INTRODUCTION

Python is a general purpose, dynamic, high level and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.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, makes it an ideal language for scripting and rapid application development.Python supports *multiple programming pattern*, including object oriented, imperative and functional or procedural programming 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 assign an integer value in an integer 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.

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. In February 1991, van Rossum published the code (labeled version 0.9.0) to alt.sources. In 1994, Python 1.0 was released with new features like: lambda, map, filter, and reduce. Python 2.0 added new features like: list comprehensions, garbage collection system. On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify fundamental flaw of the language. 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 following programming languages:

- ABC language.
- Modula-3

FEATURES

*1) **Easy to Learn and Use**-Python is easy to learn and use. It is developer-friendly and high level programming language.*

*2) **Expressive Language**-Python language is more expressive means that it is more understandable and readable.*

*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 and Macintosh etc. So, we can say that Python is a portable language.*

5) **Free and Open Source-**Python language is freely available at offical web address.The source-code is also available. Therefore it is open source.

6) **Object-Oriented Language-**Python supports object oriented language and concepts of classes and objects come 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 our python code.

8) **Large Standard Library**-Python has a large and broad library and prvides rich set of module and functions for rapid application development.

9) **GUI Programming Support-**Graphical user interfaces can be developed using Python.

10) **Integrated-**It can be easily integrated with languages like C, C++, JAVA etc.

APPLICATIONS AREA

Python is known for its general purpose nature that makes it applicable in almost each domain of software development. Python as a whole can be used in any sphere of development.

1) Web Applications-We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser etc. It also provides Frameworks such as Django, Pyramid, Flask etc to design and delelopweb based applications. Some important developments are: PythonWikiEngines, Pocoo, PythonBlog, Software etc.

2) Desktop GUI Applications- Python provides Tk GUI library to develop user interface in python based application. Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms. The Kivy is popular for writing multitouch applications.

3) Software Development-Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc.

4) Scientific and Numeric-Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.

5) Business Applications-Python is used to build Bussiness applications like ERP and e-commerce systems. Tryton is a high level application platform.

6) Console Based Application-We can use Python to develop console based applications. For example: IPython.

7) Audio or Video based Applications-Python is awesome to perform multiple tasks and can be used to develop multimedia applications. Some of real applications are: TimPlayer, cplay etc.

8) 3D CAD Applications-To create CAD application Fandango is a real application which provides full features of CAD.

9) 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.

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

VARIABLES

Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value.In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type.

Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.It is recommended to use lowercase letters for variable name. Rahul and rahul both are two different variables.

Example:
```
x=50
print x
```

MULTIPLE ASSIGNMENT

Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignment.

Example:
```
x=y=z=50
print x
print y
print z
```
Example:
```
a,b,c=5,10,15
print a
print b
print c
```

SCOPE OF VARIABLE

Scope of a variable can be determined by the part in which variable is defined. Each variable cannot be accessed in each part of a program. There are two types of variables based on Scope:

1) Local Variable.
2) Global Variable.

1) Python Local Variables

Variables declared inside a function body is known as Local Variable. These have a local access thus these variables cannot be accessed outside the function body in which they are declared.

Example:

```
def msg():
        a=10
        print "Value of a is",a
        return

msg()
print a #it will show error since variable is local
```

2) Python Global Variable

Variable defined outside the function is called Global Variable. Global variable is accessed all over program thus global variable have widest accessibility.

Example:

```
b=20
def msg():
        a=10
        print "Value of a is",a
        print "Value of b is",b
        return

        msg()
        print b
```

INPUT AND OUTPUT

Python provides methods that can be used to read and write data. Python also provides supports of reading and writing data to Files.

Python "print" Statement

"print" statement is used to print the output on the screen.print statement is used to take string as input and place that string to standard output. Whatever you want to display on output place that expression inside the inverted commas. The expression whose value is to printed place it without inverted commas.

Syntax:

print "expression" or print expression.

Example:

```
a=10
print "Welcome to the world of Python"
print a
```

INPUT FROM KEYBOARD:

Python offers two built-in functions for taking input from user, given below:

1) input()

2) raw_input()

1) input( ) function--input() function is used to take input from the user. Whatever expression is given by the user, it is evaluated and result is returned back.

Syntax:

input("Expression")

Example:

```
n=input("Enter your expression ");
print "The evaluated expression is ", n
```

2) raw_input( )--raw_input() function is used to take input from the user. It takes the input from the Standard input in the form of a string and reads the data from a line at once.

Syntax:
raw_input(?statement?)

Example:

```
n=raw_input("Enter your name ");
print "Welcome ", n
```

Example:

```
name=raw_input("Enter your name ")
math=float(raw_input("Enter your marks in Math"))
physics=float(raw_input("Enter your marks in Physics"))
chemistry=float(raw_input("Enter your marks in Chemistry"))
rollno=int(raw_input("Enter your Roll no"))
print "Welcome ",name
print "Your Roll no is ",rollno
print "Marks in Maths is ",math
print "Marks in Physics is ",physics
print "Marks in Chemistry is ",chemistry
print "Average marks is ",(math+physics+chemistry)/3
```

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. Python Tuple is used to store the sequence of immutable python objects. Tuple is similar to lists since the value of the items stored in the list can be changed whereas the tuple is immutable and the value of the items stored in the tuple can not be changed.

A tuple can be written as the collection of comma-separated values enclosed with the small brackets. A tuple can be defined as follows.

```
T1 = (101, "Ayush", 22)
T2 = ("Apple", "Banana", "Orange")
```

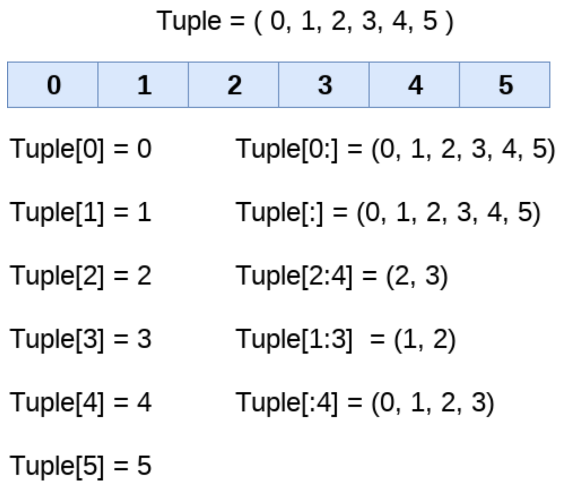
Example:

```
tuple1 = (10, 20, 30, 40, 50, 60)
print(tuple1)
count = 0
for i in tuple1:
  print("tuple1[%d] = %d"%(count, i));
```

Tuple indexing and splitting

The indexing and slicing in tuple are similar to lists. The indexing in the tuple starts from 0 and goes to length(tuple) - 1.The items in the tuple can be accessed by using the slice operator. Python also allows us to use the colon operator to access multiple items in the tuple.

Consider the following image to understand the indexing and slicing in detail.



```
Tuple[0] = 0        Tuple[0:] = (0, 1, 2, 3, 4, 5)
Tuple[1] = 1        Tuple[:] = (0, 1, 2, 3, 4, 5)
Tuple[2] = 2        Tuple[2:4] = (2, 3)
Tuple[3] = 3        Tuple[1:3]  = (1, 2)
Tuple[4] = 4        Tuple[:4] = (0, 1, 2, 3)
Tuple[5] = 5
```

Unlike lists, the tuple items can not be deleted by using the del keyword as tuples are immutable. To delete an entire tuple, we can use the del keyword with the tuple name.

```
tuple1 = (1, 2, 3, 4, 5, 6)
print(tuple1)
del tuple1[0]
print(tuple1)
del tuple1
print(tuple1)
```

Like lists, the tuple elements can be accessed in both the directions. The right most element (last) of the tuple can be accessed by using the index -1. The elements from left to right are traversed using the negative indexing.

```
tuple1 = (1, 2, 3, 4, 5)
print(tuple1[-1])
print(tuple1[-4])
```

<u>Basic Tuple operations</u>

The operators like concatenation (+), repetition (*), Membership (in) works in the same way as they work with the list. Consider the following table for more detail.

Let's say Tuple t = (1, 2, 3, 4, 5) and Tuple t1 = (6, 7, 8, 9) are declared.

Operator	Description	Example

Repetition	The repetition operator enables the tuple elements to be repeated multiple times.	T1*2 = (1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
Concatenation	It concatenates the tuple mentioned on either side of the operator.	T1+T2 = (1, 2, 3, 4, 5, 6, 7, 8, 9)
Membership	It returns true if a particular item exists in the tuple otherwise false.	print (2 in T1) prints True.

Iteration	The for loop is used to iterate over the tuple elements.	for i in T1:
		    print(i)
		Output
		12345

Length	It is used to get the length of the tuple.	len(T1) = 5

<u>Tuple inbuilt functions</u>

SN	Function	Description
1	cmp(tuple1, tuple2)	It compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false.
2	len(tuple)	It calculates the length of the tuple.
3	max(tuple)	It returns the maximum element of the tuple.
4	min(tuple)	It returns the minimum element of the tuple.
5	tuple(seq)	It converts the specified sequence to the tuple.

<u>Where use tuple</u>

Using tuple instead of list is used in the following scenario.

1. Using tuple instead of list gives us a clear idea that tuple data is constant and must not be changed.

2. Tuple can simulate dictionary without keys. Consider the following nested structure which can be used as a dictionary.

[(101, "John", 22), (102, "Mike", 28), (103, "Dustin", 30)]

3. Tuple can be used as the key inside dictionary due to its immutable nature.

LIST VS TUPLE

SN	List	Tuple
1	The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
2	The List is mutable.	The tuple is immutable.
3	The List has the variable length.	The tuple has the fixed length.
4	The list provides more functionality than tuple.	The tuple provides less functionality than the list.
5	The list Is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.	The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items can not be changed. It can be used as the key inside the dictionary.

NESTING LIST AND TUPLE

We can store list inside tuple or tuple inside the list up to any number of level.we store the tuple inside the list.

```
Employees = [(101, "Ayush", 22), (102, "john", 29), (103, "james", 45), (104, "Ben", 34)]
print("----Printing list----");
for i in Employees:
   print(i)
Employees[0] = (110, "David",22)
print();
print("----Printing list after modification----");
```

```
for i in Employees:
   print(i)
```

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([]).Dictionary is used to implement the key-value pair in python. The dictionary is the data type in python which can simulate the real-life data arrangement where some specific value exists for some particular key.

In other words, we can say that a dictionary is the collection of key-value pairs where the value can be any python object whereas the keys are the immutable python object, i.e., Numbers, string or tuple.

Dictionary simulates Java hash-map in python.

Creating the dictionary

The dictionary can be created by using multiple key-value pairs enclosed with the small brackets () and separated by the colon (:). The collections of the key-value pairs are enclosed within the curly braces {}.

The syntax to define the dictionary is given below.

```
Dict = {"Name": "Ayush","Age": 22}
```

In the above dictionary **Dict**, The keys **Name**, and **Age** are the string that is an immutable object.

Example:

```
   Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
   print(type(Employee))
print("printing Employee data .... ")
print(Employee)
```

Accessing the dictionary values

The values can be accessed in the dictionary by using the keys as keys are unique in the dictionary.

The dictionary values can be accessed in the following way.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print("Name : %s" %Employee["Name"])
print("Age : %d" %Employee["Age"])
print("Salary : %d" %Employee["salary"])
print("Company : %s" %Employee["Company"])
```

Python provides us with an alternative to use the get() method to access the dictionary values. It would give the same result as given by the indexing.

Updating dictionary values

The dictionary is a mutable data type, and its values can be updated by using the specific keys.

Example:

```
   Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
   print(type(Employee))
print("printing Employee data .... ")
print(Employee)
print("Enter the details of the new employee....");
   Employee["Name"] = input("Name: ");
   Employee["Age"] = int(input("Age: "));
   Employee["salary"] = int(input("Salary: "));
   Employee["Company"] = input("Company:");
print("printing the new data");
print(Employee)
```

Deleting elements using del keyword

The items of the dictionary can be deleted by using the del keyword as given below.

```
   Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
   print(type(Employee))
print("printing Employee data .... ")
print(Employee)
print("Deleting some of the employee data")
   del Employee["Name"]
   del Employee["Company"]
print("printing the modified information ")
print(Employee)
print("Deleting the dictionary: Employee");
   del Employee
print("Lets try to print it again ");
print(Employee)
```

Iterating Dictionary

A dictionary can be iterated using the for loop as given below.

Example:

```
    Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
    for x in Employee:
        print(x);
```

Example:

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee:
print(Employee[x]);
```

Properties of Dictionary keys

In the dictionary, we can not store multiple values for the same keys. If we pass more than one values for a single key, then the value which is last assigned is considered as the value of the key.

Example.

```
    Employee = {"Name": "John", "Age": 29,
"Salary":25000,"Company":"GOOGLE","Name":"Johnn"}
    for x,y in Employee.items():
        print(x,y)
```

Built-in Dictionary functions

The built-in python dictionary methods along with the description are given below.

SN	Function	Description
1	cmp(dict1, dict2)	It compares the items of both the dictionary and returns true if the first dictionary values are greater than the second dictionary, otherwise it returns false.
2	len(dict)	It is used to calculate the length of the dictionary.
3	str(dict)	It converts the dictionary into the printable string representation.
4	type(variable)	It is used to print the type of the passed variable.

Built-in Dictionary methods

The built-in python dictionary methods along with the description are given below.

SN	Method	Description
1	dic.clear()	It is used to delete all the items of the dictionary.
2	dict.copy()	It returns a shallow copy of the dictionary.
3	dict.fromkeys(iterable, value = None, /)	Create a new dictionary from the iterable with the values equal to value.
4	dict.get(key, default = "None")	It is used to get the value specified for the passed key.
5	dict.has_key(key)	It returns true if the dictionary contains the specified key.
6	dict.items()	It returns all the key-value pairs as a tuple.
7	dict.keys()	It returns all the keys of the dictionary.
8	dict.setdefault(key,default= "None")	It is used to set the key to the default value if the key is not specified in the dictionary
9	dict.update(dict2)	It updates the dictionary by adding the key-value pair of dict2 to this dictionary.
10	dict.values()	It returns all the values of the dictionary.

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.List in python is implemented to store the sequence of various type of data. However, python contains six data types that are capable to store the sequences but the most common and reliable type is list.

A list can be defined as a collection of values or items of different types. The items in the list are separated with the comma (,) and enclosed with the square brackets [].

A list can be defined as follows.

```
L1 = ["John", 102, "USA"]
```

```
    L2 = [1, 2, 3, 4, 5, 6]
    L3 = [1, "Ryan"]
```

If we try to print the type of L1, L2, and L3 then it will come out to be a list.

Example:

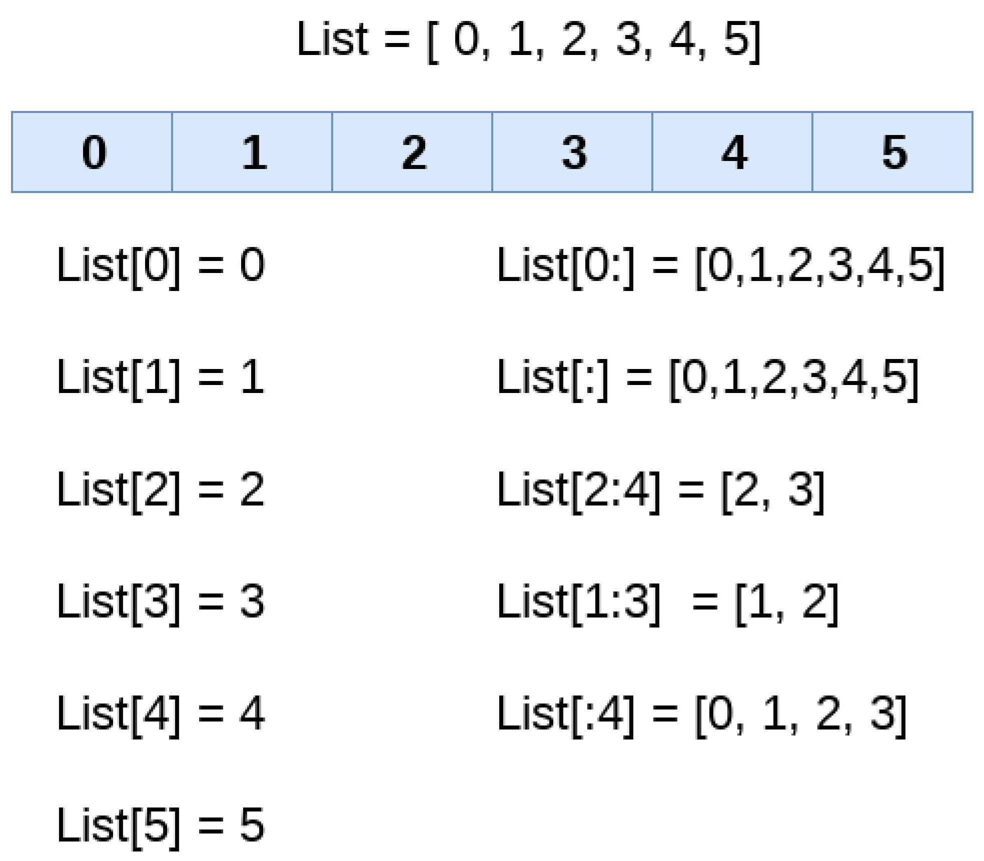
```
mp = ["John", 102, "USA"]
Dep1 = ["CS",10];
Dep2 = ["IT",11];
HOD_CS = [10,"Mr. Holding"]
HOD_IT = [11, "Mr. Bewon"]
print("printing employee data...");
print("Name : %s, ID: %d, Country: %s"%(emp[0],emp[1],emp[2]))
print("printing departments...");
print("Department 1:\nName: %s, ID: %d\nDepartment 2:\nName: %s, ID:
%s"%(Dep1[0],Dep2[1],Dep2[0],Dep2[1]));
print("HOD Details ....");
print("CS HOD Name: %s, Id: %d"%(HOD_CS[1],HOD_CS[0]));
print("IT HOD Name: %s, Id: %d"%(HOD_IT[1],HOD_IT[0]));
print(type(emp),type(Dep1),type(Dep2),type(HOD_CS),type(HOD_IT));
```

List indexing and splitting

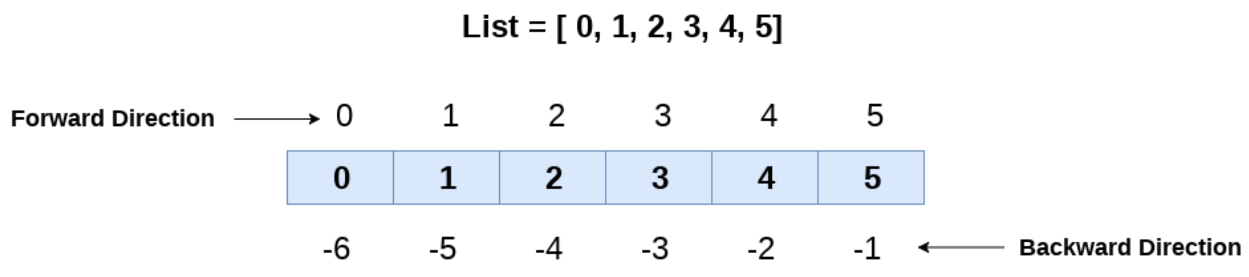
The indexing are processed in the same way as it happens with the strings. The elements of the list can be accessed by using the slice operator [].

The index starts from 0 and goes to length - 1. The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.

Consider the following example.



Unlike other languages, python provides us the flexibility to use the negative indexing also. The negative indices are counted from the right. The last element (right most) of the list has the index -1, its adjacent left element is present at the index -2 and so on until the left most element is encountered.



Updating List values

Lists are the most versatile data structures in python since they are immutable and their values can be updated by using the slice and assignment operator.

Python also provide us the append() method which can be used to add values to the string.

Consider the following example to update the values inside the list.

```
List = [1, 2, 3, 4, 5, 6]
```

```
print(List)
List[2] = 10;
print(List)
List[1:3] = [89, 78]
print(List)
```

The list elements can also be deleted by using the **del** keyword. Python also provides us the remove() method if we do not know which element is to be deleted from the list.

Consider the following example to delete the list elements.

```
    List = [0,1,2,3,4]
print(List)
    del List[0]
print(List)
    del List[3]
print(List)
```

Python List Operations

The concatenation (+) and repetition (*) operator work in the same way as they were working with the strings.

Operator	Description	Example
Repetition	The repetition operator enables the list elements to be repeated multiple times.	`L1*2 = [1, 2, 3, 4, 1, 2, 3, 4]`
Concatenation	It concatenates the list mentioned on either side of the operator.	`l1+l2 = [1, 2, 3, 4, 5, 6, 7, 8]`
Membership	It returns true if a particular item exists in a particular list otherwise false.	`print(2 in l1) prints True.`

Iteration	The for loop is used to iterate over the list elements.	`for i in l1:`
		`    print(i)`
		Output
		1
		2
		3
		4

Length	It is used to get the length of the list	`len(l1) = 4`

Iterating a List

A list can be iterated by using a for - in loop. A simple list containing four strings can be iterated as follows.

```
    List = ["John", "David", "James", "Jonathan"]
    for i in List: #i will iterate over the elements of the List and contains each element in each
iteration.
        print(i);
```

Adding elements to the list

Python provides append() function by using which we can add an element to the list. However, the append() method can only add the value to the end of the list.

Consider the following example in which, we are taking the elements of the list from the user and printing the list on the console.

```
    l =[];
    n = int(input("Enter the number of elements in the list")); #Number of elements will be
entered by the user
    for i in range(0,n): # for loop to take the input
l.append(input("Enter the item?")); # The input is taken from the user and added to the list as the
item
print("printing the list items....");
    for i in l: # traversal loop to print the list items
print(i, end = "  ");
```

Python List Built-in functions

Python provides the following built-in functions which can be used with the lists.

SN	Function	Description
1	cmp(list1, list2)	It compares the elements of both the lists.
2	len(list)	It is used to calculate the length of the list.
3	max(list)	It returns the maximum element of the list.
4	min(list)	It returns the minimum element of the list.
5	list(seq)	It converts any sequence to the list.

Python List built-in methods

SN	Function	Description
1	list.append(obj)	The element represented by the object obj is added to the list.

2	list.clear()	It removes all the elements from the list.
3	List.copy()	It returns a shallow copy of the list.
4	list.count(obj)	It returns the number of occurrences of the specified object in the list.
5	list.extend(seq)	The sequence represented by the object seq is extended to the list.
6	list.index(obj)	It returns the lowest index in the list that object appears.
7	list.insert(index, obj)	The object is inserted into the list at the specified index.
8	list.pop(obj=list[-1])	It removes and returns the last object of the list.
9	list.remove(obj)	It removes the specified object from the list.
10	list.reverse()	It reverses the list.
11	list.sort([func])	It sorts the list by using the specified compare function if given.

KEYWORDS

Python Keywords are special reserved words which convey a special meaning to the compiler/interpreter. Each keyword have a special meaning and a specific operation. These keywords can't be used as variable. Following is the List of Python Keywords.

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

1. Identifiers are the names given to the fundamental building blocks in a program.
2. These can be variables ,class ,object ,functions , lists , dictionaries etc.
3. There are certain rules defined for naming i.e., Identifiers.
   I. An identifier is a long sequence of characters and numbers.

II. II.No special character except underscore (_) can be used as an identifier.
III. III.Keyword should not be used as an identifier name.
IV. IV.Python is case sensitive. So using case is significant.
V. V.First character of an identifier can be character, underscore (_) but not digit.

OPERATORS

Operators are particular symbols that are used to perform operations on operands. It returns result that can be used in application.

TYPES OF OPERATORS

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

The following table contains the arithmetic operators that are used to perform arithmetic operations.

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)

Example:
```
2**3    => 8
 10//3    => 3
```

Relational Operators

The following table contains the relational operators that are used to check relations.

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 !=)

Assignment Operators

The following table contains the assignment operators that are used to assign values to the variables.

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

Logical Operators

The following table contains the arithmetic operators that are used to perform arithmetic operations.

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)

Example:

```
a=5>4 and 3>2
print a                 => True
b=5>4 or 3<2
print b           => True
c=not(5>4)
print c                  => False
```

Membership Operators

The following table contains the 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.

Example:

```
a=10
b=20
list=[10,20,30,40,50];
if (a in list):
    print "a is in given list"
else:
    print "a is not in given list"
if(b not in list):
    print "b is not given in list"
else:
        print "b is given in list"
```

Identity Operators

The following table contains the 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:

```
a=20
b=20
if( a is b):
   print  a,b have same identity
else:
    print a, b are different
   b=10
if( a is not b):
     print  a,b have different identity
else:
     print a,b have same identity
```

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: # This is single line comment.

2) Multi lined Comment:

Multi lined comment can be given inside triple quotes.

```
eg: """ This
       Is
       Multipline comment"'
```

IF ELSE

The If statement is used to test specified condition and if the condition is true, if block executes, otherwise else block executes.

The else statement executes when the if statement is false.

Syntax:

```
if(condition):
     statements
else:
     statements
```

Example:

```
year=2000
if year%4==0:
  print  "Year is Leap"
else:
   print "Year is not Leap"
```

NESTED IF ELSE

In python, we can use nested If Else to check multiple conditions. Python provides elif keyword to make nested If statement. This statement is like executing a if statement inside a else statement.

Syntax:

```
If statement:
      Body
elif statement:
      Body
 else:
      Body
```

Example:

```
a=10
if a>=20:
   print "Condition is True"
else:
   if a>=15:
      print "Checking second value"
   else:
      print "All Conditions are false"
```

FOR LOOP

Python for loop is used to iterate the elements of a collection in the order that they appear. This collection can be a sequence (list or string). 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.

Syntax:
```
for <variable> in <sequence>:
    statement
```

Example:cars=[1,2,3]
```
for x in cars:
                print x
```

RANGE( )

Example:
```
for n in range(5):     #STARTS WITH 0 AND ENDS BEFORE FIVE AND 1 INCREMEN
   print n
print sum
```

Example:
```
sum=0
for n in range(1,11):     #STARTS WITH 1 AND ENDS BEFORE ELEVEN AND 1 INCREME
   sum+=n
print sum
```

Example:
```
sum=0
for n in range(1,11,2):     #STARTS WITH 1 AND ENDS BEFORE 11 AND 3 INCREME
   sum+=n
print sum
```

NESTED FOR LOOPS

Loops defined within another Loop are called Nested Loops. Nested loops are used to iterate matrix elements or to perform complex computation.When an outer loop contains an inner loop in its body it is called Nested Looping.

Example:
```
for i in range(1,6):
   for j in range (1,i+1):
      print i,
    print
```

WHILE LOOP

In Python, while loop is used to execute number of statements or body till the specified condition is true. Once the condition is false, the control will come out of the loop.
Syntax:

```
while <expression>:

  Body
```

Example:

```
a=10
while a>0:
    print "Value of a is",a
    a=a-2
```

BREAK

Break statement is a jump statement which is used to transfer execution control. It breaks the current execution and in case of inner loop, inner loop terminates immediately.

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.

Example:

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

CONTINUE

Python Continue Statement is a jump statement which is used to skip execution of current iteration. After skipping, loop continue with next iteration.

Example:

```
a=0
while a<=5:
    a=a+1
    if a%2==0:
        continue
    print a
print "End of Loop"
```

PASS

In Python, pass keyword is used to execute nothing; it means, when we don't want to execute code, the pass can be used to execute empty. 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.

Example:

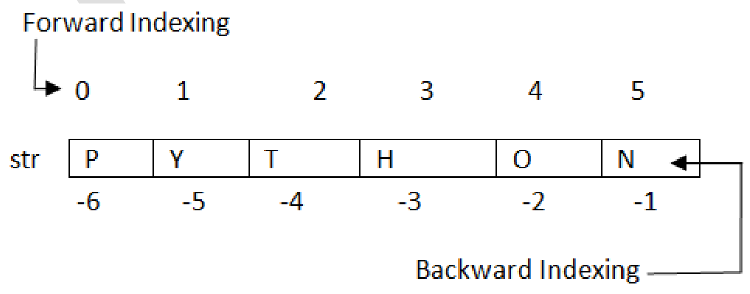
```
for i in [1,2,3,4,5]:
        if i==3:
          pass
         print "Pass when value is",i
        print i,
```

STRINGS

Python string is a built-in type text sequence. It is used to handle textual data in python. Python Strings are immutable sequences of Unicode points. Creating Strings are simplest and easy to use in Python. We can simply create Python String by enclosing a text in single as well as double quotes. Python treat both single and double quotes statements same.

Accessing Python Strings

- In Python, Strings are stored as individual characters in a contiguous memory location.
- The benefit of using String is that it can be accessed from both the directions (forward and backward).
- Both forward as well as backward indexing are provided using Strings in Python.
    - Forward indexing starts with 0,1,2,3,....
    - Backward indexing starts with -1,-2,-3,-4,....



Example:

```
name="Rajat"
length=len(name)
i=0
```

```
for n in range(-1,(-length-1),-1):
    print name[i],"\t",name[n]
    i+=1
```

Output-

```
R        t
a        a
j        j
a        a
t        R
```

STRINGS OPERATORS

To perform operation on string, Python provides basically 3 types of Operators that are given below.

1. Basic Operators.
2. Membership Operators.
3. Relational Operators.

String Basic Operators

There are two types of basic operators in String "+" and "*".

String Concatenation Operator (+)

The concatenation operator (+) concatenates two Strings and creates a new String.

Expression	**Output**
'10' + '20'	'1020'
"s" + "007"	's007'
'abcd123' + 'xyz4'	'abcd123xyz4'

String Replication Operator (*)

Replication operator uses two parameters for operation, One is the integer value and the other one is the String argument.The Replication operator is used to repeat a string number of times. The string will be repeated the number of times which is given by the integer value.

Expression	**Output**
"soono"*2	'soonosoono'
3*'1'	'111'
'$'*5	'$$$$$'

STRING FUNCTIONS AND METHODS

Python provides various predefined or built-in string functions. They are as follows:

capitalize()	It capitalizes the first character of the String.
count(string,begin,end)	It Counts number of times substring occurs in a String between begin and end index.
endswith(suffix ,begin=0,end=n)	It returns a Boolean value if the string terminates with given suffix between begin and end.
find(substring ,beginIndex, endIndex)	It returns the index value of the string where substring is found between begin index and end index.
index(subsring, beginIndex, endIndex)	It throws an exception if string is not found and works same as find() method.
isalnum()	It returns True if characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise it returns False.
isalpha()	It returns True when all the characters are alphabets and there is at least one character, otherwise False.
isdigit()	It returns True if all the characters are digit and there is at least one character, otherwise False.
islower()	It returns True if the characters of a string are in lower case, otherwise False.
isupper()	It returns False if characters of a string are in Upper case, otherwise False.
isspace()	It returns True if the characters of a string are whitespace, otherwise false.
len(string)	It returns the length of a string.

lower()	It converts all the characters of a string to Lower case.
upper()	It converts all the characters of a string to Upper Case.

FUNCTIONS

A Function is a self block of code which is used to organize the functional code. Function can be called as a section of a program that is written once and can be executed whenever required in the program, thus making code reusability. Function is a subprogram that works on data and produces some output.

Types of Functions:

There are two types of Functions.

a) Built-in Functions: Functions that are predefined and organized into a library. We have used many predefined functions in Python.

b) User- Defined: Functions that are created by the programmer to meet the requirements.

DEFINING A FUNCTION

A Function defined in Python should follow the following format:

1) Keyword def is used to start and declare a function. Def specifies the starting of function block.
2) def is followed by function-name followed by parenthesis.
3)Parameters are passed inside the parenthesis. At the end a colon is marked.
4) Python code requires indentation (space) of code to keep it associate to the declared block.
5) The first statement of the function is optional. It is ?Documentation string? of function.
6) Following is the statement to be executed.

Syntax:
```
def <function_name>(parameters):
        statement
```
Example:
```
def sum( ):
"Sample"
 print "Welcome"

  #Calling the Function
 sum()
```

FUNCTION ARGUMENT AND PARAMETER

There can be two types of data passed in the function.
1) The First type of data is the data passed in the function call. This data is called arguments.
2) The second type of data is the data received in the function definition. This data is called parameters.
Arguments can be literals, variables and expressions. Parameters must be variable to hold incoming values.
Alternatively, arguments can be called as actual parameters or actual arguments and parameters can be called as formal parameters or formal arguments.

Example:

```
def sum(x,y):
    "Going to add x and y"
     s=x+y
    print "Sum of two numbers is"
      print s
      #Calling the sum Function
     sum(10,20)
     sum(20,30)
```

FUNCTION RETURN STATEMENT

return[expression] is used to return response to the caller function. We can use expression with the return keyword. send back the control to the caller with the expression. In case no expression is given after return it will return None. In other words return statement is used to exit the function definition.

```
def sum(a,b):
        "Adding the two values"
      print "Printing within Function"
 print a+b
        return a+b
def msg():
        print "Hello"
        return

total=sum(10,20)
print “Printing Outside:”, total
msg()
print "Rest of code"
```

FUNCTION DEFAULT ARGUMENTS

Default Argument is the argument which provides the default values to the parameters passed in the function definition, in case value is not provided in the function call default value is used.

```
def msg(Id,Name,Age=21):
      "Printing the passed value"
    print Id
    print Name
    print Age
    return
#Function call
msg(Id=100,Name='Ravi',Age=20)
msg(Id=101,Name='Ratan')
```

Explanation:.

1) In first case, when msg() function is called passing three different values i.e., 100 , Ravi and 20, these values will be assigned to respective parameters and thus respective values will be printed.

2) In second case, when msg() function is called passing two values i.e., 101 and Ratan, these values will be assigned to Id and Name respectively. No value is assigned for third argument via function call and hence it will retain its default value i.e, 21.

KEYWORD ARGUMENTS

Using the Keyword Argument, the argument passed in function call is matched with function definition on the basis of the name of the parameter.

```
def msg(id,name):
    "Printing passed value"
       print id
       print name
     return
msg(id=100,name='Raj')
msg(name='Rahul',id=101)
```

Explanation:
1) In the first case, when msg() function is called passing two values i.e., id and name the position of parameter passed is same as that of function definition and hence values are initialized to respective parameters in function definition. This is done on the basis of the name of the parameter.

2) In second case, when msg() function is called passing two values i.e., name and id, although the position of two parameters is different it initialize the value of id in Function call to id in Function Definition. same with name parameter. Hence, values are initialized on the basis of name of the parameter.

ANONYMOUS FUNCTION

Anonymous Functions are the functions that are not bond to name. It means anonymous function does not has a name. Anonymous Functions are created by using a keyword "lambda". Lambda takes any number of arguments and returns an evaluated expression. Lambda is created without using the def keyword.

Syntax:

```
lambda arg1,args2,args3,?,argsn :expression
```

Example:

```
square=lambda x1: x1*x1

#Calling square as a function
print "Square of number is",square(10)
```

EXCEPTION HANDLING

Exception can be said to be any abnormal condition in a program resulting to the disruption in the flow of the program.Whenever an exception occurs the program halts the execution and thus further code is not executed. Thus exception is that error which python script is unable to tackle with.

Exception in a code can also be handled. In case it is not handled, then the code is not executed further and hence execution stops when exception occurs.

Common Exceptions

1. ZeroDivisionError: Occurs when a number is divided by zero.
2. NameError: It occurs when a name is not found. It may be local or global.
3. IndentationError: If incorrect indentation is given.
4. IOError: It occurs when Input Output operation fails.
5. EOFError: It occurs when end of the file is reached and yet operations are being performed.

EXCEPTION HANDLING:

The suspicious code can be handled by using the try block. Enclose the code which raises an exception inside the try block. The try block is followed except statement. It is then further followed by statements which are executed during exception and in case if exception does not occur.

Syntax:

```
try:
    malicious code
except Exception1:
    execute code
except Exception2:
    execute code
....
....
except ExceptionN:
    execute code
else:
    In case of no exception, execute the else block code.
```

Example:

```
try:
    a=10/0
    print a
except ArithmeticError:
        print "This statement is raising an exception"
else:
    print "Welcome"
```

The malicious code (code having exception) is enclosed in the try block. Try block is followed by except statement. There can be multiple except statement with a single try block. Except statement specifies the exception which occurred. In case that exception is occurred, the corresponding statement will be executed. At the last you can provide else statement. It is executed when no exception is occurred.

EXCEPTION(EXCEPT WITH NO EXCEPTION)

```
try:
    a=10/0;
except:
    print "Arithmetic Exception"
else:
    print "Successfully Done"
```

DECLARING MULTIPLE EXCEPTION

```
try:
    a=10/0;
except ArithmeticError,StandardError:
    print "Arithmetic Exception"
else:
    print "Successfully Done"
```

FINALLY BLOCK

In case if there is any code which the user want to be executed, whether exception occurs or not then that code can be placed inside the finally block. Finally block will always be executed irrespective of the exception.

Syntax:
```
try:
   Code
finally:
   code which is must to be executed.
```

Example:
```
try:
   a=10/0;
   print "Exception occurred"
finally:
   print "Code to be executed"
```

RAISE AN EXCEPTION

You can explicitly throw an exception in Python using ?raise? statement. raise will cause an exception to occur and thus execution control will stop in case it is not handled.

Syntax:

```
raise Exception_class,<value>
```

Example:
```
try:
   a=10
   print a
   raise NameError("Hello")
except NameError as e:
      print "An exception occurred"
      print e
```

To raise an exception, raise statement is used. It is followed by exception class name. Exception can be provided with a value that can be given in the parenthesis. (here, Hello). To access the value "as" keyword is used. "e" is used as a reference variable which stores the value of the exception.

CUSTOM EXCEPTION

Creating your own Exception class or User Defined Exceptions are known as Custom Exception.

Example:
```
class ErrorInCode(Exception):
    def __init__(self, data):
  self.data = data
    def __str__(self):
      return repr(self.data)
```

```
try:
    raise ErrorInCode(2000)
except ErrorInCode as ae:
    print "Received error:", ae.data
```

MODULE

Modules are used to categorize Pyhton code into smaller parts. A module is simply a Python file, where classes, functions and variables are defined. Grouping similar code into a single file makes it easy to access. Ex- If the content of a book is not indexed or categorized into individual chapters, the book might have turned boring and hectic. Hence, dividing book into chapters made it easy to understand.

In the same sense python modules are the files which have similar code. Thus module is simplify a python code where classes, variables and functions are defined.

Python Module Advantage

Python provides the following advantages for using module:

1) **Reusability:** Module can be used in some other python code. Hence it provides the facility of code reusability.

2) **Categorization:** Similar type of attributes can be placed in one module.

IMPORTING A MODULE

There are different ways by which you we can import a module. These are as follows:

1) USING IMPORT STATEMENT:

"import" statement can be used to import a module.

Syntax:

```
import <file_name1, file_name2,...file_name(n)="">
```

Example:

```
def add(a,b):
    c=a+b
    print c
    return
```

Save the file by the name addition.py. To import this file "import" statement is used. Create another python file in which you want to import the former python file. For that, import statement is used as given in the above example. The corresponding method can be used by file_name.method (). (Here, addition. add (), where addition is the python file and add () is the method defined in the file addition.py)

```
import addition
```

```
addition.add(10,20)
addition.add(30,40)
```

IMPORTING MULTIPLE MODULES

1) msg.py:

```
def msg_method():
   print "Today the weather is rainy"
   return
```

2) display.py:

```
def display_method():
   print "The weather is Sunny"
   return
```

3) multiimport.py:

```
import msg,display
msg.msg_method()
display.display_method()
```

2) USING FROM.. IMPORT STATEMENT:

from..import statement is used to import particular attribute from a module. In case you do not want whole of the module to be imported then you can use from ?import statement.

Syntax:

```
from  <module_name> import <attribute1,attribute2,attribute3,...attributen>
</attribute1,attribute2,attribute3,...attributen></module_name>
```

Example:

```
def circle(r):
   print 3.14*r*r
   return

def square(l):
   print l*l
   return

def rectangle(l,b):
   print l*b
   return
```

```
def triangle(b,h):
    print 0.5*b*h
    return
```

create another file area.py

```
from area import square,rectangle
square(10)
rectangle(2,5)
```

3) TO IMPORT WHOLE MODULE:

You can import whole of the module using "from? import *"

Syntax:

```
      from <module_name> import *
</module_name>
```

area1.py

```
from area import *
square(10)
rectangle(2,5)
circle(5)
triangle(10,20)
```

BUILT IN MODULES IN PYTHON

There are many built in modules in Python. Some of them are as follows:math, random , threading , collections , os , mailbox , string , time , tkinter etc..Each module has a number of built in functions which can be used to perform various functions.

1) math:

Using math module , you can use different built in mathematical functions.

Functions:

Function	**Description**
ceil(n)	It returns the next integer number of the given number

sqrt(n)	It returns the Square root of the given number.
exp(n)	It returns the natural logarithm e raised to the given number
floor(n)	It returns the previous integer number of the given number.

Example:

```
import math
a=4.6
print math.ceil(a)
print math.floor(a)
b=9
print math.sqrt(b)
print math.exp(3.0)
print math.log(2.0)
print math.pow(2.0,3.0)
print math.sin(0)
print math.cos(0)
print math.tan(45)
```

2) random:

The random module is used to generate the random numbers. It provides the following two built in functions:

Function	**Description**
random()	It returns a random number between 0.0 and 1.0 where 1.0 is exclusive.
randint(x,y)	It returns a random number between x and y where both the numbers are inclusive.

Example:

```
import random

print random.random()
print random.randint(2,8)
```

PACKAGE

A Package is simply a collection of similar modules, sub-packages etc..

Steps to create and import Package:

1) Create a directory, say Info
2) Place different modules inside the directory. We are placing 3 modules msg1.py, msg2.py and msg3.py respectively and place corresponding codes in respective modules. Let us place msg1() in msg1.py, msg2() in msg2.py and msg3() in msg3.py.
3) Create a file __init__.py which specifies attributes in each module.
__init__.py is simply a file that is used to consider the directories on the disk as the package of the Python. It is basically used to initialize the python packages.
4) Import the package and use the attributes using package.

Example:
1) Create the directory:

```
import os
os.mkdir("Info")
```

2) Place different modules in package: (Save different modules inside the Info package) msg1.py

```
def msg1():
   print "This is msg1"
```

msg2.py

```
def msg2():
   print "This is msg2"
```

msg3.py

```
def msg3():
   print "This is msg3"
```

3) Create __init__.py file:

```
from msg1 import msg1
from msg2 import msg2
from msg3 import msg3
```

4)Import package and use the attributes:

```
import Info
Info.msg1()
Info.msg2()
Info.msg3()
```

OOPS CONCEPTS

Python is an object-oriented programming language. It allows us to develop applications using Object Oriented approach. In Python, we can easily create and use classes and objects.

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

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. 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.

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

Syntax:

```
class ClassName:
   <statement-1>
   .
   .
   <statement-N>
```

Example:

```
class Student:
      def displayStudent():
   print ("hello")
emp1 = Student()
emp1.displayStudent()
```

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 problem solving approach and used 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.

CONSTRUCTORS

A constructor is a special type of method (function) which is used to initialize the instance members of the class. Constructor can be parameterized and non-parameterized as well. Constructor definition executes when we create object of the class. Constructors also verify that there are enough resources for the object to perform any start-up task.

Creating a Constructor

A constructor is a class function that begins with double underscore (_). The name of the constructor is always the same __init__().While creating an object, a constructor can accept arguments if necessary. When we create a class without a constructor, Python automatically creates a default constructor that doesn't do anything.

Every class must have a constructor, even if it simply relies on the default constructor.

Example

```
class Student:
   # Constructor - non parameterized
def __init__(self):
print("This is non parametrized constructor")
def show(self,name):
print("Hello",name)
student = Student()
student.show("amit")
```

Parameterized Constructor

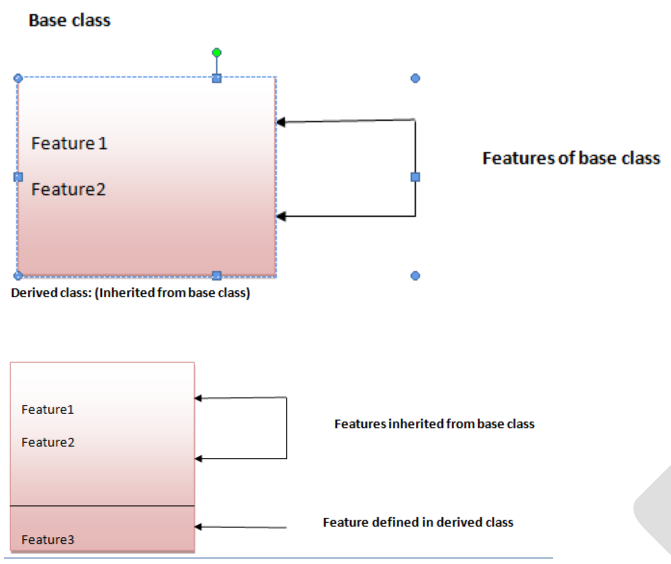
The parameterized constructors are used to set custom value for instance variables that can be used further in the application.

```
class Student:
   # Constructor - parameterized
def __init__(self, name):
print("This is parametrized constructor")
      self.name = name
def show(self):
print("Hello",self.name)
student = Student("irfan")
student.show()
```

INHERITANCE

Inheritance is a feature of Object Oriented Programming. It is used to specify that one class will get most or all of its features from its parent class. 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:


Base class
Feature 1
Feature2
Features of base class
Derived class: (Inherited from base class)
Feature1
Feature2
Features inherited from base class
Feature defined in derived class
Feature3

Syntax

```
 class DerivedClassName(BaseClassName):
    <statement-1>
    .
        <statement-N>
```

Or

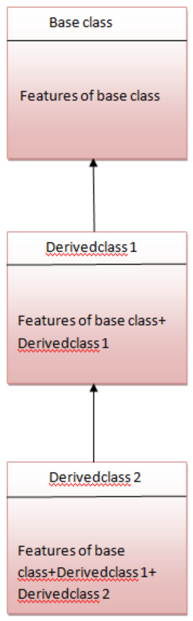
```
class DerivedClassName(modulename.BaseClassName):
    <statement-1>
    .
    .
   <statement-N>
```

Example:

```
class Animal:
    def eat(self):
      print 'Eating...'
class Dog(Animal):
   def bark(self):
      print 'Barking...'
d=Dog()
d.eat()
d.bark()
```

MULTILEVEL INHERITANCE

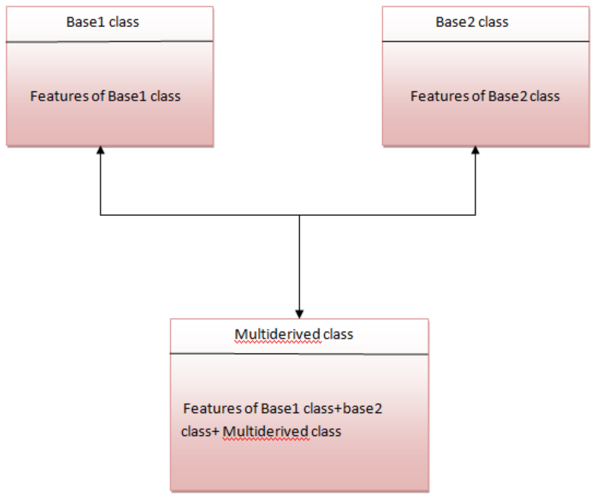
Multilevel inheritance is also possible in Python like other Object Oriented programming languages. We can inherit a derived class from another derived class, this process is known as multilevel inheritance. In Python, multilevel inheritance can be done at any depth.


Base class
Features of base class
Derivedclass 1
Features of base class+ Derivedclass 1
Derivedclass 2
Features of base class+Derivedclass 1+ Derivedclass 2

```
class Animal:
    def eat(self):
      print 'Eating...'
class Dog(Animal):
   def bark(self):
      print 'Barking...'
class BabyDog(Dog):
    def weep(self):
        print 'Weeping...'
d=BabyDog()
d.eat()
d.bark()
d.weep()
```

MULTIPLE INHERITANCE

Python supports multiple inheritance too. It allows us to inherit multiple parent classes. We can derive a child class from more than one base (parent) classes.



```
class First(object):
  def __init__(self):
    super(First, self).__init__()
    print("first")

class Second(object):
  def __init__(self):
    super(Second, self).__init__()
    print("second")

class Third(Second, First):
  def __init__(self):
    super(Third, self).__init__()
    print("third")

Third();
```

SUPER () KEYWORD

The super() method is most commonly used with __init__ function in base class. This is usually the only place where we need to do some things in a child then complete the initialization in the parent.

Example:

```
class Child(Parent):
    def __init__(self, stuff):
        self.stuff = stuff
        super(Child, self).__init__()
```

FILE HANDLING

Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again.

However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time. Here, comes the need of file handling.

Opening a file

Python provides the open( ) function which accepts two arguments, file name and access mode in which the file is accessed. The function returns a file object which can be used to perform various operations like reading, writing, etc.

The syntax to use the open( ) function is given below.

file object = open(<file-name>, <access-mode>, <buffering>)

The files can be accessed using various modes like read, write, or append. The following are the details about the access mode to open a file.

SN	Access mode	Description
1	r	It opens the file to read-only. The file pointer exists at the beginning. The file is by default open in this mode if no access mode is passed.
2	rb	It opens the file to read only in binary format. The file pointer exists at the beginning of the file.
3	r+	It opens the file to read and write both. The file pointer exists at the beginning of the file.
4	rb+	It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file.
5	w	It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
6	wb	It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
7	w+	It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn?t overwrite the previously written file. It creates a new file if no file exists. The file pointer exists

		at the beginning of the file.
8	wb+	It opens the file to write and read both in binary format. The file pointer exists at the beginning of the file.
9	a	It opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any. It creates a new file if no file exists with the same name.
10	ab	It opens the file in the append mode in binary format. The pointer exists at the end of the previously written file. It creates a new file in binary format if no file exists with the same name.
11	a+	It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name.
12	ab+	It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

Example:

```
fileptr = open("file.txt","r")
if fileptr:
   print("file is opened successfully")
```

close() method

Once all the operations are done on the file, we must close it through our python script using the close() method. Any unwritten information gets destroyed once the close() method is called on a file object. We can perform any operation on the file externally in the file system is the file is opened in python, hence it is good practice to close the file once all the operations are done.

Example:

```
fileptr = open("file.txt","r")
  if fileptr:
    print("file is opened successfully")
fileptr.close()
```

Reading the file

To read a file using the python script, the python provides us the read( ) method. The read( ) method reads a string from the file. It can read the data in the text as well as binary format.

The syntax of the read() method is given below.

fileobj.read(<count>)

Here, the count is the number of bytes to be read from the file starting from the beginning of the file. If the count is not specified, then it may read the content of the file until the end.

Example:

```
fileptr = open("file.txt","r");
content = fileptr.read(9);
print(type(content))
print(content)
fileptr.close()
```

Read Lines of the file

Python facilitates us to read the file line by line by using a function readline( ). The readline( ) method reads the lines of the file from the beginning, i.e., if we use the readline() method two times, then we can get the first two lines of the file.

Consider the following example which contains a function readline() that reads the first line of our file **"file.txt"** containing three lines.

Example:

```
fileptr = open("file.txt","r");
content = fileptr.readline();
print(type(content))
print(content)
fileptr.close()
```

Looping through the file

By looping through the lines of the file, we can read the whole file.

Example:

```
fileptr = open("file.txt","r");
for i in fileptr:
print(i)
```

Writing the file

To write some text to a file, we need to open the file using the open method with one of the following access modes.

a: It will append the existing file. The file pointer is at the end of the file. It creates a new file if no file exists.

w: It will overwrite the file if any file exists. The file pointer is at the beginning of the file.

Example.

```
fileptr = open("file.txt","a");
```

```
fileptr.write("Python is the modern day language. It makes things so simple.")
fileptr.close();
```

Creating a new file

The new file can be created by using one of the following access modes with the function open().
x: it creates a new file with the specified name. It causes an error a file exists with the same name.
a: It creates a new file with the specified name if no such file exists. It appends the content to the file if the file already exists with the specified name.
w: It creates a new file with the specified name if no such file exists. It overwrites the existing file.
Example:

```
fileptr = open("file2.txt","x");
print(fileptr)
if fileptr:
   print("File created successfully");
```

Using with statement with files

The with statement was introduced in python 2.5. The with statement is useful in the case of manipulating the files. The with statement is used in the scenario where a pair of statements is to be executed with a block of code in between.

The syntax to open a file using with statement is given below.

```
            with open(<file name>, <access mode>) as <file-pointer>:
 #statement suite
```

The advantage of using with statement is that it provides the guarantee to close the file regardless of how the nested block exits.

It is always suggestible to use the with statement in the case of file s because, if the break, return, or exception occurs in the nested block of code then it automatically closes the file. It doesn't let the file to be corrupted.

Example:

```
with open("file.txt",'r') as f:
   content = f.read();
   print(content)
```

FILE POINTER POSITIONS

Python provides the tell( ) method which is used to print the byte number at which the file pointer exists.

Example.

```
fileptr = open("file2.txt","r")
#initially the filepointer is at 0
print("The filepointer is at byte :",fileptr.tell())
#reading the content of the file
content = fileptr.read();
#after the read operation file pointer modifies. tell() returns the location of the fileptr.
print("After reading, the filepointer is at:",fileptr.tell())
```

Modifying file pointer position

In the real world applications, sometimes we need to change the file pointer location externally since we may need to read or write the content at various locations. For this purpose, the python provides us the seek() method which enables us to modify the file pointer position externally.

The syntax to use the seek() method is given below.

<file-ptr>.seek(offset[, from)

The seek() method accepts two parameters:

offset: It refers to the new position of the file pointer within the file.

from: It indicates the reference position from where the bytes are to be moved. If it is set to 0, the beginning of the file is used as the reference position. If it is set to 1, the current position of the file pointer is used as the reference position. If it is set to 2, the end of the file pointer is used as the reference position.

Example:

```
fileptr = open("file2.txt","r")
#initially the filepointer is at 0
print("The filepointer is at byte :",fileptr.tell())
#changing the file pointer location to 10.
fileptr.seek(10);
#tell() returns the location of the fileptr.
print("After reading, the filepointer is at:",fileptr.tell())
```

<u>PYTHON OS MODULE</u>

The os module provides us the functions that are involved in file processing operations like renaming, deleting, etc.

Renaming the file

The os module provides us the rename() method which is used to rename the specified file to a new name. The syntax to use the rename() method is given below.

rename(?current-name?, ?new-name?)

Example:

```
    import os;

    #rename file2.txt to file3.txt
os.rename("file2.txt","file3.txt")
```

Removing the file

The os module provides us the remove() method which is used to remove the specified file. The syntax to use the remove() method is given below.

remove(?file-name?)

Example:

```
    import os;

    #deleting the file named file3.txt
os.remove("file3.txt")
```

Creating the new directory

The mkdir() method is used to create the directories in the current working directory. The syntax to create the new directory is given below.

mkdir(?directory name?)

Example:

```
    import os;
    #creating a new directory with the name new
os.mkdir("new")
```

WRITING PYTHON OUTPUT TO THE FILES

In python, there are the requirements to write the output of a python script to a file. The **check_call()** method of module **subprocess** is used to execute a python script and write the output of that script to a file.

The following example contains two python scripts. The script file1.py executes the script file.py and writes its output to the text file **output.txt**

File.py

```
temperatures=[10,-20,-289,100]
def c_to_f(c):
    if c< -273.15:
        return "That temperature doesn't make sense!"
    else:
        f=c*9/5+32
        return f
for t in temperatures:
    print(c_to_f(t))
```

File.py

```
import subprocess

with open("output.txt", "wb") as f:
subprocess.check_call(["python", "file.py"], stdout=f)
```

The file related methods

The file object provides the following methods to manipulate the files on various operating systems.

SN	Method	Description
1	file.close()	It closes the opened file. The file once closed, it can?t be read or write any more.
2	File.fush()	It flushes the internal buffer.
3	File.fileno()	It returns the file descriptor used by the underlying implementation to request I/O from the OS.
4	File.isatty()	It returns true if the file is connected to a TTY device, otherwise returns false.
5	File.next()	It returns the next line from the file.
6	File.read([size])	It reads the file for the specified size.
7	File.readline([size])	It reads one line from the file and places the file pointer to the beginning of the new line.
8	File.readlines([sizehint])	It returns a list containing all the lines of the file. It reads the file until the EOF occurs using readline() function.

9	File.seek(offset[,from)	It modifies the position of the file pointer to a specified offset with the specified reference.
10	File.tell()	It returns the current position of the file pointer within the file.
11	File.truncate([size])	It truncates the file to the optional specified size.
12	File.write(str)	It writes the specified string to a file
13	File.writelines(seq)	It writes a sequence of the strings to a file.

MYSQL DATABASE ACCESS

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.

You can choose the right database for your application. Python Database API supports a wide range of database servers such as −

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix
- Interbase
- Oracle
- Sybase

The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following −

- Importing the API module.
- Acquiring a connection with the database.
- Issuing SQL statements and stored procedures.
- Closing the connection

MySQLdb

MySQLdb is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and is built on top of the MySQL C API.

Install MySQLdb

```
# !/usr /bin/python
import MySQLdb
```

Create Connection

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","root","root" )

# prepare a cursor object using cursor() method
cursor = db.cursor()
# disconnect from server
db.close()
```

If a connection is established with the datasource, then a Connection Object is returned and saved into db for further use, otherwise db is set to None. Next, db object is used to create a cursor object, which in turn is used to execute SQL queries. Finally, before coming out, it ensures that database connection is closed and resources are released.

Creating Database Table

Once a database connection is established, we are ready to create tables or records into the database tables using execute method of the created cursor.

Example

```
import MySQLdb
```

```
# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
        FIRST_NAME  CHAR(20) NOT NULL,
        LAST_NAME  CHAR(20),
        AGE INT,
        SEX CHAR(1),
        INCOME FLOAT )"""

cursor.execute(sql)

# disconnect from server
db.close()
```

INSERT Operation

It is required when you want to create your records into a database table.

Example

```
import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
        LAST_NAME, AGE, SEX, INCOME)
        VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
try:
  # Execute the SQL command
cursor.execute(sql)
  # Commit your changes in the database
db.commit()
except:
  # Rollback in case there is any error
db.rollback()

# disconnect from server
db.close()
```

READ Operation

READ Operation on any database means to fetch some useful information from the database.

Once our database connection is established, you are ready to make a query into this database. You can use either fetchone() method to fetch single record or fetchall() method to fetech multiple values from a database table.

- fetchone() − It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- fetchall() − It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.
- rowcount − This is a read-only attribute and returns the number of rows that were affected by an execute() method.

Example

```
import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

sql = "SELECT * FROM EMPLOYEE \
     WHERE INCOME > '%d'" % (1000)
try:
  # Execute the SQL command
cursor.execute(sql)
  # Fetch all the rows in a list of lists.
  results = cursor.fetchall()
  for row in results:
fname = row[0]
lname = row[1]
    age = row[2]
    sex = row[3]
    income = row[4]
    # Now print fetched result
    print "fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \
         (fname, lname, age, sex, income )
except:
  print "Error: unable to fecth data"

# disconnect from server
```

```
db.close()
```

Update Operation

UPDATE Operation on any database means to update one or more records, which are already available in the database.

The following procedure updates all the records having SEX as 'M'. Here, we increase AGE of all the males by one year.

Example

```
import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1
                          WHERE SEX = '%c'" % ('M')
try:
   # Execute the SQL command
cursor.execute(sql)
   # Commit your changes in the database
db.commit()
except:
   # Rollback in case there is any error
db.rollback()

# disconnect from server
db.close()
```

DELETE Operation

DELETE operation is required when you want to delete some records from your database. Following is the procedure to delete all the records from EMPLOYEE where AGE is more than 20 −

Example

```
import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
```

```
try:
   # Execute the SQL command
cursor.execute(sql)
   # Commit your changes in the database
db.commit()
except:
   # Rollback in case there is any error
db.rollback()

# disconnect from server
db.close()
```

COMMIT Operation
Commit is the operation, which gives a green signal to database to finalize the changes, and after this operation, no change can be reverted back.

Here is a simple example to call commit method.

db.commit()

ROLLBACK Operation
If you are not satisfied with one or more of the changes and you want to revert back those changes completely, then use rollback() method.

Here is a simple example to call rollback() method.

db.rollback()

Disconnecting Database
To disconnect Database connection, use close() method.

db.close()
If the connection to a database is closed by the user with the close() method, any outstanding transactions are rolled back by the DB. However, instead of depending on any of DB lower level implementation details, your application would be better off calling commit or rollback explicitly.