**Python Tutorial**

Python is a simple, easy to learn, powerful, high level and object-oriented programming language.

Python is an interpreted scripting language also. Guido Van Rossum is known as the founder of python programming.

[**Introduction to Python**](https://www.javatpoint.com/what-is-python)

It covers the topics such as python programming, features, history, versions, how to install, example, how to execute, variables, keywords, identifiers, literals, operators and comments.

[**Control Statement**](https://www.javatpoint.com/python-if-statement)

The control statement in python covers if statement, for loop, while loop, do while loop, break statement, continue statement and pass statement.

[**Python Strings**](https://www.javatpoint.com/python-strings)

The string chapter in python provides the full functionality to work on strings such as accessing string, applying string operators, details of slice notation, applying different functions etc.

[**Python Lists**](https://www.javatpoint.com/python-lists)

The list chapter in python covers the data structure part such as storing data in list, accessing data, manipulating data etc.

[**Python Tuples**](https://www.javatpoint.com/python-tuples)

A sequence of immutable objects is known as tuple, it covers accessing tuple, adding tuple, replacating tuple, updating tuple etc.

[**Python Dictionary**](https://www.javatpoint.com/python-dictionary)

The python dictionary provides details about dictionary operations.

[**Python Functions**](https://www.javatpoint.com/python-functions)

It provides a list of python functions with its implementations.

[**Python Files I/O**](https://www.javatpoint.com/python-files-io)

How to write data into file and read data from file in python?

[**Python Modules**](https://www.javatpoint.com/python-modules)

What is python module? What are the usage of modules?

[**Python Exceptions**](https://www.javatpoint.com/python-exception-handling)

It explains the errors and exceptions in python.

# What is Python

**Python** is an object-oriented, high level language, interpreted, dynamic and multipurpose programming language.

Python is *easy to learn* yet powerful and versatile scripting language which makes it attractive for Application Development.

Python's syntax and *dynamic typing* with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas.

Python supports *multiple programming pattern*, including object oriented programming, imperative and functional programming or procedural styles.

Python is not intended to work on special area such as web programming. That is why it is known as *multipurpose* because it can be used with web, enterprise, 3D CAD etc.

We don't need to use data types to declare variable because it is *dynamically typed* so we can write a=10 to declare an integer value in a variable.

Python makes the development and debugging *fast* because there is no compilation step included in python development and edit-test-debug cycle is very fast.

**Python Features**

There are a lot of features provided by python programming language.

**1) Easy to Use:**

Python is easy to very easy to use and high level language. Thus it is programmer-friendly language.

**2) Expressive Language:**

Python language is more expressive. The sense of expressive is the code is easily understandable.

**3) Interpreted Language:**

Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

**4) Cross-platform language:**

Python can run equally on different platforms such as Windows, Linux, Unix , Macintosh etc. Thus, Python is a portable language.

**5) Free and Open Source:**

Python language is freely available(www.python.org).The source-code is also available. Therefore it is open source.

**6) Object-Oriented language:**

Python supports object oriented language. Concept of classes and objects comes into existence.

**7) Extensible:**

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in your python code.

**8) Large Standard Library:**

Python has a large and broad library.

**9) GUI Programming:**

Graphical user interfaces can be developed using Python.

**10) Integrated:**

It can be easily integrated with languages like C, C++, JAVA etc.

**Python History**

* Python laid its foundation in the late 1980s.
* The implementation of Python was started in the December 1989 by **Guido Van Rossum** at CWI in Netherland.
* *ABC programming language* is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.
* Python is influenced by programming languages like:
  + ABC language.
  + Modula-3
* **Python Version**
* Python programming language is being updated regularly with new features and support. There are a lot of updation in python versions, started from 1994 to current date.
* A list of python versions with its released date is given below.

|  |  |
| --- | --- |
| **Python Version** | **Released Date** |
| Python 1.0 | January 1994 |
| Python 1.5 | December 31, 1997 |
| Python 1.6 | September 5, 2000 |
| Python 2.0 | October 16, 2000 |
| Python 2.1 | April 17, 2001 |
| Python 2.2 | December 21, 2001 |
| Python 2.3 | July 29, 2003 |
| Python 2.4 | November 30, 2004 |
| Python 2.5 | September 19, 2006 |
| Python 2.6 | October 1, 2008 |
| Python 2.7 | July 3, 2010 |
| Python 3.0 | December 3, 2008 |
| Python 3.1 | June 27, 2009 |
| Python 3.2 | February 20, 2011 |
| Python 3.3 | September 29, 2012 |

**Python Applications**

Python as a whole can be used in any sphere of development.

Let us see what are the major regions where Python proves to be handy.

**1) Console Based Application**

Python can be used to develop console based applications. For example: **IPython**.

**2) Audio or Video based Applications**

Python proves handy in multimedia section. Some of real applications are: TimPlayer, cplay etc.

**3) 3D CAD Applications**

Fandango is a real application which provides full features of CAD.

**4) Web Applications**

Python can also be used to develop web based application. Some important developments are: PythonWikiEngines, Pocoo, PythonBlogSoftware etc.

**5) Enterprise Applications**

Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.

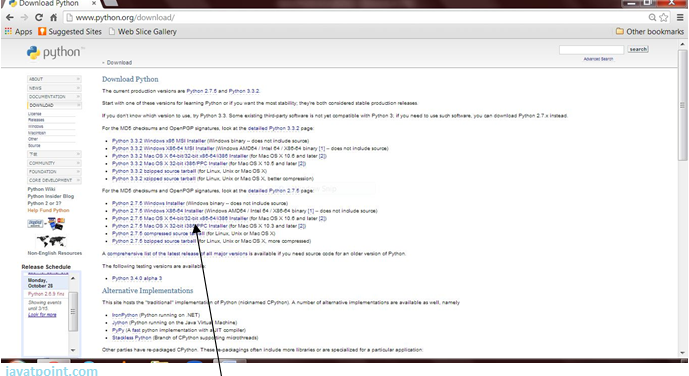
**6) Applications for Images**

Using Python several application can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

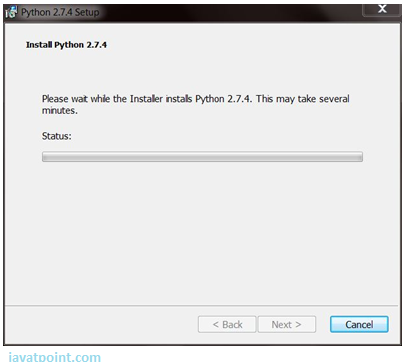
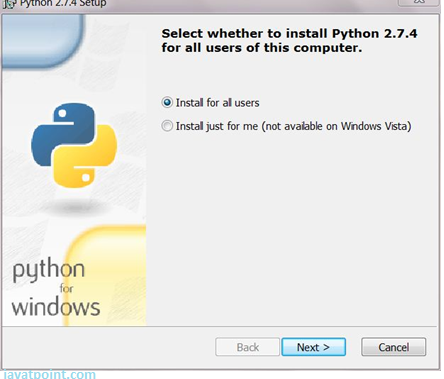
There are several such applications which can be developed using Python

**HOW TO INSTALL PYTHON**

1. To install Python, firstly download the Python distribution from www.python.org/download.



2. Having downloaded the Python distribution now execute it. Double click on the downloaded software. Follow the steps for installation:



Click the Finish button and Python will be installed on your system.

**SETTING PATH IN PYTHON**

Before starting working with Python, a specific path is to set.

* Your Python program and executable code can reside in any directory of your system, therefore Operating System provides a specific search path that index the directories Operating System should search for executable code.
* The Path is set in the Environment Variable of My Computer properties:
* To set path follow the steps:

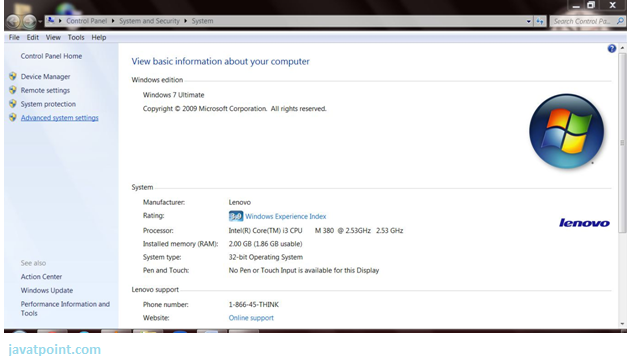
Right click on My Computer ->Properties ->Advanced System setting ->Environment Variable ->New

In Variable name write path and in Variable value copy path up to C://Python(i.e., path where Python is installed). Click Ok ->Ok.

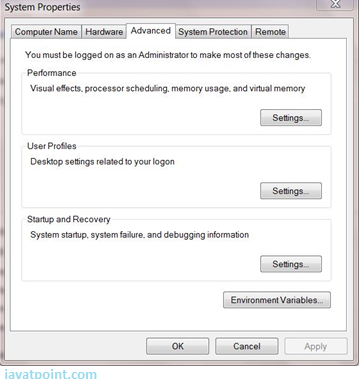
Path will be set for executing Python programs.

1. Right click on My Computer and click on properties.

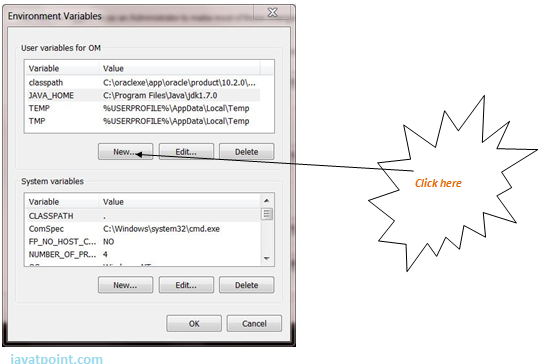
2. Click on Advanced System settings



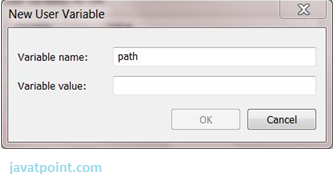
3. Click on Environment Variable tab.



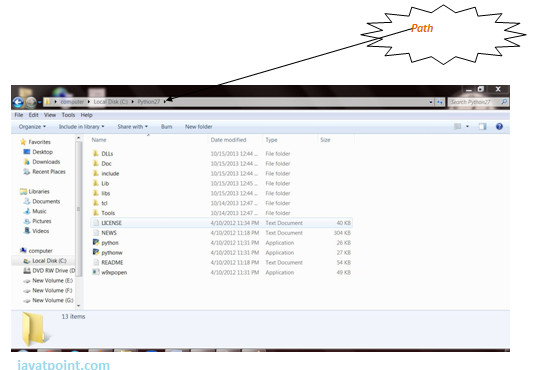
4. Click on new tab of user variables.



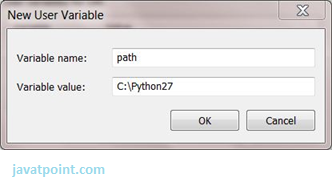
5. Write path in variable name



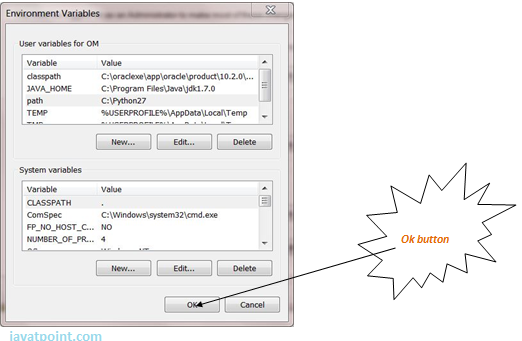
6. Copy the path of Python folder



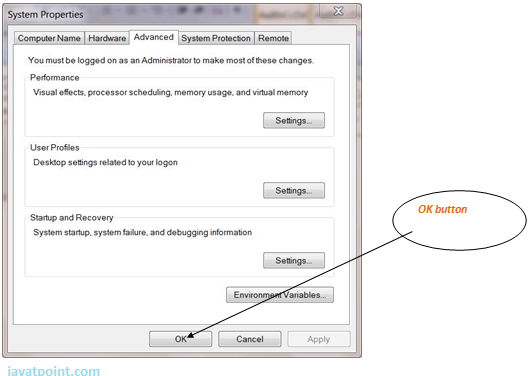
7. Paste path of Python in variable value.



8. Click on Ok button:



9. Click on Ok button:



# Python Example

Python code is simple and easy to run. Here is a simple Python code that will print "Welcome to Python".

A simple python example is given below.

1. >>> a="Welcome To Python"
2. >>> print a
3. Welcome To Python
4. >>>

**Explanation:**

* Here we are using IDLE to write the Python code. Detail explanation to run code is given in Execute Python section.
* A variable is defined named "a" which holds "Welcome To Python".
* "print" statement is used to print the content. Therefore "print a" statement will print the content of the variable. Therefore, the output "Welcome To Python" is produced.

## Python 3.4 Example

In python 3.4 version, you need to add parenthesis () in a string code to print it.

1. >>> a=("Welcome To Python Example")
2. >>> print a
3. Welcome To Python Example
4. >>>

# How to execute python

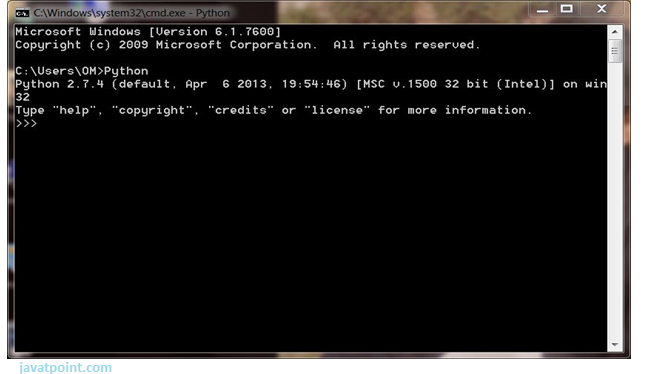
There are three different ways of working in Python:

## 1) Interactive Mode:

You can enter python in the command prompt and start working with Python.

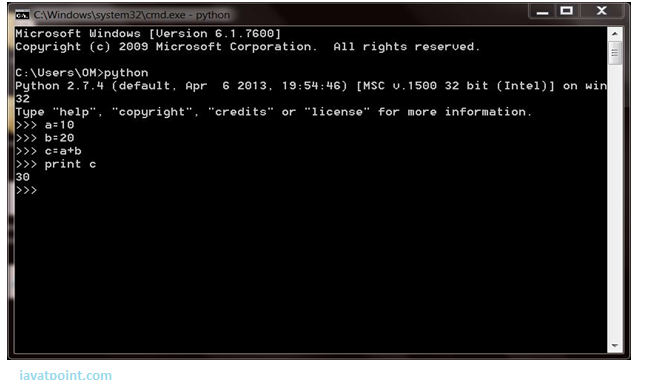


Press Enter key and the Command Prompt will appear like:



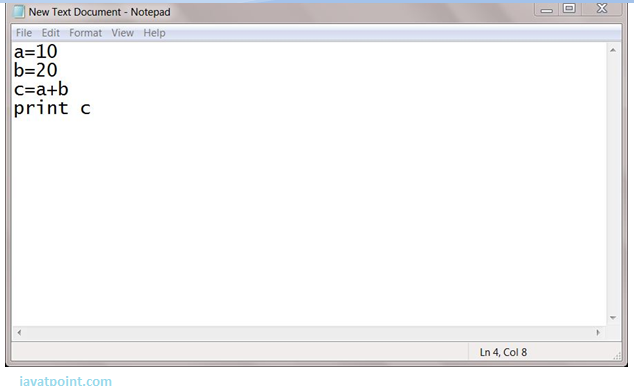
Now you can execute your Python commands.

**Eg:**

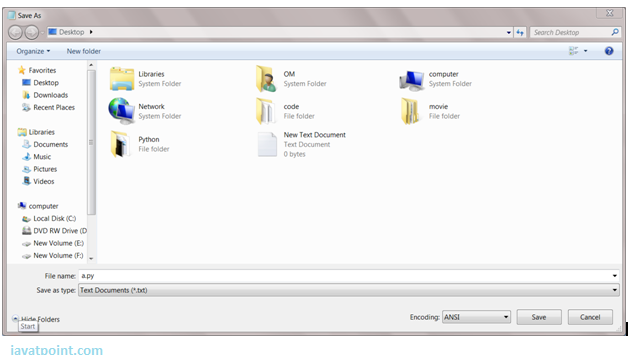


## 2) Script Mode:

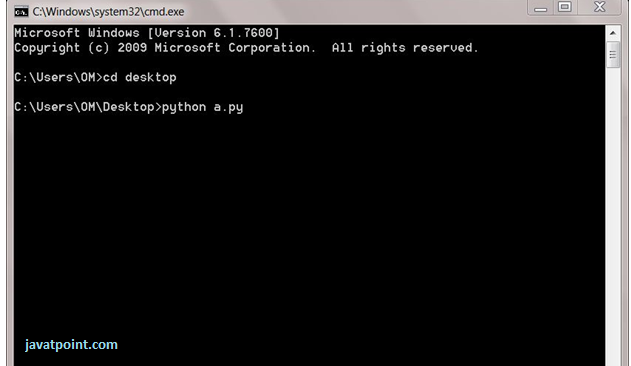
Using Script Mode , you can write your Python code in a separate file using any editor of your Operating System.



Save it by .py extension.



Now open Command prompt and execute it by :



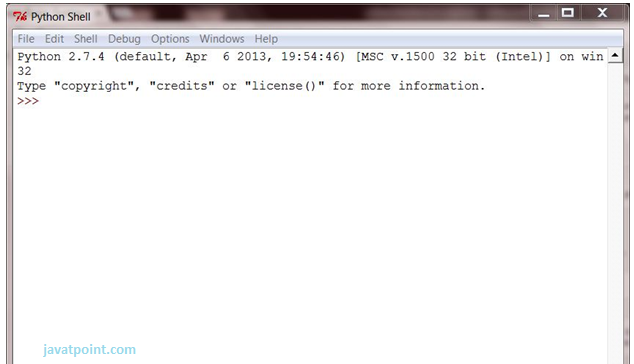
NOTE: Path in the command prompt should be where you have saved your file. In the above case file should be saved at desktop.

## 3) Using IDE: (Integrated Development Environment)

You can execute your Python code using a Graphical User Interface (GUI).

All you need to do is:

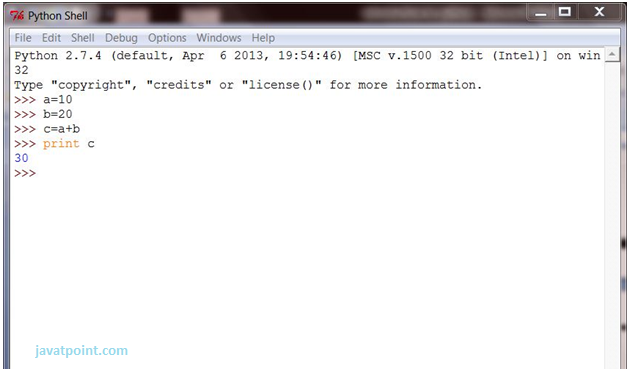
Click on Start button -> All Programs -> Python -> IDLE(Python GUI)



You can use both Interactive as well as Script mode in IDE.

**1) Using Interactive mode:**

Execute your Python code on the Python prompt and it will display result simultaneously.

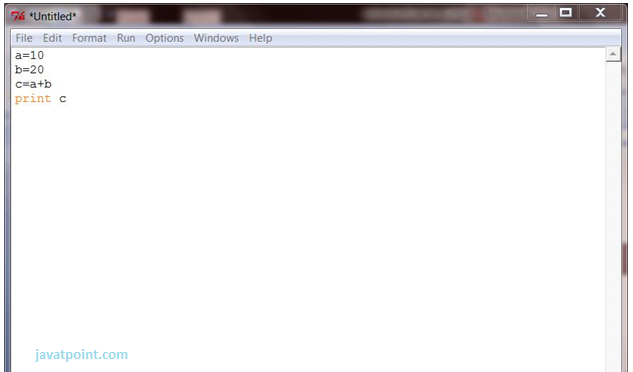


**2) Using Script Mode:**

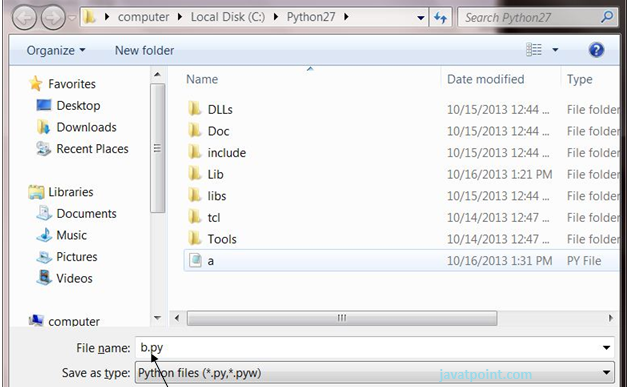
i) Click on Start button -> All Programs -> Python -> IDLE(Python GUI)

ii) Python Shell will be opened. Now click on File -> New Window.

A new Editor will be opened . Write your Python code here.



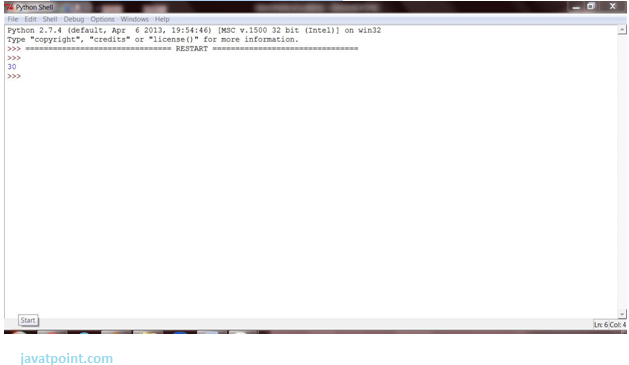
Click on file -> save as



Run then code by clicking on Run in the Menu bar.

Run -> Run Module

Result will be displayed on a new Python shell as:



# Python Variables

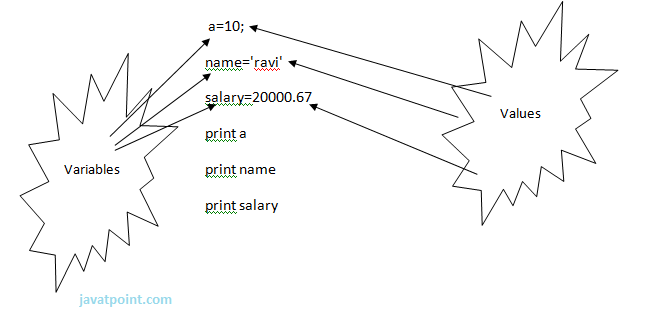
Variable is a name of the memory location where data is stored. Once a variable is stored that means a space is allocated in memory.

## Assigning values to Variable:

We need not to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

The assignment is done using the equal (=) operator.

**Eg:**



**Output:**

1. >>>
2. 10
3. ravi
4. 20000.67
5. >>>

## Multiple Assignment:

Multiple assignment can be done in Python at a time.

There are two ways to assign values in Python:

**1. Assigning single value to multiple variables:**

**Eg:**

1. x=y=z=50
2. print x
3. print y
4. print z

**Output:**

1. >>>
2. 50
3. 50
4. 50
5. >>>

**2.Assigning multiple values to multiple variables:**

**Eg:**

1. a,b,c=5,10,15
2. print a
3. print b
4. print c

**Output:**

1. >>>
2. 5
3. 10
4. 15
5. >>>

The values will be assigned in the order in which variables appears.

### Basic Fundamentals:

This section contains the basic fundamentals of Python like :

**i)Tokens and their types.**

**ii) Comments**

**a)Tokens:**

* Tokens can be defined as a punctuator mark, reserved words and each individual word in a statement.
* Token is the smallest unit inside the given program.

There are following tokens in Python:

* Keywords.
* Identifiers.
* Literals.
* Operators.

### Tuples:

* Tuple is another form of collection where different type of data can be stored.
* It is similar to list where data is separated by commas. Only the difference is that list uses square bracket and tuple uses parenthesis.
* Tuples are enclosed in parenthesis and cannot be changed.

**Eg:**

1. >>> tuple=('rahul',100,60.4,'deepak')
2. >>> tuple1=('sanjay',10)
3. >>> tuple
4. ('rahul', 100, 60.4, 'deepak')
5. >>> tuple[2:]
6. (60.4, 'deepak')
7. >>> tuple1[0]
8. 'sanjay'
9. >>> tuple+tuple1
10. ('rahul', 100, 60.4, 'deepak', 'sanjay', 10)
11. >>>

### Dictionary:

* Dictionary is a collection which works on a key-value pair.
* It works like an associated array where no two keys can be same.
* Dictionaries are enclosed by curly braces ({}) and values can be retrieved by square bracket([]).

**Eg:**

1. >>> dictionary={'name':'charlie','id':100,'dept':'it'}
2. >>> dictionary
3. {'dept': 'it', 'name': 'charlie', 'id': 100}
4. >>> dictionary.keys()
5. ['dept', 'name', 'id']
6. >>> dictionary.values()
7. ['it', 'charlie', 100]
8. >>>

**Python Keywords**

Keywords are special reserved words which convey a special meaning to the compiler/interpreter. Each keyword have a special meaning and a specific operation. List of Keywords used in Python are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| True | False | None | and | as |
| asset | def | class | continue | break |
| else | finally | elif | del | except |
| global | for | if | from | import |
| raise | try | or | return | pass |
| nonlocal | in | not | is | lambda |

**Identifiers**

Identifiers are the names given to the fundamental building blocks in a program.

These can be variables ,class ,object ,functions , lists , dictionaries etc.

There are certain rules defined for naming i.e., Identifiers.

I. An identifier is a long sequence of characters and numbers.

II.No special character except underscore ( \_ ) can be used as an identifier.

III.Keyword should not be used as an identifier name.

IV.Python is case sensitive. So using case is significant.

V.First character of an identifier can be character, underscore ( \_ ) but not digit.

**Python Literals**

Literals can be defined as a data that is given in a variable or constant.

Python support the following literals:

**I. String literals:**

String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes for a String.

**Eg:**

"Aman" , '12345'

**Types of Strings:**

There are two types of Strings supported in Python:

a).Single line String- Strings that are terminated within a single line are known as Single line Strings.

**Eg:**

1. >>> text1='hello'

b).Multi line String- A piece of text that is spread along multiple lines is known as Multiple line String.

There are two ways to create Multiline Strings:

**1). Adding black slash at the end of each line.**

**Eg:**

1. >>> text1='hello\
2. user'
3. >>> text1
4. 'hellouser'
5. >>>

**2).Using triple quotation marks:-**

**Eg:**

1. >>> str2='''''welcome
2. to
3. SSSIT'''
4. >>> print str2
5. welcome
6. to
7. SSSIT
8. >>>

**II.Numeric literals:**

Numeric Literals are immutable. Numeric literals can belong to following four different numerical types.

|  |  |  |  |
| --- | --- | --- | --- |
| **Int(signed integers)** | **Long(long integers)** | **float(floating point)** | **Complex(complex)** |
| Numbers( can be both positive and negative) with no fractional part.eg: 100 | Integers of unlimited size followed by lowercase or uppercase L eg: 87032845L | Real numbers with both integer and fractional part eg: -26.2 | In the form of a+bj where a forms the real part and b forms the imaginary part of complex number. eg: 3.14j |

**III. Boolean literals:**

A Boolean literal can have any of the two values: True or False.

**IV. Special literals.**

Python contains one special literal i.e., None.

None is used to specify to that field that is not created. It is also used for end of lists in Python.

Eg:

1. >>> val1=10
2. >>> val2=None
3. >>> val1
4. 10
5. >>> val2
6. >>> print val2
7. None
8. >>>

**V.Literal Collections.**

Collections such as tuples, lists and Dictionary are used in Python.

**List:**

* List contain items of different data types. Lists are mutable i.e., modifiable.
* The values stored in List are separated by commas(,) and enclosed within a square brackets([]). We can store different type of data in a List.
* Value stored in a List can be retrieved using the slice operator([] and [:]).
* The plus sign (+) is the list concatenation and asterisk(\*) is the repetition operator.

**Eg:**

1. >>> list=['aman',678,20.4,'saurav']
2. >>> list1=[456,'rahul']
3. >>> list
4. ['aman', 678, 20.4, 'saurav']
5. >>> list[1:3]
6. [678, 20.4]
7. >>> list+list1
8. ['aman', 678, 20.4, 'saurav', 456, 'rahul']
9. >>> list1\*2
10. [456, 'rahul', 456, 'rahul']
11. >>>

**Python Operators**

Operators are particular symbols which operate on some values and produce an output.

The values are known as Operands.

**Eg:**

1. 4 + 5 = 9

Here 4 and 5 are Operands and (+) , (=) signs are the operators. They produce the output 9.

**Python supports the following operators:**

1. Arithmetic Operators.
2. Relational Operators.
3. Assignment Operators.
4. Logical Operators.
5. Membership Operators.
6. Identity Operators.
7. Bitwise Operators.

**Arithmetic Operators:**

|  |  |
| --- | --- |
| **Operators** | **Description** |
| // | Perform Floor division(gives integer value after division) |
| + | To perform addition |
| - | To perform subtraction |
| \* | To perform multiplication |
| / | To perform division |
| % | To return remainder after division(Modulus) |
| \*\* | Perform exponent(raise to power) |

**eg:**

1. >>> 10+20
2. 30
3. >>> 20-10
4. 10
5. >>> 10\*2
6. 20
7. >>> 10/2
8. 5
9. >>> 10%3
10. 1
11. >>> 2\*\*3
12. 8
13. >>> 10//3
14. 3
15. >>>

**Relational Operators:**

|  |  |
| --- | --- |
| **Operators** | **Description** |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| == | Equal to |
| != | Not equal to |
| <> | Not equal to(similar to !=) |

**eg:**

1. >>> 10<20
2. True
3. >>> 10>20
4. False
5. >>> 10<=10
6. True
7. >>> 20>=15
8. True
9. >>> 5==6
10. False
11. >>> 5!=6
12. True
13. >>> 10<>2
14. True
15. >>>

**Assignment Operators:**

|  |  |
| --- | --- |
| **Operators** | **Description** |
| = | Assignment |
| /= | Divide and Assign |
| += | Add and assign |
| -= | Subtract and Assign |
| \*= | Multiply and assign |
| %= | Modulus and assign |
| \*\*= | Exponent and assign |
| //= | Floor division and assign |

**eg:**

1. >>> c=10
2. >>> c
3. 10
4. >>> c+=5
5. >>> c
6. 15
7. >>> c-=5
8. >>> c
9. 10
10. >>> c\*=2
11. >>> c
12. 20
13. >>> c/=2
14. >>> c
15. 10
16. >>> c%=3
17. >>> c
18. 1
19. >>> c=5
20. >>> c\*\*=2
21. >>> c
22. 25
23. >>> c//=2
24. >>> c
25. 12
26. >>>

**Logical Operators:**

|  |  |
| --- | --- |
| **Operators** | **Description** |
| and | Logical AND(When both conditions are true output will be true) |
| or | Logical OR (If any one condition is true output will be true) |
| not | Logical NOT(Compliment the condition i.e., reverse) |

**eg:**

1. a=5>4 and 3>2
2. print a
3. b=5>4 or 3<2
4. print b
5. c=not(5>4)
6. print c

**Output:**

1. >>>
2. True
3. True
4. False
5. >>>

**Membership Operators:**

|  |  |
| --- | --- |
| **Operators** | **Description** |
| in | Returns true if a variable is in sequence of another variable, else false. |
| not in | Returns true if a variable is not in sequence of another variable, else false. |

**eg:**

1. a=10
2. b=20
3. list=[10,20,30,40,50];
4. if (a in list):
5. print "a is in given list"
6. else:
7. print "a is not in given list"
8. if(b not in list):
9. print "b is not given in list"
10. else:
11. print "b is given in list"

**Output:**

1. >>>
2. a is in given list
3. b is given in list
4. >>>

**Identity Operators:**

|  |  |
| --- | --- |
| **Operators** | **Description** |
| is | Returns true if identity of two operands are same, else false |
| is not | Returns true if identity of two operands are not same, else false. |

**Example:**

1. a=20
2. b=20
3. if( a is b):
4. print  ?a,b have same identity?
5. else:
6. print ?a, b are different?
7. b=10
8. if( a is not b):
9. print  ?a,b have different identity?
10. else:
11. print ?a,b have same identity?

**Output:**

1. >>>
2. a,b have same identity
3. a,b have different identity
4. >>>

**Python Comments**

Python supports two types of comments:

**1) Single lined comment:**

In case user wants to specify a single line comment, then comment must start with ?#?

**Eg:**

1. # This is single line comment.

**2) Multi lined Comment:**

Multi lined comment can be given inside triple quotes.

**eg**

1. ''''' This
2. Is
3. Multipline comment'''

**eg:**

1. #single line comment
2. print "Hello Python"
3. '''''This is
4. multiline comment''

'

**Python If Statements**

The if statement in python is same as c language which is used test a condition. If condition is true, statement of if block is executed otherwise it is skipped.

**Syntax of python if statement:**

1. if(condition):
2. statements

**Example of if statement in python**

1. a=10
2. if a==10:
3. print  "Hello User"

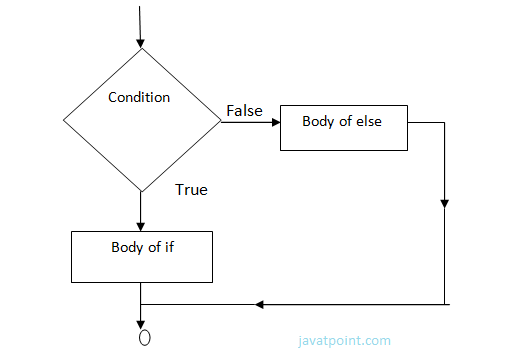
**Output:**

Hello User

**Python If Else Statements**

**Syntax:**

1. if(condition):  False
2. statements
3. else:   True
4. statements



**Example-**

1. year=2000
2. if year%4==0:
3. print  "Year is Leap"
4. else:
5. print "Year is not Leap"

**Output:**

Year is Leap

**Nested If Else Statement:**

When we need to check for multiple conditions to be true then we use elif Statement.

This statement is like executing a if statement inside a else statement.

**Syntax:**

1. If statement:
2. Body
3. elif statement:
4. Body
5. else:
6. Body

**Example:**

1. a=10
2. if a>=20:
3. print "Condition is True"
4. else:
5. if a>=15:
6. print "Checking second value"
7. else:
8. print "All Conditions are false"

**Output:**

All Conditions are false.

# For Loop

for Loop is used to iterate a variable over a sequence(i.e., list or string) in the order that they appear.

**Syntax:**

1. for <variable> in <sequence>:

**Output:**

1. 1
3. 7
5. 9

**Explanation:**

* Firstly, the first value will be assigned in the variable.
* Secondly all the statements in the body of the loop are executed with the same value.
* Thirdly, once step second is completed then variable is assigned the next value in the sequence and step second is repeated.
* Finally, it continues till all the values in the sequence are assigned in the variable and processed.

**Program to display table of Number:**

1. num=2
2. for a in range (1,6):
3. print  num \* a

**Output:**

1. 2
3. 4
5. 6
7. 8
9. 10

**Program to find sum of Natural numbers from 1 to 10.**

1. sum=0
2. for n in range(1,11):
3. sum+=n
4. print sum

**Output:**

1. 55

## Nested Loops

Loops defined within another Loop is called Nested Loop.

When an outer loop contains an inner loop in its body it is called Nested Looping.

**Syntax:**

1. for  <expression>:
2. for <expression>:
3. Body

**eg:**

1. for i in range(1,6):
2. for j in range (1,i+1):
3. print i,
4. print

**Output:**

1. >>>
2. 1
3. 2 2
4. 3 3 3
5. 4 4 4 4
6. 5 5 5 5 5
7. >>>

**Explanation:**

For each value of Outer loop the whole inner loop is executed.

For each value of inner loop the Body is executed each time.

**Program to print Pyramid:**

1. for i in range (1,6):
2. for j in range (5,i-1,-1):
3. print "\*",
4. print

**Output:**

1. >>>
2. \* \* \* \* \*
3. \* \* \* \*
4. \* \* \*
5. \* \*
6. \*

**While Loop**

while Loop is used to execute number of statements or body till the condition passed in while is true. Once the condition is false, the control will come out of the loop.

**Syntax:**

1. while <expression>:
2. Body

Here, body will execute multiple times till the expression passed is true. The Body may be a single statement or multiple statement.

**Eg:**

1. a=10
2. while a>0:
3. print "Value of a is",a
4. a=a-2

print "Loop is Completed"

**Output:**

1. >>>
2. Value of a is 10
3. Value of a is 8
4. Value of a is 6
5. Value of a is 4
6. Value of a is 2
7. Loop is Completed
8. >>>

**Explanation:**

* Firstly, the value in the variable is initialized.
* Secondly, the condition/expression in the while is evaluated. Consequently if condition is true, the control enters in the body and executes all the statements . If the condition/expression passed results in false then the control exists the body and straight away control goes to next instruction after body of while.
* Thirdly, in case condition was true having completed all the statements, the variable is incremented or decremented. Having changed the value of variable step second is followed. This process continues till the expression/condition becomes false.
* Finally Rest of code after body is executed.

**Program to add digits of a number:**

1. n=153
2. sum=0
3. while n>0:
4. r=n%10
5. sum+=r
6. n=n/10
7. print sum

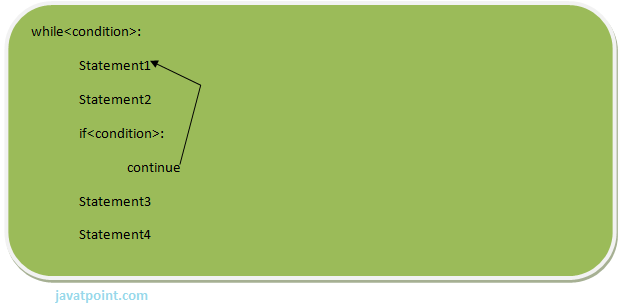
**Output:**

1. >>>
2. 9
3. >>>

**Python Break**

break statement is a jump statement that is used to pass the control to the end of the loop.

When break statement is applied the control points to the line following the body of the loop , hence applying break statement makes the loop to terminate and controls goes to next line pointing after loop body.



**eg:**

1. for i in [1,2,3,4,5]:
2. if i==4:
3. print "Element found"
4. break
5. print i,

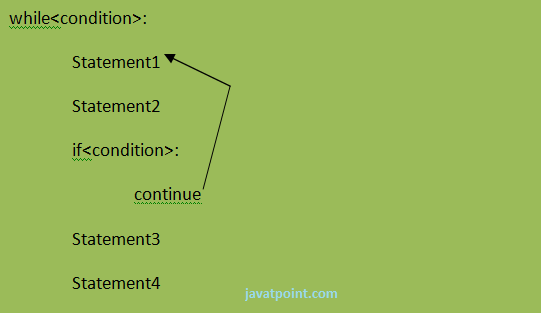
**Output:**

1. >>>
2. 1 2 3 Element found
3. >>>

**Flow chart of break:**

**Continue Statement**

continue Statement is a jump statement that is used to skip the present iteration and forces next iteration of loop to take place. It can be used in while as well as for loop statements.



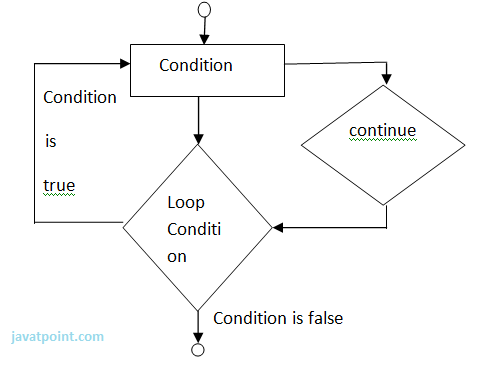
**eg:**

1. a=0
2. while a<=5:
3. a=a+1
4. if a%2==0:
5. continue
6. print a
7. print "End of Loop"

**Output:**

1. >>>
2. 1
3. 3
4. 5
5. End of Loop
6. >>>

**Flow chart of continue:-**



**Python Pass**

When you do not want any code to execute, pass Statement is used. It is same as the name refers to. It just makes the control to pass by without executing any code. If we want to bypass any code pass statement can be used.

**Syntax:**

1. pass

**eg:**

1. for i in [1,2,3,4,5]:
2. if i==3:
3. pass
4. print "Pass when value is",i
5. print i,

**Output:**

1. >>>
2. 1 2 Pass when value is 3
3. 3 4 5
4. >>>

# Python OOPs Concepts

Python is an object-oriented programming language. You can easily create and use classes and objects in Python.

Major principles of object-oriented programming system are given below:

* Object
* Class
* Method
* Inheritance
* Polymorphism
* Data Abstraction
* Encapsulation

## Object

Object is an entity that has state and behavior. It may be anything. It may be physical and logical. For example: mouse, keyboard, chair, table, pen etc.

Everything in Python is an object, and almost everything has attributes and methods. All functions have a built-in attribute \_\_doc\_\_, which returns the doc string defined in the function source code.

## Class

Class can be defined as a collection of objects. It is a logical entity that has some specific attributes and methods. For example: if you have an employee class then it should contain an attribute and method i.e. an email id, name, age, salary etc.

**Syntax:**

1. class ClassName:
2. <statement-1>
3. .
4. .
5. .
6. <statement-N>

## Method

Method is a function that is associated with an object. In Python, method is not unique to class instances. Any object type can have methods.

## Inheritance

Inheritance is a feature of object-oriented programming. It specifies that one object acquires all the properties and behaviors of parent object. By using inheritance you can define a new class with a little or no changes to the existing class. The new class is known as derived class or child class and from which it inherits the properties is called base class or parent class.

It provides re-usability of the code.

## Polymorphism

Polymorphism is made by two words "poly" and "morphs". Poly means many and Morphs means form, shape. It defines that one task can be performed in different ways. For example: You have a class animal and all animals talk. But they talk differently. Here, the "talk" behavior is polymorphic in the sense and totally depends on the animal. So, the abstract "animal" concept does not actually "talk", but specific animals (like dogs and cats) have a concrete implementation of the action "talk".

## Encapsulation

Encapsulation is also the feature of object-oriented programming. It is used to restrict access to methods and variables. In encapsulation, code and data are wrapped together within a single unit from being modified by accident.

## Data Abstraction

Data abstraction and encapsulation both are often used as synonyms. Both are nearly synonym because data abstraction is achieved through encapsulation.

Abstraction is used to hide internal details and show only functionalities. Abstracting something means to give names to things, so that the name captures the core of what a function or a whole program does.

## Object-oriented vs Procedure-oriented Programming languages

|  |  |  |
| --- | --- | --- |
| **Index** | **Object-oriented Programming** | **Procedural Programming** |
| 1. | Object-oriented programming is an approach to problem solving where computation is done by using objects. | Procedural programming uses a list of instructions to do computation step by step. |
| 2. | It makes development and maintenance easier. | In procedural programming, It is not easy to maintain the codes when project becomes lengthy. |
| 3. | It simulates the real world entity. So real world problems can be easily solved through oops. | It doesn't simulate the real world. It works on step by step instructions divided in small parts called functions. |
| 4. | It provides data hiding. so it is more secure than procedural languages. You cannot access private data from anywhere. | Procedural language doesn't provide any proper way for data binding so it is less secure. |
| 5. | Example of object-oriented programming languages are: C++, Java, .Net, Python, C# etc. | Example of procedural languages are: C, Fortran, Pascal, VB etc. |

# Python Object

Python is an object oriented programming language. So its main focus is on objects unlike procedure oriented programming languages which mainly focuses on functions.

In object oriented programming language, object is simply a collection of data (variables) and methods (functions) that act on those data.

## Python Class

A class is a blueprint for the object. Let's understand it by an example:

Suppose a class is a prototype of a building. A building contains all the details about the floor, doors, windows, etc. we can make another buildings (as many as we want) based on these details. So building is a class and we can create many objects from a class.

An object is also called an instance of a class and the process of creating this object is known as instantiation.

Python classes contain all the standard features of Object Oriented Programming. A python class is a mixture of class mechanism of C++ and Modula-3.

## Define a class in Python

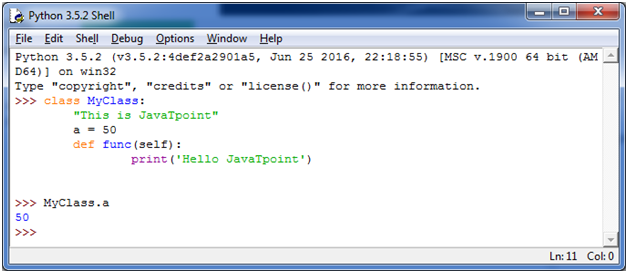
In Python, a class is defined by using a keyword class like a function definition begins with the keyword def.

**Syntax of a class definition:**

1. class ClassName:
2. <statement-1>
3. .
4. .
5. .
6. <statement-N>

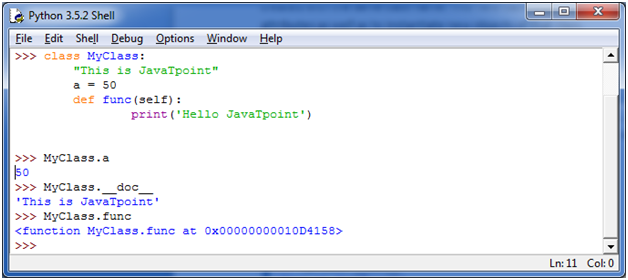
A class creates a new local namespace to define its all attribute. These attributes may be data or functions.

**See this example:**



There are also some special attributes that begins with double underscore (\_\_). For example: \_\_doc\_\_ attribute. It is used to fetch the docstring of that class. When we define a class, a new class object is created with the same class name. This new class object provides a facility to access the different attributes as well as to instantiate new objects of that class.

**See this example:**

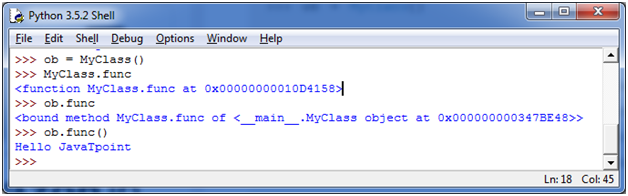


## Create an Object in Python

We can create new object instances of the classes. The procedure to create an object is similar to a function call.

Let's take an example to create a new instance object "ob". We can access attributes of objects by using the object name prefix.

**See this example:**

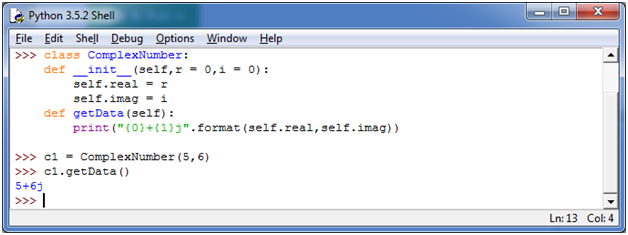


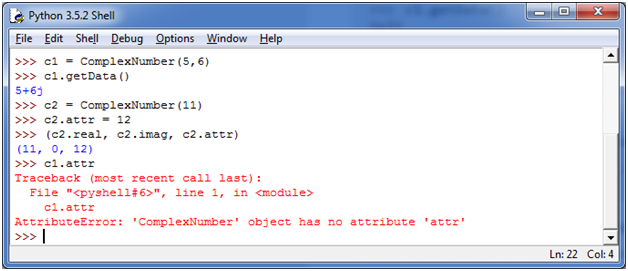
Here, attributes may be data or method. Method of an object is corresponding functions of that class. For example: MyClass.func is a function object and ob.func is a method object.

## Python Object Class Example

1. class Student:
2. def \_\_init\_\_(self, rollno, name):
3. self.rollno = rollno
4. self.name = name
5. def displayStudent(self):
6. print "rollno : ", self.rollno,  ", name: ", self.name
7. emp1 = Student(121, "Ajeet")
8. emp2 = Student(122, "Sonoo")
9. emp1.displayStudent()
10. emp2.displayStudent()

**Output:**

1. rollno :  121 , name:  Ajeet
2. rollno :  122 , name:  Sonoo
3. **Python Constructors**
4. A constructor is a special type of method (function) that is called when it instantiates an object using the definition found in your class. The constructors are normally used to initialize (assign values) to the instance variables. Constructors also verify that there are enough resources for the object to perform any start-up task.
5. **Creating a constructor:**
6. A constructor is a class function that begins with double underscore (\_). The name of the constructor is always the same \_\_init\_\_().
7. While creating an object, a constructor can accept arguments if necessary. When you create a class without a constructor, Python automatically creates a default constructor that doesn't do anything.
8. Every class must have a constructor, even if it simply relies on the default constructor.
9. **Let's take an example:**
10. Let's create a class named ComplexNumber, having two functions \_\_init\_\_() function to initialize the variable and getData() to display the number properly.
11. **See this example:**
12. 
13. You can create a new attribute for an object and read it well at the time of defining the values. But you can't create the attribute for already defined objects.
14. **See this example:**



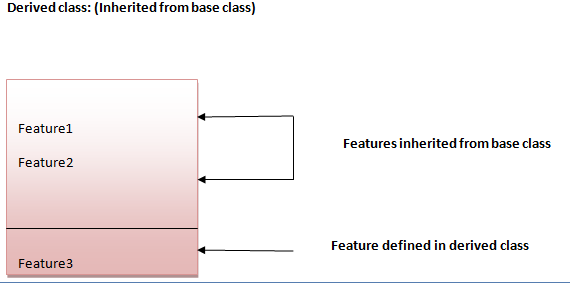
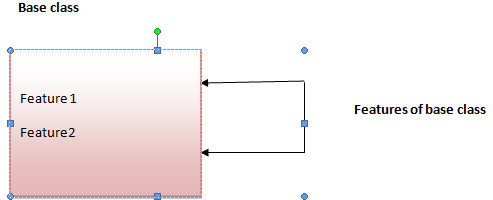
# Inheritance in Python

## What is Inheritance

Inheritance is used to specify that one class will get most or all of its features from its parent class. It is a feature of Object Oriented Programming. It is a very powerful feature which facilitates users to create a new class with a few or more modification to an existing class. The new class is called child class or derived class and the main class from which it inherits the properties is called base class or parent class.

The child class or derived class inherits the features from the parent class, adding new features to it. It facilitates re-usability of code.

**Image representation:**



**Syntax 1:**

1. class DerivedClassName(BaseClassName):
2. <statement-1>
3. .
4. .
5. .
6. <statement-N>

**Syntax 2:**

1. class DerivedClassName(modulename.BaseClassName):
2. <statement-1>
3. .
4. .
5. .
6. <statement-N>

## Parameter explanation:

The name BaseClassName must be defined in a scope containing the derived class definition. You can also use other arbitrary expressions in place of a base class name. This is used when the base class is defined in another module.

## Python Inheritance Example

Let's see a simple python inheritance example where we are using two classes: Animal and Dog. Animal is the parent or base class and Dog is the child class.

Here, we are defining eat() method in Animal class and bark() method in Dog class. In this example, we are creating instance of Dog class and calling eat() and bark() methods by the instance of child class only. Since, parent properties and behaviors are inherited to child object automatically, we can call parent and child class methods by the child instance only.

1. class Animal:
2. def eat(self):
3. print 'Eating...'
4. class Dog(Animal):
5. def bark(self):
6. print 'Barking...'
7. d=Dog()
8. d.eat()
9. d.bark()

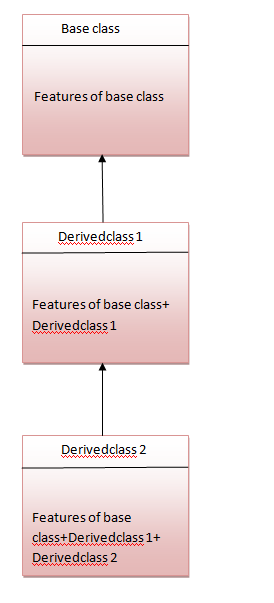
**Output:**

1. Eating...
2. Barking...

# Multilevel Inheritance in Python

Multilevel inheritance is also possible in Python unlike other programming languages. You can inherit a derived class from another derived class. This is known as multilevel inheritance. In Python, multilevel inheritance can be done at any depth.

**Image representation:**



## Python Multilevel Inheritance Example

1. class Animal:
2. def eat(self):
3. print 'Eating...'
4. class Dog(Animal):
5. def bark(self):
6. print 'Barking...'
7. class BabyDog(Dog):
8. def weep(self):
9. print 'Weeping...'
10. d=BabyDog()
11. d.eat()
12. d.bark()
13. d.weep()

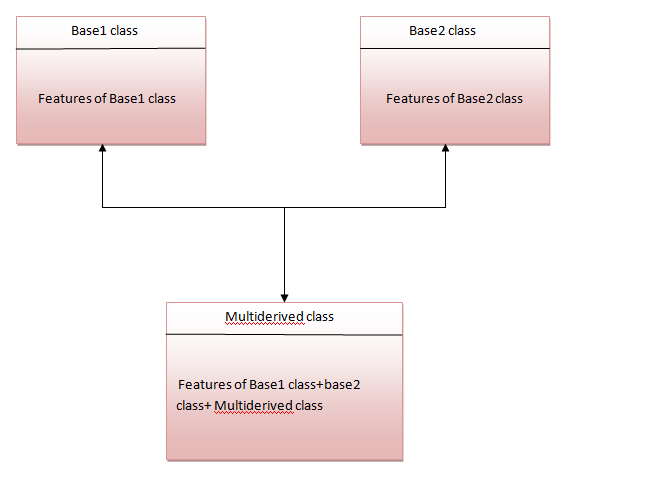
**Output:**

1. Eating...
2. Barking...
3. Weeping

# Multiple Inheritance in Python

Python supports multiple inheritance also. You can derive a child class from more than one base (parent) class.

**Image representation:**



The multiderived class inherits the properties of both class base1 and base2.

Let's see the syntax of multiple inheritance in Python.

**Syntax:**

1. class DerivedClassName(Base1, Base2, Base3):
2. <statement-1>
3. .
4. .
5. .
6. <statement-N>

**Or**

1. class Base1:
2. pass
4. class Base2:
5. pass
7. class MultiDerived(Base1, Base2):
8. pass

**Example:**

1. class First(object):
2. def \_\_init\_\_(self):
3. super(First, self).\_\_init\_\_()
4. print("first")
6. class Second(object):
7. def \_\_init\_\_(self):
8. super(Second, self).\_\_init\_\_()
9. print("second")
11. class Third(Second, First):
12. def \_\_init\_\_(self):
13. super(Third, self).\_\_init\_\_()
14. print("third")
16. Third();

**Output:**

1. first
2. second
3. third

## Why super () keyword

The most commonly super() is used with \_\_init\_\_ function in base classes. This is usually the only place where you need to do some things in a child then complete the initialization in the parent.

**See this example:**

1. class Child(Parent):
2. def \_\_init\_\_(self, stuff):
3. self.stuff = stuff
4. super(Child, self).\_\_init\_\_()

## Composition in Python

Composition is used to do the same thing which can be done by inheritance.