Python

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language.

Some more features –

- **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Installing Python

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python https://www.python.org/

Unix and Linux Installation

Here are the simple steps to install Python on Unix/Linux machine.

- Open a Web browser and go to https://www.python.org/downloads/.
- Follow the link to download zipped source code available for Unix/Linux.
- Download and extract files.

- Editing the *Modules/Setup* file if you want to customize some options.
- run ./configure script
- make
- make install

This installs Python at standard location */usr/local/bin* and its libraries at*/usr/local/lib/pythonXX* where XX is the version of Python.

Windows Installation

Here are the steps to install Python on Windows machine.

- Open a Web browser and go to https://www.python.org/downloads/.
- Follow the link for the Windows installer *python-XYZ.msi* file where XYZ is the version you need to install.
- To use this installer *python-XYZ.msi*, the Windows system must support Microsoft Installer 2.0. Save the installer file to your local machine and then run it to find out if your machine supports MSI.
- Run the downloaded file. This brings up the Python install wizard, which is really easy to use. Just accept the default settings, wait until the install is finished, and you are done.

Setting path at Unix/Linux

To add the Python directory to the path for a particular session in Unix −

- **In the csh shell** − type setenv PATH "$PATH:/usr/local/bin/python" and press Enter.
- **In the bash shell (Linux)** − type export ATH="$PATH:/usr/local/bin/python" and press Enter.
- **In the sh or ksh shell** − type PATH="$PATH:/usr/local/bin/python" and press Enter.

Setting path at Windows

To add the Python directory to the path for a particular session in Windows

At the command prompt – type path %path%;C:\Python and press Enter.

Note – C:\Python is the path of the Python directory

Running Python

You can navigate to IDLE IDE for python and start coding!

Getting Started –

Type the following text at the Python prompt and press Enter –

```
>>> print ("Hello, World!")
```

If you are running the older version of Python (Python 2.x), use of parenthesis as **inprint** function is optional. This produces the following result –

```
Hello, World!
```

Declaring Variables

Variables are nothing but reserved memory locations to store values. It means that when you create a variable, you reserve some space in the memory.

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string


print (counter)
print (miles)
print (name)
```

Here, 100, 1000.0 and "John" are the values assigned to counter, miles, and name variables, respectively. This produces the following result –

```
100
1000.0
John
```

Data Types in Python

Python has five standard data types –

- Numbers
- String
- List
- Tuple
- Dictionary

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example –

```
var1 = 1
var2 = 10
```

Python supports three different numerical types –

- int (signed integers)
- float (floating point real values)
- complex (complex numbers)

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows either pair of single or double quotes.

```
str = 'Hello World!'
print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th
print (str[2:])      # Prints string starting from 3rd character
print (str * 2)      # Prints string two times
print (str + "TEST") # Prints concatenated string
```

This will produce the following result –

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One of the differences between them is that all the items belonging to a list can be of different data type.

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print (list)          # Prints complete list
print (list[0])       # Prints first element of the list
print (list[1:3])     # Prints elements starting from 2nd till 3rd
print (list[2:])      # Prints elements starting from 3rd element
print (tinylist * 2)  # Prints list two times
print (list + tinylist) # Prints concatenated lists
```

This produces the following result –

```
['abcd', 786, 2.23, 'john', 70.200000000000003]
abcd
[786, 2.23]
[2.23, 'john', 70.200000000000003]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john']
```

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parenthesis.

The main difference between lists and tuples are – Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only** lists. For example –

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print (tuple)           # Prints complete tuple
print (tuple[0])        # Prints first element of the tuple
print (tuple[1:3])      # Prints elements starting from 2nd till 3rd
print (tuple[2:])       # Prints elements starting from 3rd element
print (tinytuple * 2)   # Prints tuple two times
print (tuple + tinytuple) # Prints concatenated tuple
```

This produces the following result –

```
('abcd', 786, 2.23, 'john', 70.200000000000003)
abcd
(786, 2.23)
(2.23, 'john', 70.200000000000003)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john')
```

Python Dictionary

Python's dictionaries are kind of hash-table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example –

```
dict = {}

dict['one'] = "This is one"

dict[2]     = "This is two"


tinydict = {'name': 'john','code':6734, 'dept': 'sales'}



print (dict['one'])       # Prints value for 'one' key

print (dict[2])           # Prints value for 2 key

print (tinydict)          # Prints complete dictionary

print (tinydict.keys())   # Prints all the keys

print (tinydict.values()) # Prints all the values
```

This produces the following result –

```
This is one
This is two
{'name': 'john', 'dept': 'sales', 'code': 6734}
dict_keys(['name', 'dept', 'code'])
dict_values(['john', 'sales', 6734])
```

Data Type Conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type-names as a function.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

S.No.	Function & Description
1	**int(x [,base])** Converts x to an integer. The base specifies the base if x is a string.
2	**float(x)** Converts x to a floating-point number.
3	**complex(real [,imag])** Creates a complex number.
4	**str(x)** Converts object x to a string representation.
5	**repr(x)** Converts object x to an expression string.
6	**eval(str)** Evaluates a string and returns an object.
7	**tuple(s)** Converts s to a tuple.
8	**list(s)** Converts s to a list.

9	**set(s)** Converts s to a set.
10	**dict(d)** Creates a dictionary. d must be a sequence of (key,value) tuples.
11	**frozenset(s)** Converts s to a frozen set.
12	**chr(x)** Converts an integer to a character.
13	**unichr(x)** Converts an integer to a Unicode character.
14	**ord(x)** Converts a single character to its integer value.
15	**hex(x)** Converts an integer to a hexadecimal string.
16	**oct(x)** Converts an integer to an octal string.

Python Arithmetic Operators

Assume variable **a** holds the value 10 and variable **b** holds the value 21, then

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 31
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -11
* Multiplication	Multiplies values on either side of the operator	a * b = 210
/ Division	Divides left hand operand by right hand operand	b / a = 2.1
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 1
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Operators in Python

Python Comparison Operators

These operators compare the values on either side of them and decide the relation among them. They are also called Relational operators.

Assume variable **a** holds the value 10 and variable **b** holds the value 20, then

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!= b) is true.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Assignment Operators

Assume variable **a** holds the value 10 and variable **b** holds the value 20, then

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent

		to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constants or variables or any other identifier names. All the Python keywords contain lowercase letters only.

and	exec	Not
as	finally	Or
assert	for	Pass
break	from	Print

class	global	Raise
continue	if	Return
def	import	Try
del	in	While
elif	is	With
else	lambda	Yield
except		

Decision Making – IF Loop

The IF statement is similar to that of other languages. The **if** statement contains a logical expression using which the data is compared and a decision is made based on the result of the comparison.

```
var1 = 100
if var1:
   print ("1 - Got a true expression value")
   print (var1)

var2 = 0
if var2:
   print ("2 - Got a true expression value")
   print (var2)
print ("Good bye!")
```

Output

When the above code is executed, it produces the following result –

```
1 - Got a true expression value
100
Good bye!
```

There may be a situation when you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested **if**construct.

In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

Example

```
# !/usr/bin/python3

num = int(input("enter number"))

if num%2 == 0:

   if num%3 == 0:

      print ("Divisible by 3 and 2")

   else:

      print ("divisible by 2 not divisible by 3")

else:

   if num%3 == 0:

      print ("divisible by 3 not divisible by 2")

   else:

      print  ("not Divisible by 2 not divisible by 3")
```

Output

When the above code is executed, it produces the following result –

```
enter number8

divisible by 2 not divisible by 3


enter number15

divisible by 3 not divisible by 2
```

```
enter number12
Divisible by 3 and 2

enter number5
not Divisible by 2 not divisible by 3
```

For Loop

Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

Example

range() generates an iterator to progress integers starting with 0 up to n-1. To obtain a list object of the sequence, it is type casted to list(). Now this list can be iterated using thefor statement.

```
>>> for var in list(range(5)):
   print (var)
```

Output

This will produce the following output.

```
0
1
2
3
4
```

While Loop

Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

Example

```
#!/usr/bin/python3


count = 0

while (count < 9):

   print ('The count is:', count)

   count = count + 1


print ("Good bye!")
```

Output

When the above code is executed, it produces the following result –

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

Reading Input from Keyboard

Python 2 has two versions of input functions, **input()** and **raw_input()**. The input() function treats the received data as string if it is included in quotes '' or "", otherwise the data is treated as number.

In Python 3, raw_input() function is deprecated. Further, the received data is always treated as string.

```
In Python 2


>>> x = input('something:')

something:10 #entered data is treated as number

>>> x
```

```
10


>>> x = input('something:')

something:'10' #eentered data is treated as string

>>> x

'10'


>>> x = raw_input("something:")

something:10 #entered data is treated as string even without ''

>>> x

'10'


>>> x = raw_input("something:")

something:'10' #entered data treated as string including ''

>>> x

"'10'"


In Python 3


>>> x = input("something:")

something:10

>>> x

'10'


>>> x = input("something:")

something:'10' #entered data treated as string with or without ''

>>> x

"'10'"


>>> x = raw_input("something:") # will result NameError

Traceback (most recent call last):

   File "<pyshell#3>", line 1, in

  <module>
```

```
   x = raw_input("something:")
NameError: name 'raw_input' is not defined
```

Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

```
# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print (str)
   return

# Now you can call printme function
printme("This is first call to the user defined function!")
printme("Again second call to the same function")
```

When the above code is executed, it produces the following result –

```
This is first call to the user defined function!
Again second call to the same function
```

File I/O Handling

Writing data to the file

```
#!/usr/bin/python3


# Open a file

fo = open("foo.txt", "w")

fo.write( "Python is a great language.\nYeah its great!!\n")


# Close opend file

fo.close()
```

The above method would create foo.txt file and would write given content in that file and finally it would close that file. If you would open this file, it would have the following content –

```
Python is a great language.
Yeah its great!!
```

Reading Data from a file

```
# Open a file

fo = open("foo.txt", "r+")

str = fo.read(10)

print ("Read String is : ", str)


# Close opened file
```

```
fo.close()
```

This produces the following result –

```
Read String is :  Python is
```

Object Oriented Programming

Python has been an object-oriented language since the time it existed. Due to this, creating and using classes and objects are downright easy.

Example

Following is an example of a simple Python class –

```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print "Total Employee %d" % Employee.empCount

   def displayEmployee(self):
      print "Name : ", self.name,  ", Salary: ", self.salary
```

- The variable *empCount* is a class variable whose value is shared among all the instances of a in this class. This can be accessed as *Employee.empCount* from inside the class or outside the class.

- The first method *__init__()* is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class.

- You declare other class methods like normal functions with the exception that the first argument to each method is *self*. Python adds the *self* argument to the list for you; you do not need to include it when you call the methods.

PyQt Tutorial – GUI Designing

Download PyQt4 python window binary from

http://www.lfd.uci.edu/~gohlke/pythonlibs/

PyQt is a GUI widgets toolkit. It is a Python interface for **Qt**, one of the most powerful, and popular cross-platform GUI library. PyQt is a blend of Python programming language and the Qt library. This introductory tutorial will assist you in creating graphical applications with the help of PyQt.

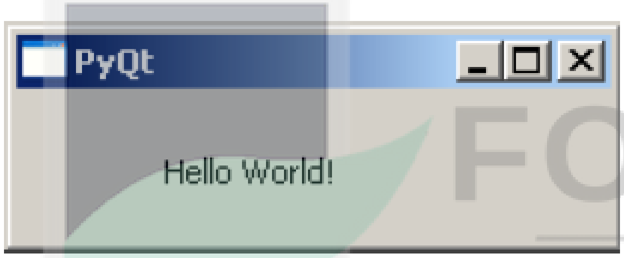
Example 1

Creating a simple GUI application using PyQt involves the following steps –

- Import QtGui module.
- Create an application object.
- A QWidget object creates top level window. Add QLabel object in it.
- Set the caption of label as “hello world”.
- Define the size and position of window by setGeometry() method.
- Enter the mainloop of application by **app.exec_()** method.

```
import sys
from PyQt4 import QtGui
app = QtGui.QApplication(sys.argv)
w = QtGui.QWidget()
b = QtGui.QLabel(w)
b.setText("Hello World!")
w.setGeometry(100,100,200,50)
b.move(50,20)
w.setWindowTitle(“PyQt”)
w.show()
sys.exit(app.exec_())
```

output –



Here is the list of Widgets which are frequently used.

Sr.No	Widgets & Description
1	**QLabel** A QLabel object acts as a placeholder to display non-editable text or image, or a movie of animated GIF. It can also be used as a mnemonic key for other widgets.
2	**QLineEdit** QLineEdit object is the most commonly used input field. It provides a box in which one line of text can be entered. In order to enter multi-line text, QTextEdit object is required.

3	**QPushButton** In PyQt API, the QPushButton class object presents a button which when clicked can be programmed to invoke a certain function.
4	**QRadioButton** A QRadioButton class object presents a selectable button with a text label. The user can select one of many options presented on the form. This class is derived from QAbstractButton class.
5	**QCheckBox** A rectangular box before the text label appears when a QCheckBox object is added to the parent window. Just as QRadioButton, it is also a selectable button.
6	**QComboBox** A QComboBox object presents a dropdown list of items to select from. It takes minimum screen space on the form required to display only the currently selected item.
7	**QSpinBox** A QSpinBox object presents the user with a textbox which displays an integer with up/down button on its right.
8	**QSlider Widget & Signal** QSlider class object presents the user with a groove over which a handle can be moved. It is a classic widget to control a bounded value.
9	**QMenuBar, QMenu & QAction** A horizontal QMenuBar just below the title bar of a QMainWindow object is reserved for displaying QMenu objects.
10	**QToolBar** A QToolBar widget is a movable panel consisting of text buttons, buttons with icons or other widgets.

11	**QInputDialog** This is a preconfigured dialog with a text field and two buttons, OK and Cancel. The parent window collects the input in the text box after the user clicks on Ok button or presses Enter.
12	**QFontDialog** Another commonly used dialog, a font selector widget is the visual appearance of QDialog class. Result of this dialog is a Qfont object, which can be consumed by the parent window.
13	**QFileDialog** This widget is a file selector dialog. It enables the user to navigate through the file system and select a file to open or save. The dialog is invoked either through static functions or by calling exec_() function on the dialog object.
14	**QTab** If a form has too many fields to be displayed simultaneously, they can be arranged in different pages placed under each tab of a Tabbed Widget. The QTabWidget provides a tab bar and a page area.
15	**QStacked** Functioning of QStackedWidget is similar to QTabWidget. It also helps in the efficient use of window's client area.
16	**QSplitter** If a form has too many fields to be displayed simultaneously, they can be arranged in different pages placed under each tab of a Tabbed Widget. The QTabWidget provides a tab bar and a page area.
17	**QDock** A dockable window is a subwindow that can remain in floating state or can be attached to the main window at a specified position. Main

	window object of QMainWindow class has an area reserved for dockable windows.
18	**QStatusBar** QMainWindow object reserves a horizontal bar at the bottom as the status bar. It is used to display either permanent or contextual status information.
19	**QList** QListWidget class is an item-based interface to add or remove items from a list. Each item in the list is a QListWidgetItem object. ListWidget can be set to be multiselectable.
20	**QScrollBar** A scrollbar control enables the user to access parts of the document that is outside the viewable area. It provides visual indicator to the current position.
21	**QCalendar** QCalendar widget is a useful date picker control. It provides a month-based view. The user can select the date by the use of the mouse or the keyboard, the default being today's date.

Cx_Freeze

cx_Freeze normally produces a folder containing an executable file for your program, along with the shared libraries (DLLs or .so files) needed to run it. You can make a simple Windows installer using a setup script with the bdist_msi option, or a Mac disk image with bdist_dmg. For a more advanced Windows installer, use a separate tool like Inno Setup to package the files cx_Freeze collects.

Python modules for your executables are stored in a zip file. Packages are stored in the file system by default but can also be included in the zip file.

```
import sys
from cx_Freeze import setup, Executable


# GUI applications require a different base on Windows (the default is for a
# console application).

setup(executables = [Executable("guifoo.py")])
```

The script is invoked as follows:

```
python setup.py build
```

This command will create a subdirectory called build with a further subdirectory starting with the letters exe. and ending with the typical identifier for the platform that distutils uses. This allows for multiple platforms to be built without conflicts.

Openpyxl

Openpyxl is used to read and write data to excel file in python

Sample code for writing data to Excel File. Openpyxl so not support xls files, it supports xlsx file.

```
Import openpyxl
wb=openpyxl.load_workbook('file.xlsx')
sheet=wb.get_sheet_by_name('Sheet1')
sheet.cell(row=2,column=1).value='Python'
sheet.cell(row=2,column=2).value='Hello'
wb.save('file.xlsx')
```

Theano

Theano is a module used for Machine Learning, A sample code for single layer perceptron training is –

```
import theano
import numpy as np



x=theano.tensor.fvector('x')
w=theano.shared(np.asarray([0.2,0.3]),'w')
target=theano.tensor.fvector('target')
y=(x*w).sum()

cost=theano.tensor.sqr(target[0]-y)

gradi=theano.tensor.grad(cost,[w])
wn=w-(0.1*gradi[0])
updates=[(w,wn)]
```

```
f=theano.function([x,target],y,updates=updates)


for i in range(0,10):
    output=f([1.0,1.0],[20.0])
    print output
```

Happy Coding!