Unit -1

Topics to be covered:

Introduction: History of Python, Need of Python Programming, Applications, Basics of Python Programming Using the REPL(Shell), Running Python Scripts, Variables, Assignment, Keywords, Input-Output, Indentation.

History of Python

1. Python was created by ***Guido Van Rossum***, who was a Dutch person from Netherlands in the Year 1991.
2. Guideo Van Rossum created it when he was working in National Research Institute of Mathemetics and Computer Science in Netherlands.
3. He thought to create a scripting language as a "Hobby" in Christmas break in 1980. He studied all the languages like ABC (All Basic Code), C, C++, Modula-3, Smalltalk, Algol-68 and Unix Shell and collected best features.
4. He stared implementing it from 1989 and finished and released first working version of Python in 1991.
5. He named it as "Python",being a big fan of "Monty Python's Flying Circus" comedy show broadcasted in BBC from 1969 to 1974.

Versions History:

i. **Python 1.0 (1994)**
   a. Python 1.5(1997)
   b. Python 1.6 (2000)

ii. **Python 2.0 (2000)**
   a. Python 2.1 (2001)
   b. Python 2.2 (2001)
   c. Python 2.3 (2003)
   d. Python 2.4 (2004)
   e. Python 2.5 (2006)
   f. Python 2.6 (2008)
   g. Python 2.7 (2010)

iii. **Python 3.0 (2008)**
   a. Python 3.1(2009)
   b. Python 3.2(2011)
   c. Python 3.3 (2012)
   d. Python 3.4(2014)

Note: Python 3.0 was designed to rectify the problems in the older versions. It has lot many new features when compared to python.

Need of Python Programming

The following are the factors to use Python

1. ***Software quality*** - Python code is designed to be *readable*, and hence reusable and maintainable

2. ***Developer productivity*** - Python boosts developer productivity many times beyond compiled or statically typed languages such as C, C++, and Java. Python code is typically *one-third to one-fifth the size of* equivalent C, C++ and Java Programs.
3. ***Program portability*** - Most Python programs run unchanged on *all major computer platforms*. Porting Python code between Linux, Windows, and Mac.
4. ***Support libraries***- Python comes with a large collection of prebuilt and portable functionality, known as the *standard library*. This library supports an array of application-level programming tasks, from text pattern matching to network scripting.
5. ***Component integration*** - Python scripts can easily communicate with other parts of an application, using a variety of integration mechanisms. Such integrations allow Python to be used as a product *customization and extension* tool. Today, Python code can invoke C and C++ libraries.
6. ***Enjoyment***- It is easy to use. It gives programmer flexibility to write programs.

Applications of Python

Python's most common applications are as follow:

Systems Programming: Python's built-in interfaces to operating-system services make it ideal for writing portable, maintainable system-administration tools and utilities (sometimes called *shell tools*). Python programs can search files and directory trees, launch other programs, do parallel processing with processes and threads, and so on.

GUIs: Python's simplicity and rapid turnaround also make it a good match for graphical user interface programming on the desktop. Python comes with a standard object-oriented interface to the Tk GUI API called *tkinter* (*Tkinter* in 2.X) that allows Python programs to implement portable GUIs with a native look and feel. Python/tkinter GUIs run un-changed on Microsoft Windows, X Windows (on Unix and Linux), and the Mac OS (both Classic and OS X).

Internet Scripting: Python comes with standard Internet modules that allow Python programs to perform a wide variety of networking tasks, in client and server modes. Scripts can communicate over sockets, extract form information sent to server-side CGI(common gateway interface) scripts, transfer files by FTP(file transfer protocol), parse and generate XML and JSON documents, send, receive, compose, and parse email, and fetch web pages by URLs(uniform resource locator).

Component Integration: Python scripts can easily communicate with other parts of an application, using a variety of integration mechanisms. Such integrations allow Python to be used as a product *customization and extension* tool. Today, Python code can invoke C and C++ libraries.

Database Programming: For traditional database demands, there are Python interfaces to all commonly used relational database systems—Sybase, Oracle, Informix, ODBC, MySQL, PostgreSQL, SQLite, and more. The Python world has also defined a *portable database API* for accessing SQL database systems from Python scripts, which looks the same on a variety of underlying database systems.

Rapid Prototyping: To Python programs, components written in Python and C look the same. Because of this, it's possible to prototype systems in Python initially, and then move selected components to a compiled language such as C or C++ for delivery. Unlike some prototyping tools, Python doesn't require a complete rewrite once the prototype has solidified. It increases efficiency and reduce time and cost.

Numeric and Scientific Programming: Python has a module called "NumPy" to support all numerical programming. Python also has "SciPy" to support all the scientific application. Hence python is suitable for both numeric and scientific applications.

Using NumPy, a developer can perform the following operations −

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

And More: Gaming, Images, Data Mining, Robots, Excel...

Python is commonly applied in more domains than can be covered here. For example, you'll find tools that allow you to use Python to do:

- Game programming and multimedia with *pygame*, *cgkit*, *pyglet*, *PySoy*, *Panda3D*, and others
- Serial port communication on Windows, Linux, and more with the *PySerial* ex-tension
- Image processing with *PIL* and its newer *Pillow* fork, *PyOpenGL*, *Blender*, *Maya*, and more
- Robot control programming with the *PyRo* toolkit
- Natural language analysis with the *NLTK* package
- Instrumentation on the *Raspberry Pi* and *Arduino* boards
- Mobile computing with ports of Python to the Google *Android* and Apple *iOS* platforms
- Excel spreadsheet function and macro programming with the *PyXLL* or *DataNi-tro* add-ins
- Media file content and metadata tag processing with *PyMedia*, *ID3*, *PIL*/*Pillow*, and more
- Artificial intelligence with the *PyBrain* neural net library and the *Milk* machine learning toolkit
- Expert system programming with *PyCLIPS*, *Pyke*, *Pyrolog*, and *pyDatalog*
- Network monitoring with *zenoss*, written in and customized with Python
- Python-scripted design and modeling with *PythonCAD*, *PythonOCC*, *FreeCAD*, and others
- Document processing and generation with *ReportLab*, *Sphinx*, *Cheetah*, *PyPDF*, and so on
- Data visualization with *Mayavi*, *matplotlib*, *VTK*, *VPython*, and more
- XML parsing with the xml library package, the xmlrpclib module, and third-party extensions
- JSON and CSV file processing with the json and csv modules

--------*******--------

Features of Python

Open source: Python is publicly available open source software, any one can use source code that doesn't cost anything.

Easy-to-learn: Popular (scripting/extension) language, clear and easy syntax, no type declarations, automatic memory management, high-level data types and operations, design to read (more English like syntax) and write (shorter code compared to C, C++, and Java) fast.

High-level Language: High-level language (closer to human) refers to the higher level of concept from machine language (for example assembly languages). Python is an example of a high-level language like C, C++, Perl, and Java with low-level optimization.

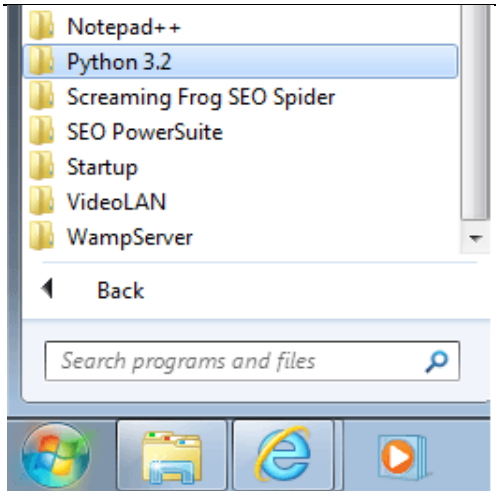
Portable: High level languages are portable, which means they are able to run across all major hardware and software platforms with few or no change in source code. Python is portable and can be used on Linux, Windows, Macintosh, Solaris, FreeBSD, OS/2, Amiga, AROS, AS/400 and many more.
Object-Oriented: Python is a full-featured object-oriented programming language, with features such as classes, inheritance, objects, and overloading.
Python is Interactive: Python has an interactive console where you get a Python prompt (command line) and interact with the interpreter directly to write and test your programs. This is useful for mathematical programming.
Interpreted: Python programs are interpreted, takes source code as input, and then compiles (**to portable byte-code**) each statement and executes it immediately. No need to compiling or linking.
Extendable: Python is often referred to as a "**glue**" language, meaning that it is capable to work in mixed-language environment. The Python interpreter is easily extended and can add a new built-in function or modules written in C/C++/Java code.
Libraries: Databases, web services, networking, numerical packages, graphical user interfaces, 3D graphics, others.
Supports: Support from online Python community

Basics of Python Programming Using the REPL(Shell)

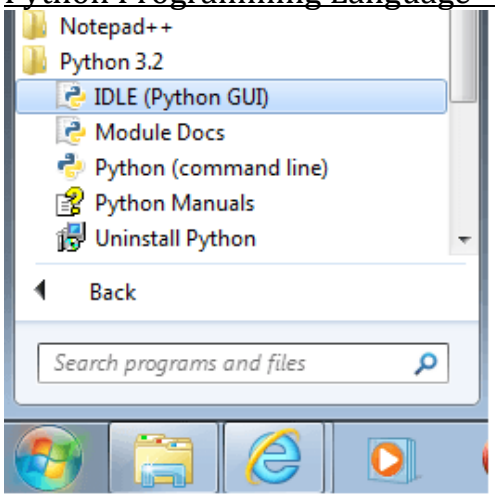
A ***Read-Eval-Print Loop*** (REPL) is a simple, interactive computer programming environment. The term 'REPL' is usually used to refer to a LISP interactive environment but can be applied to command line shells and similar environments for programming languages like Python, Ruby etc.

Python IDLE: Interactive Mode

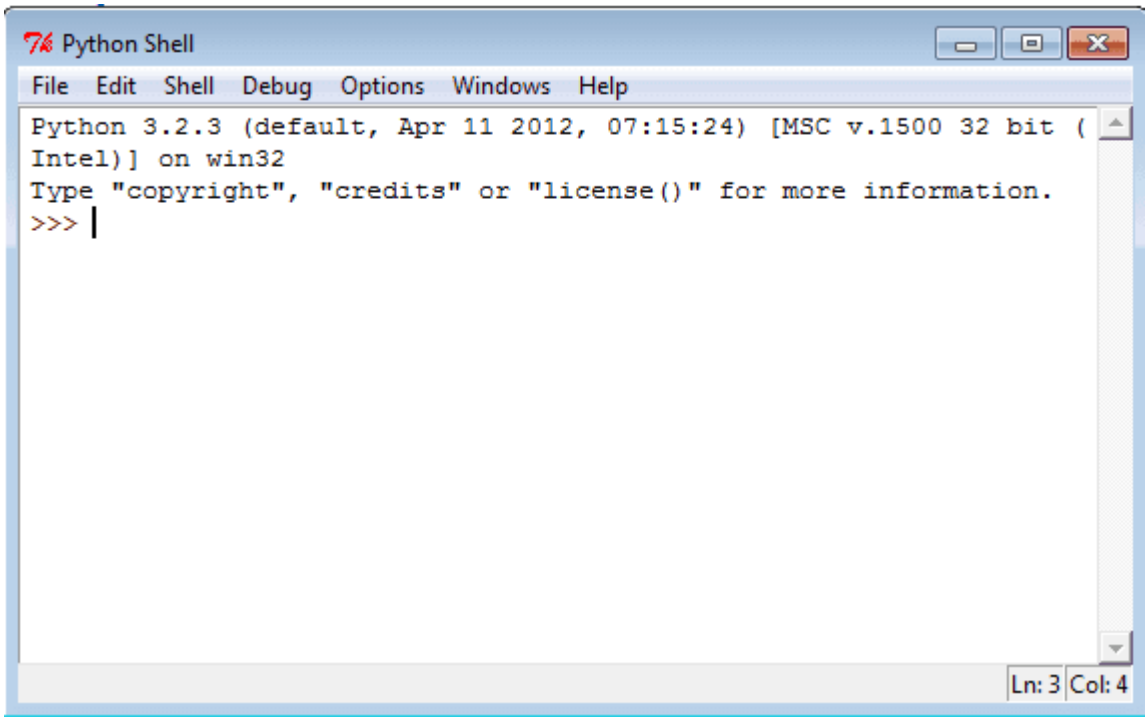
Let us assume that we've already installed Python (here we installed Python 3.2 on a standard pc with windows 7 OS). Click on start button and find Python 3.2 tab in installed programs.



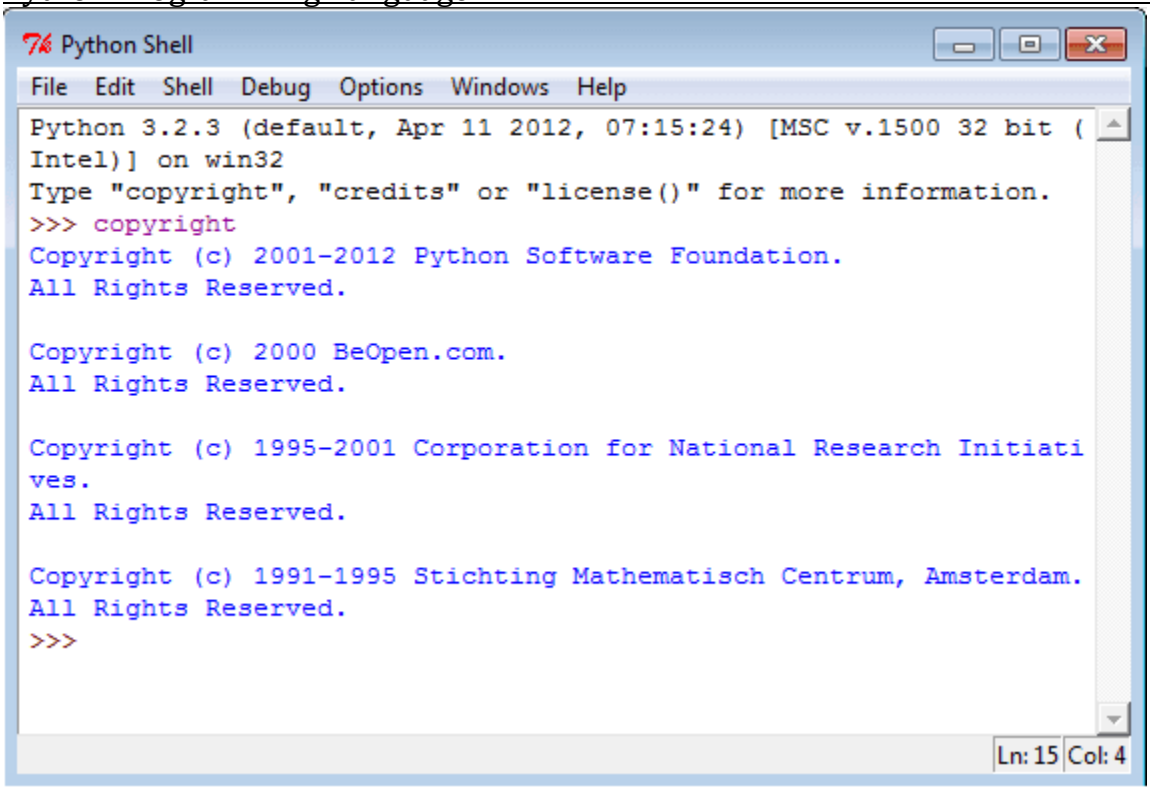
Now clicking on Python 3.2 tab you can see the Python program development tool named IDLE (Python GUI).



To start IDLE click on IDLE (Python GUI) icon, you will see a new window opens up and the cursor is waiting beside '>>>' sign which is called command prompt.



This mode is called interactive mode as you can interact with IDLE directly, you type something (single unit in a programming language) and press enter key Python will execute it, but you can not execute your entire program here. At the command prompt type copyright and press enter key Python executes the copyright information.

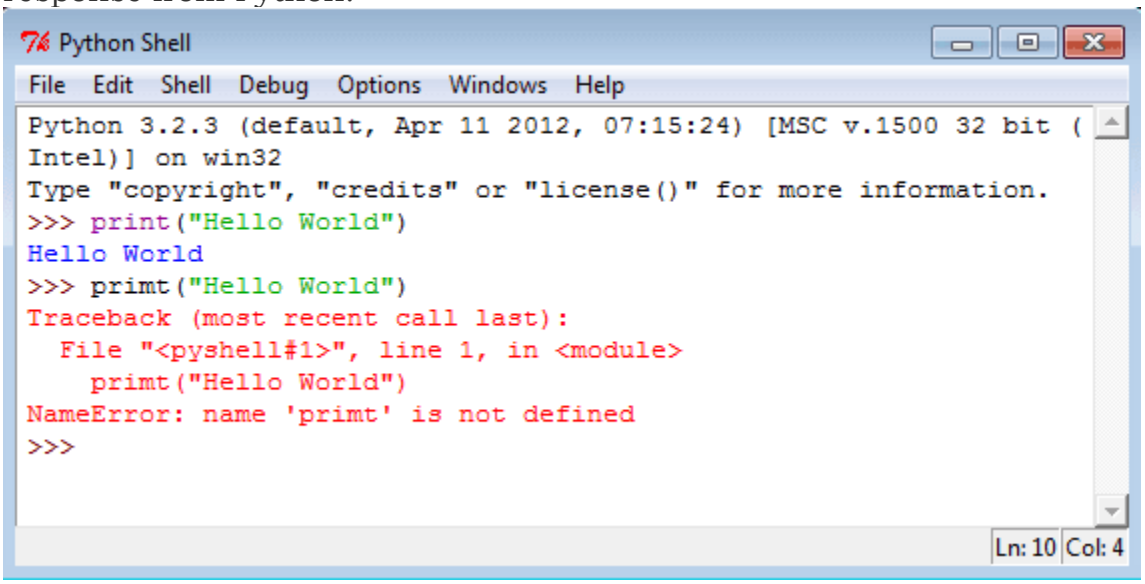


Now Python is ready to read new command. Let's execute the following commands one by one.

Command -1 : print("Hello World")

Command -2 : primt("Hello World")

The first command is correct and but the second one has a syntax error, here is the response from Python.

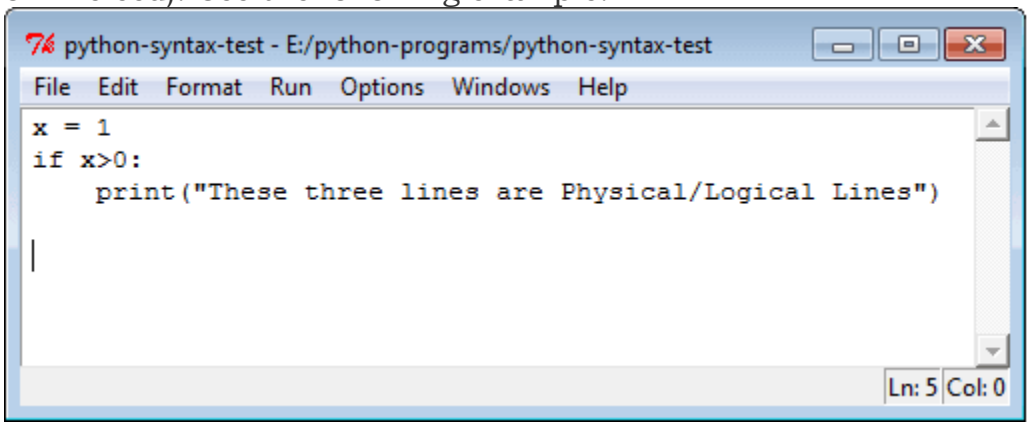


Python line structure:

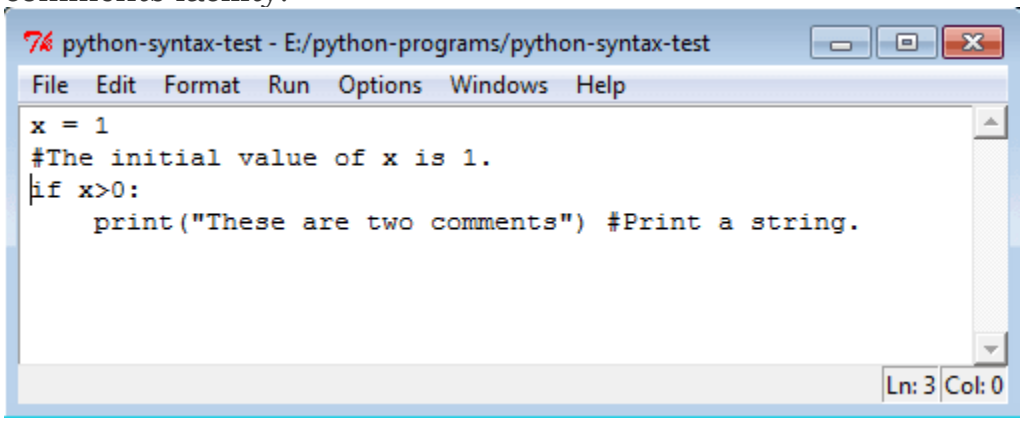
A Python program is divided into a number of logical lines and every logical line is terminated by the token NEWLINE. A logical line is created from one or more physical lines.

A line contains only spaces, tabs, form feeds possibly a comment, is known as a blank

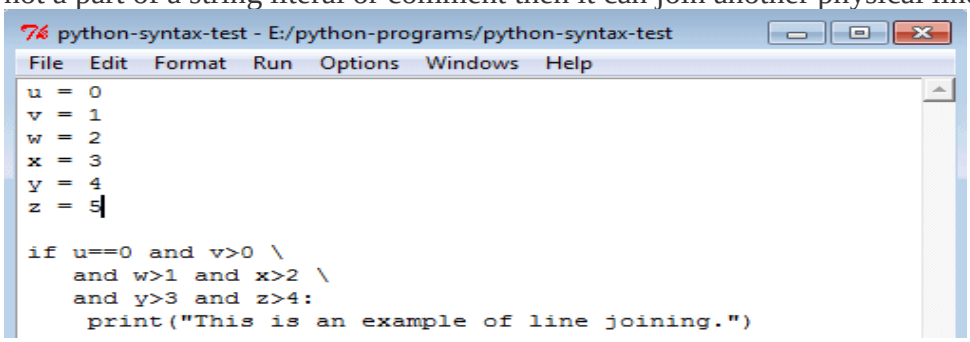
line, and Python interpreter ignores it.
A physical line is a sequence of characters terminated by an end-of-line sequence (in windows it is called CR LF or return followed by a linefeed and in Unix, it is called LF or linefeed). See the following example.



Comments in Python:
A comment begins with a hash character(#) which is not a part of the string literal and ends at the end of the physical line. All characters after the # character up to the end of the line are part of the comment and the Python interpreter ignores them. See the following example. It should be noted that Python has no multi-lines or block comments facility.

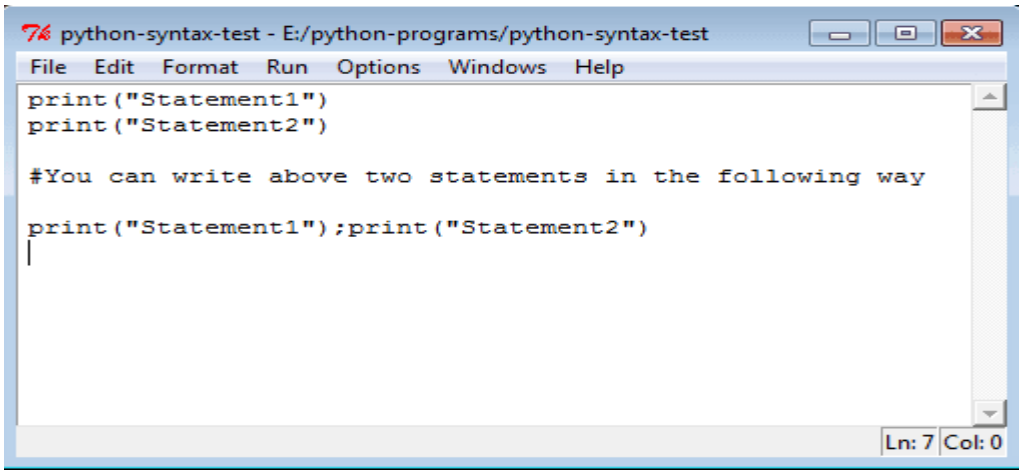


Joining two lines:
When you want to write a long code in a single line you can break the logical line in two or more physical lines using backslash character(\). Therefore when a physical line ends with a backslash characters(\) and not a part of a string literal or comment then it can join another physical line. See the following example.



Multiple statements on a single line:

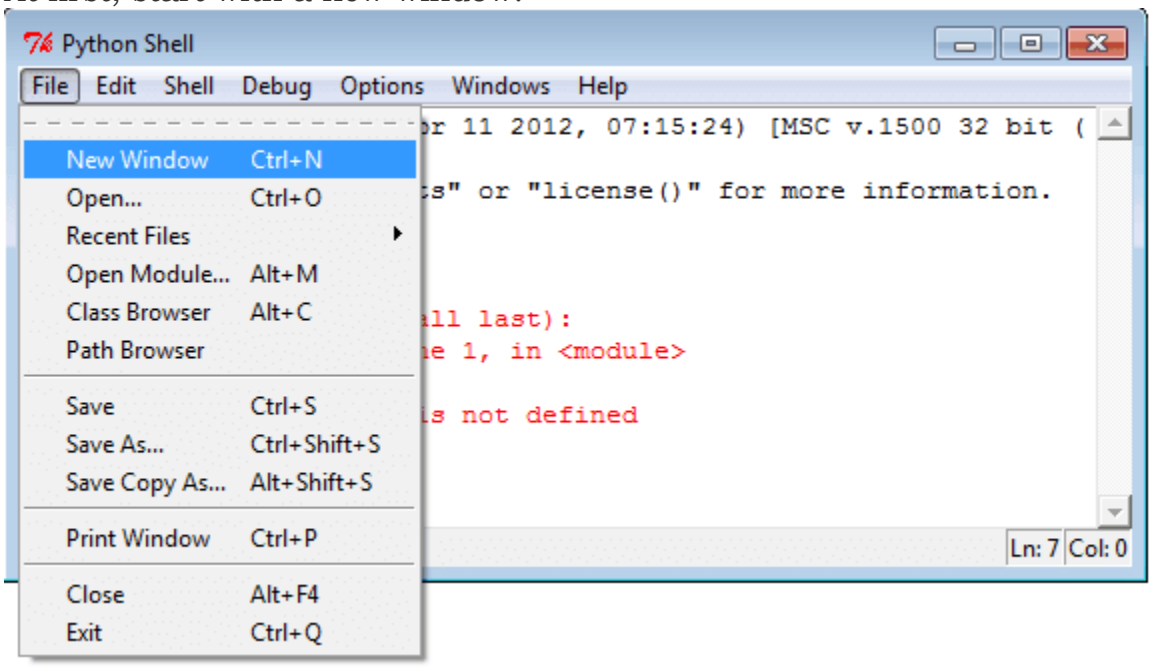
You can write two separate statements into a single line using a semicolon (;) character between two line.



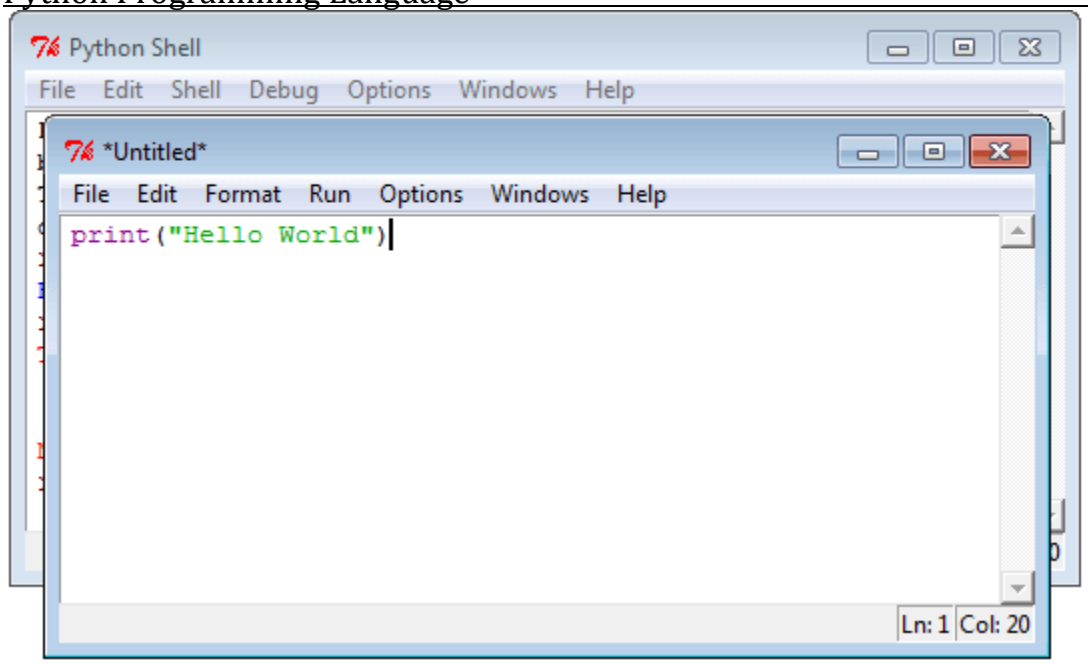
Running Python Scripts

Python IDLE: Development mode

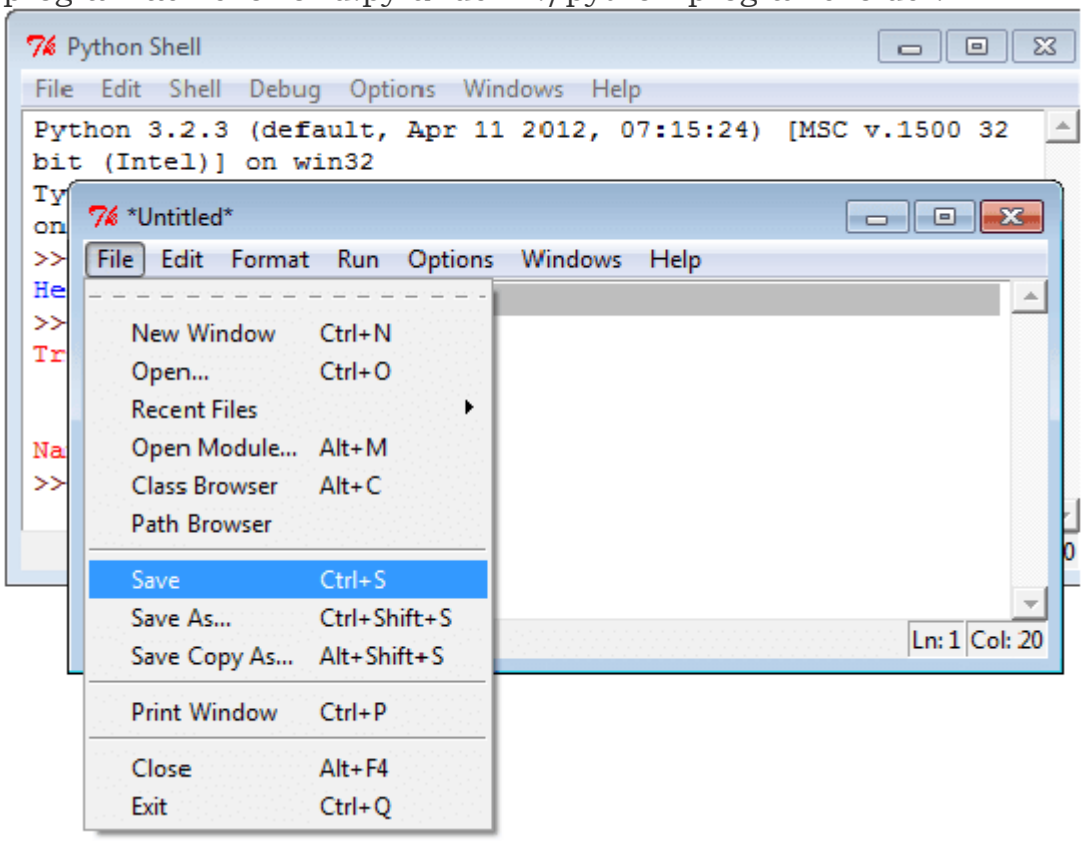
At first, start with a new window.



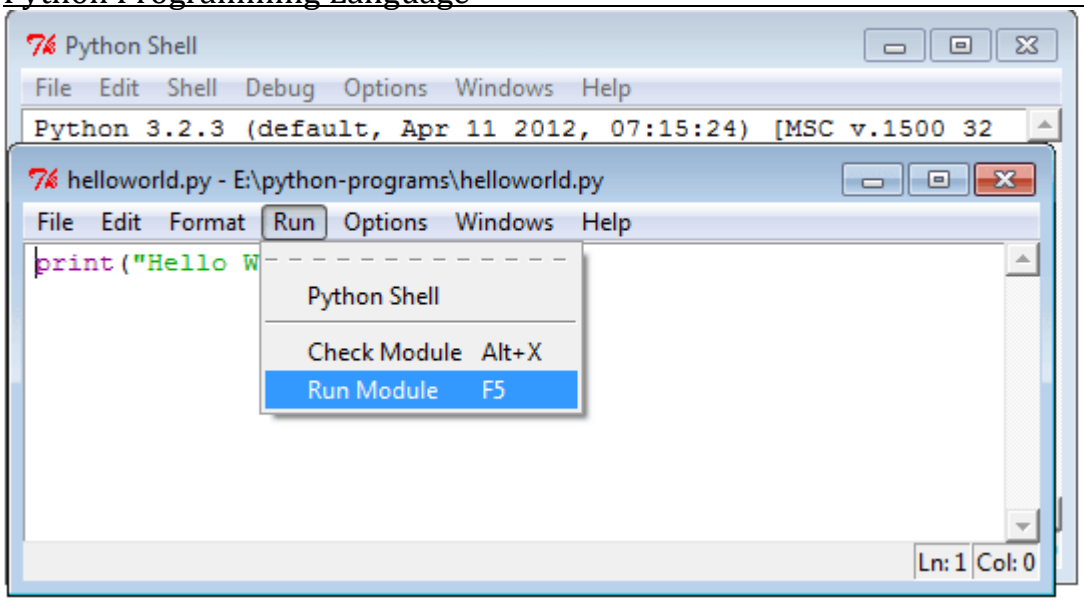
Clicking on "New window" under file menu a new window will come. Type print "Hello World" in the new window.



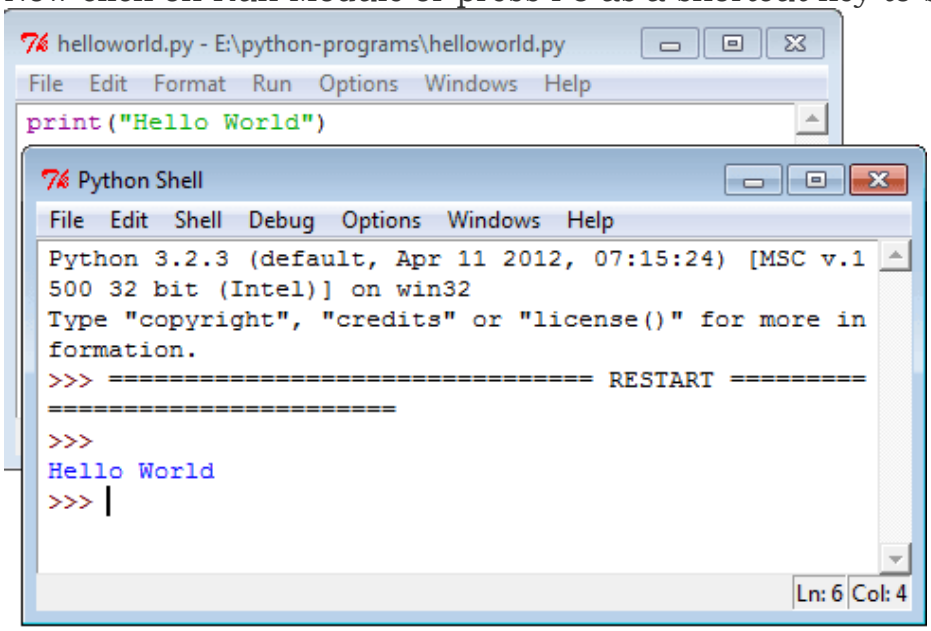
Let's save (Save command is located under the File menu) the file now. We save the program as helloworld.py under E:/python-programs folder.



To run the program select Run menu.

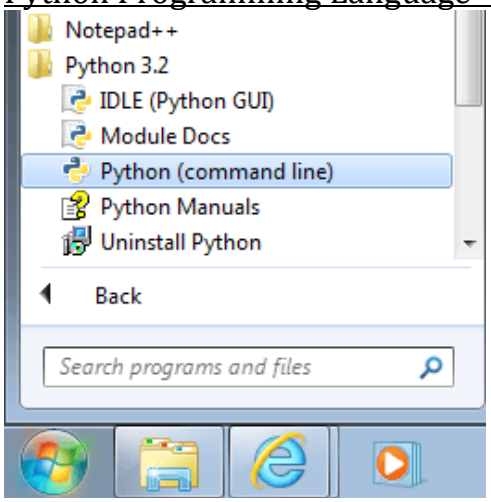


Now click on Run Module or press F5 as a shortcut key to see the result.

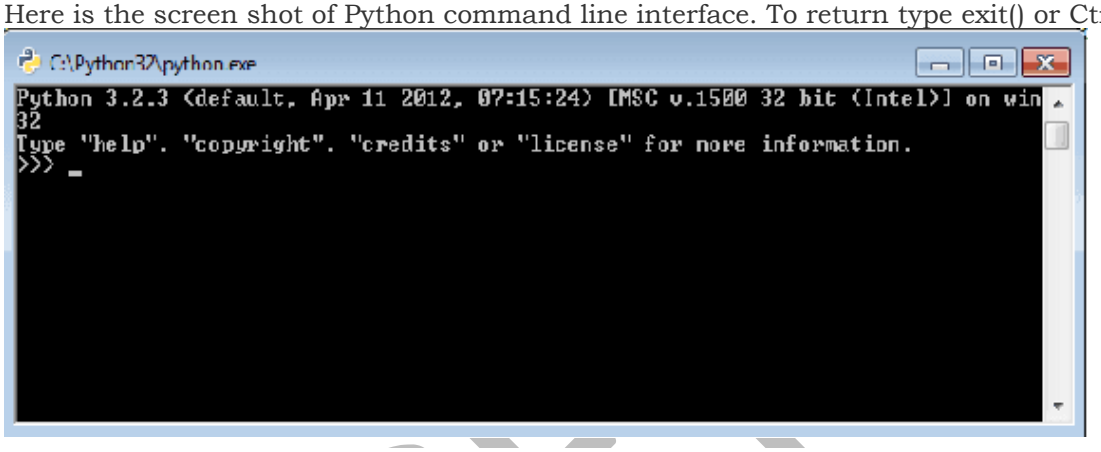


Python Command line interface:

There are some users who prefer command line intersection, rather than a GUI interface. To go Python command line, click on Python 3.2 tab then click on Python(command line).



Here is the screen shot of Python command line interface. To return type exit() or Ctrl+Z plus Enter key.



Python Variables

A variable is a memory location where a programmer can store a value. Example : roll_no, amount, name etc.

- Value is either string, numeric etc. Example : "Sara", 120, 25.36
- Variables are created when first assigned.
- Variables must be assigned before being referenced.
- The value stored in a variable can be accessed or updated later.
- No declaration required
- The type (string, int, float etc.) of the variable is determined by Python
- The interpreter allocates memory on the basis of the data type of a variable.

Variable name rules:

- Must begin with a letter (a - z, A - B) or underscore (_)
- Other characters can be letters, numbers or _
- Case Sensitive
- Can be any (reasonable) length
- There are some reserved words which you cannot use as a variable name because Python uses them for other things.

Python Assignment statements:

The assignment statement creates new variables and gives them values. Basic assignment statement in Python is :
Syntax

```
<variable> = <expr>
```

Where the equal sign (=) is used to assign value (right side) to a variable name (left side). See the following statements :

view plaincopy to clipboardprint?

```
>>> Item_name = "Computer" #A String
>>> Item_qty = 10 #An Integer
>>> Item_value = 1000.23 #A floating point
>>> print(Item_name)
Computer
>>> print(Item_qty)
10
>>> print(Item_value)
1000.23
>>>
```

One thing is important, assignment statement read right to left only.
Example :
a = 12 is correct, but 12 = a does not make sense to Python, which creates a syntax error. Check it in Python Shell.

view plaincopy to clipboardprint?

```
>>> a = 12
>>> 12 = a
SyntaxError: can't assign to literal
>>>
```

Multiple Assignment:
The basic assignment statement works for a single variable and a single expression. You can also assign a single value to more than one variables simultaneously.
Syntax

```
var1=var2=var3...varn= = <expr>
```

Example :

```
x = y = z = 1
```

Now check the individual value in Python Shell.

view plaincopy to clipboardprint?

```
>>> x = y = z = 1
>>> print(x)
1
>>> print(y)
1
>>> print(z)
1
>>>
```

Here is an another assignment statement where the variables assign many values at the same time.
Syntax

```
<var>, <var>, ..., <var> = <expr>, <expr>, ..., <expr>
```

Example :

```
x, y, z = 1, 2, "abcd"
```

In the above example x, y and z simultaneously get the new values 1, 2 and "abcd".

view plaincopy to clipboardprint?

```
>>> x,y,z = 1,2,"abcd"
>>> print(x)
1
>>> print(y)
2
>>> print(z)
abcd
```

You can reuse variable names by simply assigning a new value to them :

view plaincopy to clipboardprint?

```
>>> x = 100
>>> print(x)
100
>>> x = "Python"
>>> print(x)
Python
>>>
```

Swap variables:

Python swap values in a single line and this applies to all objects in python.

Syntax

```
var1, var2 = var2, var1
```

Example :

view plaincopy to clipboardprint?

```
>>> x = 10
>>> y = 20
>>> print(x)
10
>>> print(y)
20
>>> x, y = y, x
>>> print(x)
20
>>> print(y)
10
>>>
```

Local and global variables in python:

In Python, variables that are only referenced inside a function are implicitly global. If a variable is assigned a value anywhere within the function's body, it's assumed to be a local unless explicitly declared as global.

Example :

```
var1 = "Python"
def func1():
    var1 = "PHP"
    print("In side func1() var1 = ",var1)

def func2():
    print("In side func2() var1 = ",var1)
func1()
func2()
```

Output :

```
In side func1() var1 =  PHP
In side func2() var1 =  Python
```

You can use a global variable in other functions by declaring it as global keyword :

Example :

```
def func1():
    global var1
    var1 = "PHP"
    print("In side func1() var1 = ",var1)

def func2():
    print("In side func2() var1 = ",var1)
func1()
func2()
```

Output :

```
In side func1() var1 =  PHP
In side func2() var1 =  PHP
```

Python Reserved words/ Key words:

The following identifiers are used as reserved words of the language, and cannot be used as ordinary identifiers.

False	Class	finally	is	return
None	Continue	for	lambda	try
True	Def	from	nonlocal	while
and	Del	global	not	with
as	El	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Input and Output functions:

Python provides numerous built-in functions that are readily available to us at the python prompt.
Some of the functions like input() and print() are widely used for standard input and output operations respectively. Let us see the output section first.

Python Output Using print() function

We use the print() function to output data to the standard output device (screen).

```
print('This sentence is output to the screen')
# Output: This sentence is output to the screen
a = 5
print('The value of a is', a)
# Output: The value of a is 5
```

In the second print() statement, we can notice that a space was added between the string and the value of variable a.This is by default, but we can change it.
The actual syntax of the print() function is

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Here, objects is the value(s) to be printed.
The sep separator is used between the values. It defaults into a space character.
After all values are printed, end is printed. It defaults into a new line.
The file is the object where the values are printed and its default value is sys.stdout(screen). Here are an example to illustrate this.

```
print(1,2,3,4)
# Output: 1 2 3 4
print(1,2,3,4,sep='*')
# Output: 1*2*3*4
print(1,2,3,4,sep='#',end='&')
# Output: 1#2#3#4&
```

Output formatting:

Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method. This method is visible to any string object.

```
>>> x = 5; y = 10
>>> print('The value of x is {} and y is {}'.format(x,y))
The value of x is 5 and y is 10
```

Here the curly braces {} are used as placeholders. We can specify the order in which it is printed by using numbers (tuple index)

```
print('I love {0} and {1}'.format('bread','butter'))
# Output: I love bread and butter
print('I love {1} and {0}'.format('bread','butter'))
# Output: I love butter and bread
```

We can even use keyword arguments to format the string.

```
>>> print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))
Hello John, Goodmorning
```

We can even format strings like the old `sprintf()` style used in C programming language. We use the `%` operator to accomplish this.

```
>>> x = 12.3456789
>>> print('The value of x is %3.2f' %x)
The value of x is 12.35
>>> print('The value of x is %3.4f' %x)
The value of x is 12.3457
```

Python Input:

Up till now, our programs were static. The value of variables were defined or hard coded into the source code.

To allow flexibility we might want to take the input from the user. In Python, we have the `input()` function to allow this. The syntax for `input()` is

```
input([prompt])
```

where `prompt` is the string we wish to display on the screen. It is optional.

```
>>> num = input('Enter a number: ')
Enter a number: 10
>>> num
'10'
```

Here, we can see that the entered value `10` is a string, not a number. To convert this into a number we can use `int()` or `float()` functions.

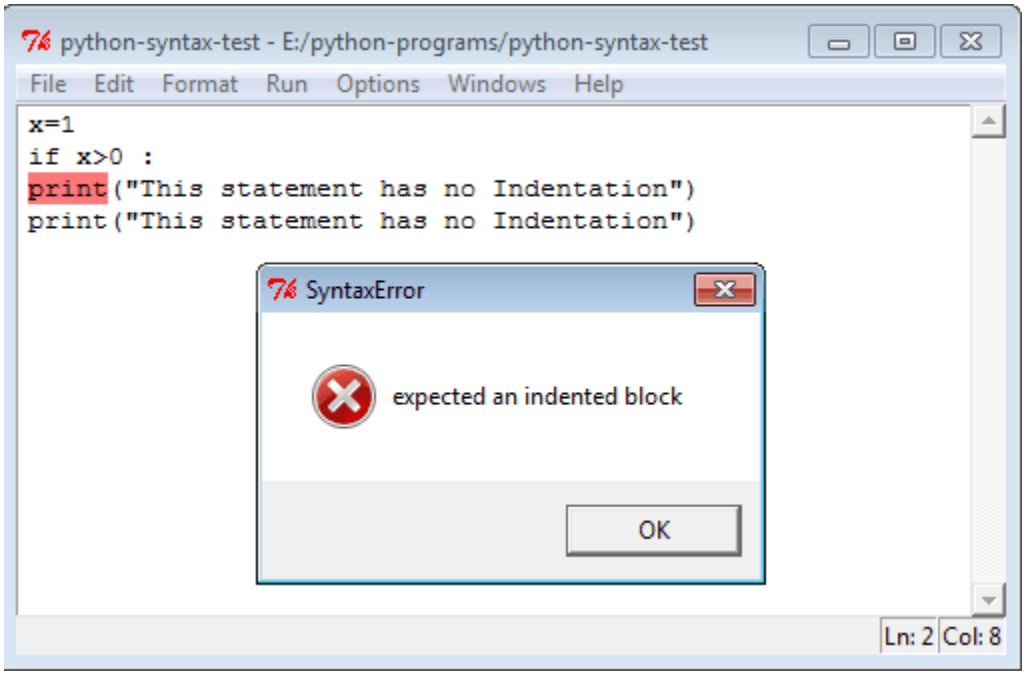
```
>>> int('10')
10
>>> float('10')
10.0
```

This same operation can be performed using the `eval()` function. But it takes it further. It can evaluate even expressions, provided the input is a string

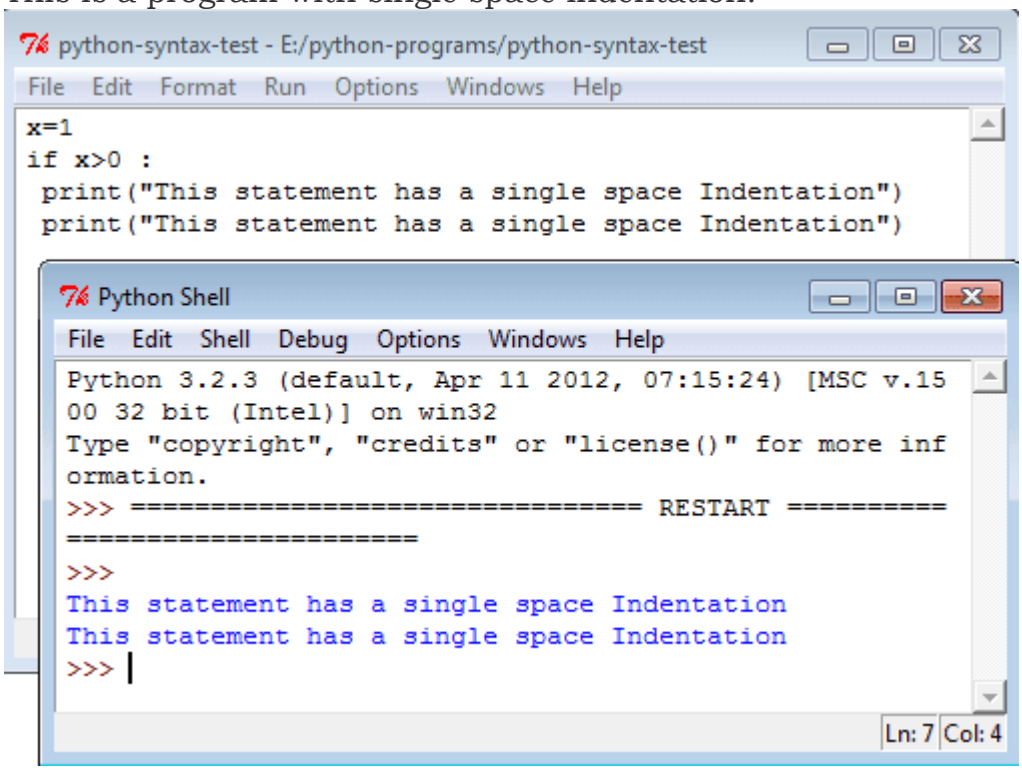
```
>>> int('2+3')
Traceback (most recent call last):
  File "<string>", line 301, in runcode
  File "<interactive input>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '2+3'
>>> eval('2+3')
5
```

Indentation:

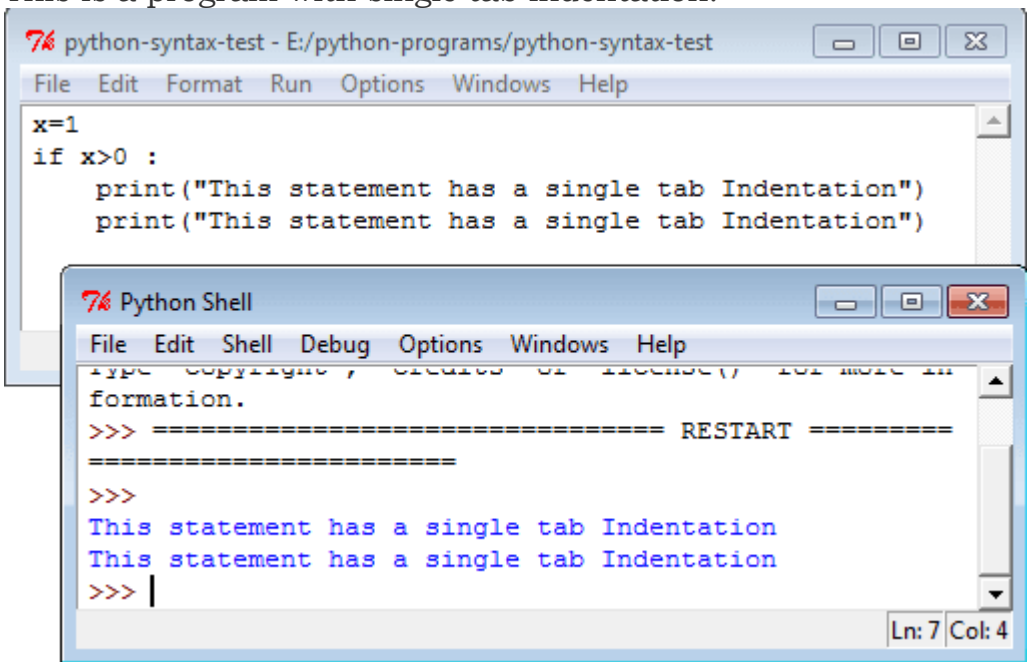
Python uses whitespace (spaces and tabs) to define program blocks whereas other languages like C, C++ use braces ({}) to indicate blocks of codes for class, functions or flow control. The number of whitespaces (spaces and tabs) in the indentation is not fixed, but all statements within the block must be the indented same amount. In the following program, the block statements have no indentation.



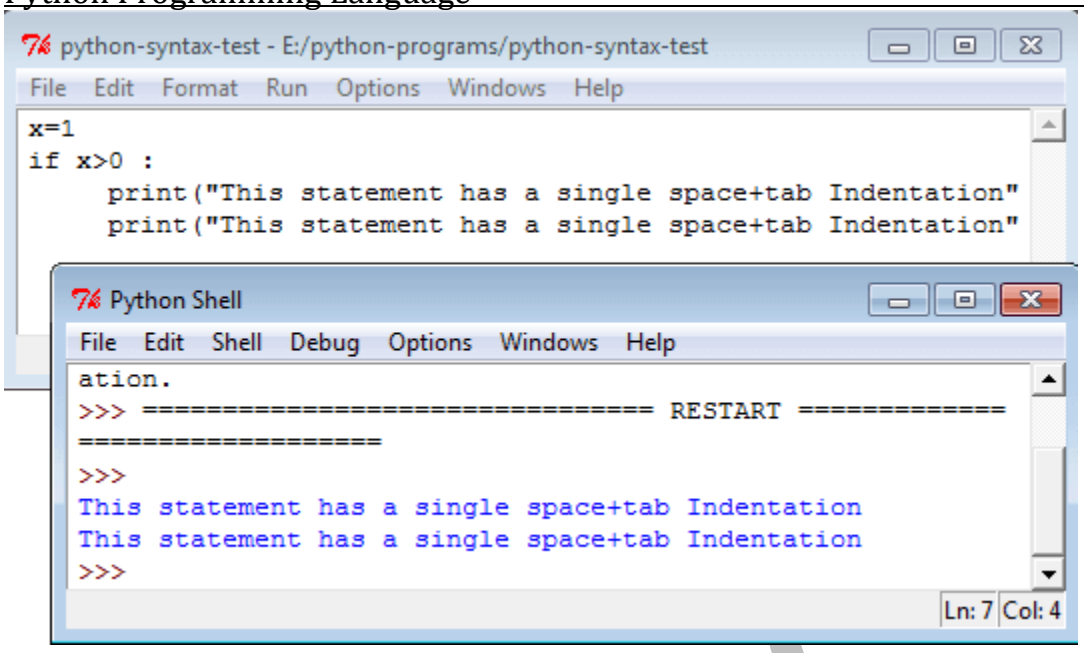
This is a program with single space indentation.



This is a program with single tab indentation.



Here is an another program with an indentation of a single space + a single tab.


python-syntax-test - E:/python-programs/python-syntax-test
File Edit Format Run Options Windows Help
x=1
if x>0 :
print("This statement has a single space+tab Indentation"
print("This statement has a single space+tab Indentation"
Python Shell
File Edit Shell Debug Options Windows Help
ation.
>>> ================================ RESTART ============
==================
>>>
This statement has a single space+tab Indentation
This statement has a single space+tab Indentation
>>>
Ln: 7 Col: 4