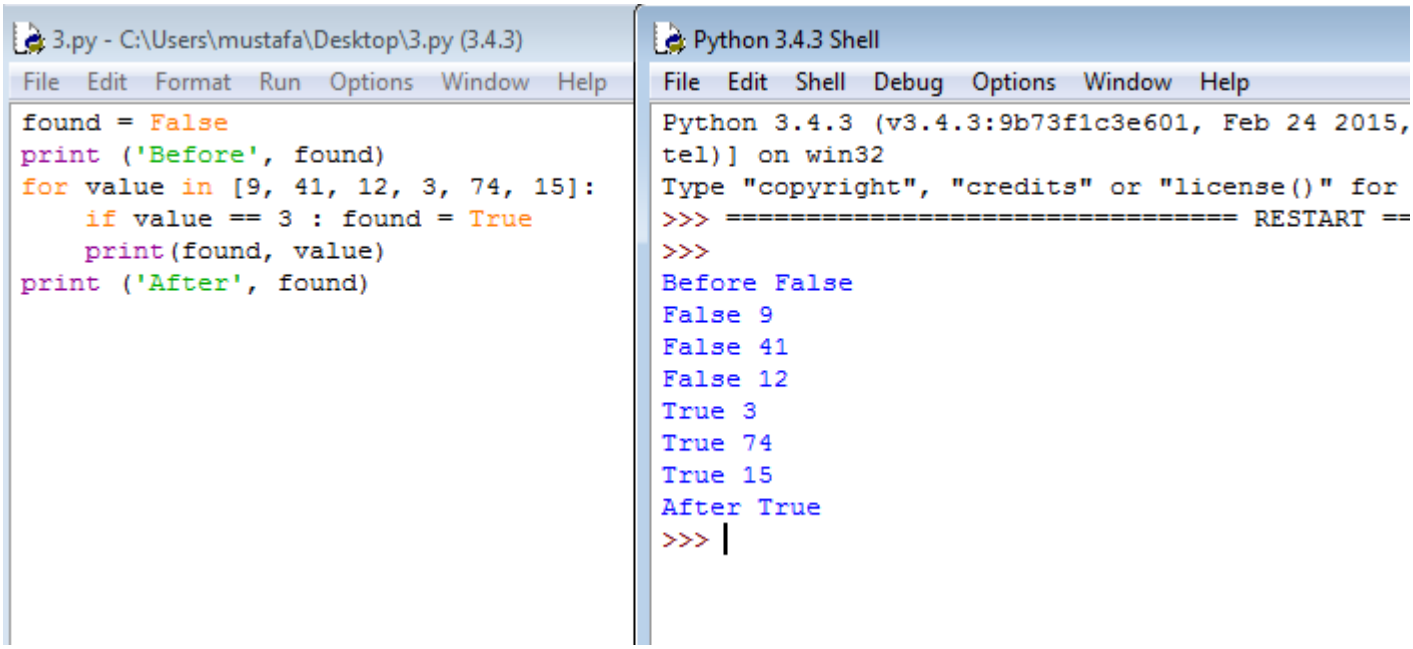
السطر الثالث: عبارة for للبحث بداخل القائمة

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 بأستخدام القيم المنطقية صح (True) وخطأ (False):



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.

```
largest = None
smallest = None
while True:
    num = raw_input("Enter an integer number: ")
    if num == "done" : break
    try:
        number=int(num)
        if largest is None:
            if smallest is None:
                largest=number
                smallest=number
        elif number>largest:
            largest=number
        elif number<smallest:
            smallest=number
    except:
        print "Invalid input"
print "Maximum is", largest
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

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 ☺