# PROGRAMMING LANGUAGES

...


**GROUP-7**

| | |
|---|---|
| Jitendra Kumar | 1401CS19 |
| Sathya Prakash | 1401CS06 |
| Anupam Das | 1401CS52 |

# A) INTRODUCTION

- Full name of the language is **Python** .
- Python is a widely used high level, general purpose, interpreted, dynamic programming language.
- Python was conceived in the late 1980s, and its implementation began in December 1989 by its creator **Guido van Rossum** at Centrum Wiskunde & Informatica (CWI) in the Netherlands.
- Many of Python's features originated from an interpreted language called **ABC**.
- Python was first released in February 1991 and was developed by
- Python Software Foundation.

- Python's  design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java .
- Top Websites like Youtube, Dropbox, Google, Quora, Yahoo Maps, Instagram are made upon Python(Django framework).
- Languages influenced by python are  Ruby,JavaScript,Falcon,Swift,Nim and many more developing languages.
- Python is also used for web development and for making games. Pygame is the tool which one can use to make game easily and pygame contains all libraries.
- Python 2.0 released in 2000 had major new features
    - Garbage collector  2. Support for unicode
- Python 3.0 - major,backwards incompatible release in 2008.

# B) PYTHON PROGRAMMING PARADIGMS

Python is **multi paradigm language.** You can write programs or libraries that are largely “procedural, object oriented, reflexive or functional in python” .

**OBJECT-ORIENTED :** Python is an object oriented where one can create different classes and objects(instances of class) and many methods.

```
class compute:
        def __init__(self):
                print ("")

        def comp_add(self,x,y):
                return ( x + y )

print ("\nDemonstrating object oriented python programming")
obj = compute()
sum_1_2 = obj.comp_add(1,2)
print("sum of 1 and 2 is : ")
print (sum_1_2)
```

```
Demonstrating object oriented python programming

sum of 1 and 2 is :
3
```

# PROCEDURAL-PROGRAMMING

Procedure is also known as routines or subroutines.In Procedural programming , computation is carried out by sequence of actions Procedural programming is also referred to as imperative programming. Statement oriented programming. Every statement changes the machine state .

```
class compute:
        def __init__(self):
                print ("")

        def add_list(self,Mylist):
                print ("\nDemonstrating procedural python programming")
                print ("-----------------------------------------")
                print("Sum of  the  element  in the  list : ")
                ans=0
                if type(mylist) is list:
                        for x in mylist:
                                ans+=x
                return ans
obj = compute()
mylist = [1, 2, 3]
print(obj.add_list(mylist))
```

```
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python procedural.py


Demonstrating procedural python programming
-----------------------------------------
Sum of  the  element  in the  list :
6
```

# FUNCTIONAL PYTHON PROGRAMMING

Python programmes written in functional style usually won't go to the extreme of avoiding all the I/O or all assignments ,instead they'll provide a functional interface but will use non-functional features internally.

```
class compute:
        def __init__(self):
                print ("")

        def comp_product(self,x,y): #functional
                print("-----------------------------------------------")
                print ("calculated using functional programing paradigm")
                print("-----------------------------------------------")
                return ( x * y )

obj = compute()
product  = obj.comp_product(4,5)
print product
```

```
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python functional.py

-----------------------------------------------
calculated using functional programing paradigm
-----------------------------------------------
20
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $
```

# C) How to execute program

- Major interpretations of python :

CPython :- CPython is the default, most widely used implementation of the Python programming language. It is written in C. CPython is a source code interpreter

**Jython** :- **Jython is a Java implementation of Python that combines expressive power with clarity. Jython is freely available for both commercial and non-commercial use and is distributed with source code.**

IronPython:- IronPython is an open-source implementation of the Python programming language which is tightly integrated with the .NET Framework. IronPython can use the .NET Framework and Python libraries, and other .NET languages can use Python code just as easily.

- PyPy :-
    a. PyPy is a Python interpreter and just-in-time compiler. PyPy focuses on speed, efficiency and compatibility with the original CPython interpreter.
- Then depending on operating system we have different ways of execution.

- Python interpreter program translate Python code into computer instructions.

# For Linux

1. Write code in text file and save it with ".py" extension(eg. first.py).
2. Now open the terminal and change the current directory to the directory where the first.py file is located.
3. Now just run python first.py.

   $ python <filename.py>

   Eg - $ python first.py
4. If python interpreter is installed in computer this command will execute without any error and will give the output otherwise first download python interpreter and then install it and then run above command.
5. In most of the linux systems python interpreter is installed by default.

# For Windows

1. Write code in text file and save it with .py extension (eg. first.py).
2. Open command prompt and navigate to directory in which first.py is saved using cd command.
3. Try >python first.py.
4. If you get message like python is not recognized then python interpreter isn't on path.Then try calling it like this (assuming Python2.6, installed in the usual location):
   > C:\Python26\python.exe first.py
5. In order to run program your operating looks in various places and tries to match the name of the program / command you typed with some programs along the way.

In windows : control panel > system > advanced > |Environment Variables| > system variables -> Path. this needs to include: C:\Python26; (or equivalent). If you put it at the front, it will be the first place looked. You can also add it at the end, which is possibly saner.Then restart your prompt, and try typing 'python'. If it all worked, you should get a ">>>" prompt.

- We can also use PyCharm to executive python code. PyCharm is python ide .

# D) Rules to Construct variables and different key-words in python

- Variables names must start with a *letter or an underscore.*
  Eg. _flag , flag_
- The remainder of your variable name may consist of letters, numbers and underscores.
  Eg variable_1 ,_777,etc
- Names are **case sensitive.** _flag_one , _FLAG_ONE , _Flag_One are each a different variable.
- Variables are used to store some value so each variable takes some space according to it's data type.
- Python variable do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to variable. The equal sign(=) is used to assign a value to variable.Variable assignment works left to right.

Eg. var1 = 25 #integer assignment

var2 = “string” #string assignment

var3 = 10.25 #float assignment

- We can also assign same value to multiple variables

  Eg. p=q=r=1;
- We can also assign as follows

  Var1,var2,var3 = 100,”string”,10.25
- We can’t use any keyword as a variable in python.
- Python swap values in single line and this applies to all objects in python.

  Eg. var1,var2 = var2,var1

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

# Keywords in python

| Keywords | Description |
|---|---|
| True,false | Results of logical and comparison operation in python |
| None | Absence of value or null value |
| and,or,not | Logical operators |
| as | To create alias while importing module |
| assert | Used for debugging purpose |
| break | Used to come outside loop |
| continue | To continue the loop |
| class | Used to define new user defined class |
| def | Used to define new method/function |
| del | Used to delete reference of an object |
| if,else,elif | Used for conditional branching |
| except,raise,try | Used with exceptions in python |
| finally | Used with try..except to close up resource |
| for,while | Used for looping |
| global | To declare variable inside function which is global |
| in | To test sequence contains value or not |
| is | For testing object identity |
| lambda | Used to create anonymous function |
| return | For returning from function |
| pass | Null statement in python |

# E) Type Systems in Python

- **Duck Typing** : Duck typing is concerned with establishing the suitability of an object for some purpose. With normal typing, suitability is assumed to be determined by an object's type only. In duck typing, an object's suitability is determined by the presence of certain methods and properties (with appropriate meaning), rather than the actual type of the object.

```
#duck typing
print("-----------------------------------------")
print ("Demonstrating duck typing")
print("-----------------------------------------")
_hello += 10
print (_hello)

_hello = float(_hello)
print ("Now Type of varibale _hello after type casting is : ",type(_hello))
```

```
-----------------------------------------
Demonstrating duck typing
-----------------------------------------
15
('Now Type of varibale _hello after type casting is : ', <type 'float'>)
-----------------------------------------
Demonstrating recursion
-----------------------------------------
('factorial by recursion ', 120)
```

- **Dynamic Typing** :
  - A language has dynamic typing when it does not associate values strictly with a specific type, but it is designed to "decide" what the type of a value should be at runtime, based on how you are attempting to use it.

```
#dynamically typed
print("-----------------------------------------------")
print ("Demonstrating dynamically typed")
print("-----------------------------------------------")
_hello = "hello python"
print ("Type of varibale _hello is : ",type(_hello))
_hello = 5
print ("Now Type of varibale _hello is : ",type(_hello))
```

```
-----------------------------------------------
Demonstrating dynamically typed
-----------------------------------------------
('Type of varibale _hello is : ', <type 'str'>)
('Now Type of varibale _hello is : ', <type 'int'>)
```

Here, the compiler did not complain that you are attempting to multiply a string by a number and refuse to compile the program (such as would happen in languages like C, C++, C# and Java). It produced code to forward the arguments count and 2 to the multiplication operator just like you asked and moved on. With the program now compiled, dynamic typing comes into effect at runtime. When the multiplication operator gets around to look at its operands, it checks to see what is the current type of each one. As before, it's a string and an int. But the operator knows that it can only multiply two integers (let's ignore floats for simplicity), so it has  produced an integer value from the string.

- **Strong Typing** : Strong typing probably means that variables have a well-defined type and that there are strict rules about combining variables of different types in expressions.

```
a   = 9
b = "9"
c = concatenate(a, b)   // produces "99"
d = add(a, b)           // produces 18
```

- This is not possible in python as it is a **strongly typed language**. To do something like this, we need to do casting.

```
a   = 9
b = "9"
c = concatenate(  str(a),  b)    // produces "99"
d = add(a,  int(b)  )              //produces 18
```

# F)Type casting:

- Python is moderately type-checked. Implicit conversion is defined for numeric types (as well as booleans), so one may validly multiply a complex number by a long integer (for instance) without explicit casting. However, there is no implicit conversion between (e.g.) numbers and strings; a string is an invalid argument to a mathematical function expecting a number.

```
x = '100'
y = '-90'
print("\nwithout typecasting\n")
print x + y
print("\nafter typecasting\n")
print int(x) + int(y)
print("\nfloat value\n")
print float(x) + float(y)
print("\nanother example of typecasting\n")
print int(float(x))
```

```
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python typecasting.py

without typecasting

100-90

after typecasting

10

float value

10.0

another example of typecasting

100
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python typecasting.py
```

# Scoping rules:

Python follows the LGB Rule

- Name references search at most three scopes: local, then global, then built-in.
- Name assignments create or change local names by default.
- "Global" declarations map assigned names to an enclosing module's scope.

When you use an unqualified name inside a function, Python searches three scopes—the local (L), then the global (G), and then the built-in (B)—and stops at the first place the name is found.

When you assign a name in a function (instead of just referring to it in an expression), Python always creates or changes the name in the local scope, unless it's declared to be global in that function.

When outside a function (i.e., at the top-level of a module or at the interactive prompt), the local scope is the same as the global—a module's namespace.

Example:

```
#global scope
X = 99              #X and func assigned in module : global
Def func(Y):        #Y and Z assigned in function : locals
#local scope
Z = X+Y             #X is not assigned ,so it's a global
Return Z
func(1)             #func in module:result = 100
```

Global Names : X,func

X is a global because it's assigned at the top level of the module file ;it can be referenced inside the function without being declared global , func is global for the same reason.

Local Names : Y,Z

Y and Z are both local to the function (and exist only while the function runs),because  they are both assigned a value in the function definition;Z by virtue of the = statement , and Y because arguments are always passed by assignment (more on this minute).

# Bindings:

In computing, a binding from a programming language to a library or operating system service is an application programming interface (API) providing glue code to use that library or service in a particular programming language.

Binding generally refers to a mapping of one thing to another. In the context of software libraries, bindings are wrapper libraries that bridge two programming languages so that a library written for one language can be used in another language. Many software libraries are written in system programming languages such as C or C++; in order to use these libraries from another (usually higher-level) language such as Python, a binding to the library must be created in that language, possibly requiring the recompilation of the language's code depending on the amount of modification necessary.

There are several ways to call code written in C from Python.First, there is the ctypes module in the standard library. It allows you to load a dynamic-link library (DLL on Windows, shared libraries .so on Linux) and call functions from these libraries, directly from Python. Such libraries are usually written in C.

Second, in case of CPython there is the Python/C API. It can be used in two major ways:

Examples:

Ctypes exports the cdll and on windows windll and oledll objects,for loading dynamic link libraries.

>>>from ctypes import *

>>>print windll.kernel32

<WinDLL ‘kernel32’ ,handle...at..>

>>>print cdll.msvcrt

<CDLL ‘msvcrt’, handle....at.....>

>>>libc = cdll.msvcrt

>>>Windows appends the usual .dll file suffix automatically

On Linux,it is required to specify the filename including the extension to load a library,so attribute access can not be used to load libraries.Either the LoadLibrary()

Method of the dll loaders should be used,or you should load the library by creating instance of CDLL by calling the constructor.

# G) Expressions

**Boolean Expressions** : A boolean expression (or logical expression) evaluates to one of two states true or false. Python provides the boolean type that can be either set to False or True. Examples:: Assume variable a holds 10 and variable b holds 20, then

| Operator | Description | Example |
|---|---|---|
| and Logical AND | If both the operands are true then condition becomes true. | (a and b) is true. |
| or Logical OR | If any of the two operands are non-zero then condition becomes true. | (a or b) is true. |
| not Logical NOT | Used to reverse the logical state of its operand. | Not(a and b) is false. |

| Operator | Description | Example |
|---|---|---|
| + Addition | Adds values on either side of the operator. | a + b = 30 |
| - Subtraction | Subtracts right hand operand from left hand operand. | a – b = -10 |
| * Multiplication | Multiplies values on either side of the operator | a * b = 200 |
| / Division | Divides left hand operand by right hand operand | b / a = 2 |
| % Modulus | Divides left hand operand by right hand operand and returns remainder | b % a = 0 |
| ** Exponent | Performs exponential (power) calculation on operators | a**b =10 to the power 20 |
| // | Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity): | 9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0 |

# Assignment Expressions : Expressions containing assignment operators.

| Operator | Description | Example |
|---|---|---|
| = | Assigns values from right side operands to left side operand | c = a + b assigns value of a + b into c |
| += Add AND | It adds right operand to the left operand and assign the result to left operand | c += a is equivalent to c = c + a |
| -= Subtract AND | It subtracts right operand from the left operand and assign the result to left operand | c -= a is equivalent to c = c - a |
| *= Multiply AND | It multiplies right operand with the left operand and assign the result to left operand | c *= a is equivalent to c = c * a |
| /= Divide AND | It divides left operand with the right operand and assign the result to left operand | c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a |
| %= Modulus AND | It takes modulus using two operands and assign the result to left operand | c %= a is equivalent to c = c % a |
| **= Exponent AND | Performs exponential (power) calculation on operators and assign value to the left operand | c **= a is equivalent to c = c ** a |
| //= Floor Division | It performs floor division on operators and assign value to the left operand | c //= a is equivalent to c = c // a |

# EXAMPLE :

```
# -*- coding: utf-8 -*-
import sys
a = 21
b = 10
c = 0

c = a + b
print "Line 1  Value of c is ", c
c += a
print "Line 2  Value of c is ", c
c *= a
print "Line 3  Value of c is ", c
c /= a
print "Line 4  Value of c is ", c
c  = 2
c %= a
print "Line 5  Value of c is ", c
c **= a
print "Line 6  Value of c is ", c
c //= a
print "Line 7  Value of c is ", c
```

```
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python arithmeticexpression.py
Line 1   Value of c is  31
Line 2   Value of c is  52
Line 3   Value of c is  1092
Line 4   Value of c is  52
Line 5   Value of c is  2
Line 6   Value of c is  2097152
Line 7   Value of c is  99864
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $
```

# H) VARIOUS FEATURES AND CONSTRUCTS :

- **Recursion:** Python supports recursion as similar to all other languages. Like other available programming languages , python also supports a function calling itself as a subroutine inside a python programme.

  Example :

```
def fact_recur(z):                  #recursion
        if(z==0):
                return 1
        return z*(fact_recur(z-1))

print("demonstrating recursion\n")
print("factorial of 5 is :" , fact_recur(5))
```

```
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python recursion.py
demonstrating recursion

('factorial of 5 is :', 120)
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $
```

- **Conditional Statement**: In order to write useful programs the conditional statements are highly useful to check the constraints on the variables.In python we can write all the conditional statements .

```
import sys
choice = str(sys.argv[1])
if choice == 'a':
    print("You chose 'a'.")
elif choice == 'b':
    print("You chose 'b'.")
elif choice == 'c':
    print("You chose 'c'.")
else:
    print("Invalid choice.")
```

```
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python conditional.py a
You chose 'a'.
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python conditional.py b
You chose 'b'.
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python conditional.py c
You chose 'c'.
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python conditional.py n
Invalid choice.
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $
```

- **Iterative Statements**:- Repeated execution of a set of statements is called "Iteration". Python supports all sort of iterative statements.

```
def fact_iter_while(z):
        ans=1;
        while(z > 1):
                ans = ans * z
                z= z-1
        return ans

print ("Demonstrating Iteration using For Loop\n")
print ("factorial by iteration for loop", fact_iter_while(6))
```

```
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python iterative.py
Demonstrating Iteration using For Loop

('factorial by iteration while loop', 720)
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python iterative.py
```

# I) VARIOUS DATA STRUCTURES SUPPORTED:

- The builtins data structures are: “lists, tuples, dictionaries, strings, sets and frozensets”.
- Lists, strings and tuples are ordered sequences of objects. Unlike strings that contain only characters, list and tuples can contain any type of objects. Lists and tuples are like arrays. Tuples like strings are immutables. Lists are mutables so they can be extended or reduced at will. Sets are mutable unordered sequence of unique elements whereas frozensets are immutable sets.
- Lists are enclosed in brackets: l = [1, 2, "a"]
- Tuples are enclosed in parentheses: t = (1, 2, "a")
- Tuples are faster and consume less memory.
- Dictionaries are built with curly brackets: d = {"a":1, "b":2}

- **List:**

```
>>> l = [1, 2, 3]
>>> l[0]
1
>>> l.append(4)
>>> l
[1, 2, 3, 4]
>>> l.index(3)
2
>>> l.insert(2,5)
>>> l
[1, 2, 5, 3, 4]
>>> l.remove(5)
>>> l
[1, 2, 3, 4]
>>> l.pop()
4
>>> l
[1, 2, 3]
>>> l.count(3)
1
```

```
>>> l.sort()
>>> l
[1, 2, 3]
>>> l.sort(reverse=True)
>>> l
[3, 2, 1]
>>> l.reverse()
>>> l
[1, 2, 3]
>>>
```

- **Tuples:**

```
>>> l=(1,2,3)
>>> l
(1, 2, 3)
>>> l[0]
1
>>> singleton=(1,)
>>> singleton
(1,)
>>> (1,)*5
(1, 1, 1, 1, 1)
>>> s1=(1,0)
>>> s1+=(2,)
>>> s1
(1, 0, 2)
>>> l.count(0)
0
>>> l.index(2)
1
>>> d = dict([('jan', 1), ('feb', 2), ('march', 3)])
>>> d['feb']
2
>>>
```

- **Dict:**

```
>>> d = {'first':'string value', 'second':[1,2]}
>>> d.keys()
dict_keys(['second', 'first'])
>>> d.values()
dict_values([[1, 2], 'string value'])
>>> d['first']
'string value'
>>>
>>> d = {'first':'string value', 'second':[1,2]}
>>> d.items()
dict_items([('second', [1, 2]), ('first', 'string value')])
```

```
>>> d = {'first':'string value', 'second':[1,2]}
>>> d.get('first')
'string value'
>>> d.popitem()
('second', [1, 2])
>>> d
{'first': 'string value'}
>>>
```

- **Strings:**

```
>>> text = 'The surface of the circle is 2 pi R = '
>>> text
'The surface of the circle is 2 pi R = '
>>> u"\u0041"
'A'
>>> text[0]
'T'
>>> "44".isdigit()
True
>>> "44".isalpha()
False
>>> "44".isalnum()
True
>>> "Aa".isupper()
False
>>> "aa".islower()
True
>>> "Aa".istitle()
True
>>> text.isspace()
False
>>> text.count('s')
2
>>>
```

- **Sets:**

```
>>> a = set([1, 1, 2, 3, 4])
>>> a
{1, 2, 3, 4}
>>> b = set([3, 4, 5, 6])
>>> a | b # Union
{1, 2, 3, 4, 5, 6}
>>> a & b # Intersection
{3, 4}
>>> a < b # Subset
False
>>> a - b # Difference
{1, 2}
>>> a ^ b # Symmetric Difference
{1, 2, 5, 6}
>>> c = a.intersection(b)
>>> c
{3, 4}
>>> c.issuperset(a)
False
>>> a.copy()
{1, 2, 3, 4}
>>>
```

- **Frozen sets:**

```
>>> a = frozenset([1, 2, 3])
>>> b = frozenset([2, 3, 4])
>>> a.union(b)
frozenset({1, 2, 3, 4})
>>>
```

Sets are mutable, and may therefore not be used, for example, as keys in dictionaries.

Another problem is that sets themselves may only contain immutable (hashable) values, and thus may not contain other sets.

Because sets of sets often occur in practice, there is the frozenset type, which represents immutable (and, therefore, hashable) sets.

# J) APPLICATIONS OF PYTHON

- Python offers many choices for web development:
  - Frameworks such as Django and Pyramid.
  - Micro-frameworks such as Flask and Bottle.
- Python is widely used in scientific and numeric computing:
  - SciPy is a collection of packages for mathematics, science, and engineering.
  - Pandas is a data analysis and modeling library.
  - IPython is a powerful interactive shell that features easy editing and recording of a work session, and supports visualizations and parallel computing.
- Python is often used as a support language for software developers, for build control and management, testing, and in many other ways.
  - SCons for build control.
  - Buildbot and Apache Gump for automated continuous compilation and testing.
  - Roundup or Trac for bug tracking and project management.

- Python's standard library supports many Internet protocols:
  - E-mail processing
  - Support for FTP, IMAP, and other Internet protocols.

- Desktop Graphical user interface (GUI)
  - The tk  GUI library is included with most binary distributions in python
  - GTK+
  - Microsoft foundation classes through the win32 extension

- The python package index lists thousands of third party modules of python

# K) A SAMPLE PROGRAM TO ILLUSTRATE ALL THE ABOVE POINTS

```
4  class compute:
5          def __init__(self):
6                  print ("")
7
8          def comp_add(self,x,y): #functional
9                  print("-------------------------------------------------")
10                 print ("calculated using functional programing paradigm")
11                 print("-------------------------------------------------")
12                 return ( x + y )
13
14         def sum_fifty(self):
15                 print("-------------------------------------------------")
16                 print ("Demonstrating Reflective programing paradigm")
17                 print("-------------------------------------------------")
18                 ans=0
19                 for i in range(50):
20                         ans += i
21                 print ("the sum from 1 to 50 is", ans)
22
23         def add_list(self,Mylist):#procedural
24                 print("-------------------------------------------------")
25                 print("Demonstrating procedural programing paradigm")
26                 print("-------------------------------------------------")
27                 print("Sum of  the  element  in the  list : ")
28                 ans=0
29                 if type(Mylist) is list:
30                         for x in Mylist:
31                                 ans+=x
32                 return ans
33
```

```
print("************************")
print ("Python Program Started")
print("************************")

comp = compute()
sum_1_2 = comp.comp_add(1,2)
print("sum of 1 and 2 is : ")
print (sum_1_2)
l = [1, 2, 3]
print(comp.add_list(l))


print("-------------------------------------------------")
print("Demonstrating use of set in python")
SET = set([1,1,2,2,3,4,5])
print SET
print("-------------------------------------------------")

print("-------------------------------------------------")
print("Demonstrating use of frozenset in python")
a = frozenset([1,2,3])
b = frozenset([2,3,4])
c = a.union(b)
print c

print("-------------------------------------------------")
print("-------------------------------------------------")
print ("Demonstrating use   of list  in python")
MYLIST = [1,2,3,4,5,6,7,8,9]
print MYLIST
print("-------------------------------------------------")
```

```
print("-------------------------------------------------")

print("-------------------------------------------------")
print("Demonstrating use  of string in python")
text = 'My name is  jitendra kumar'
print text[0]
print text[8]
print "44".isdigit()
print "44".isalpha()
print "44".isupper()
print "AA".isupper()
print "aa".islower()
print("-------------------------------------------------")
print("-------------------------------------------------")
print("Demostrating use of tuples in python")
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )
print ("tup1[0] : ", tup1[0])
print ("tup2[1:5] : " ,tup2[1:5])
print("-------------------------------------------------")
print("-------------------------------------------------")
print ("Demonstrating  use of dictionary in python")
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry
print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
print("-------------------------------------------------")

```

```
def fact_recur(z):              #recursion
        if(z==0):
                return 1
        return z*(fact_recur(z-1))

def fact_iter_for(z):
        ans=1;
        for y in range(z):
                ans = ans *(y+1)
        return ans
def fact_iter_while(z):
        ans=1;
        while(z > 1):
                ans = ans * z
                z= z-1
        return ans
print("-----------------------------------------------")
print("demonstrating recursion")
print("-----------------------------------------------")
print("factorial of 5 is :")
print(fact_recur(5))
print("-----------------------------------------------")

class_name = "compute"
method = "sum_fifty"
obj = globals()[class_name]()
getattr(obj,method)()

print("\n\n\n")

```

```
print("------------------------------------------------")
print ("Demonstrating Strongly Typed")
print("------------------------------------------------")
#strongly typed
try:
        _hello = "hello" + 5 + goodbye
        print (_hello)
except:
        print ("error due to strongly typed")
print("\n\n\n")

#dynamically typed
print("------------------------------------------------")
print ("Demonstrating dynamically typed")
print("------------------------------------------------")
_hello = "hello python"
print ("Type of varibale _hello is : ",type(_hello))
_hello = 5
print ("Now Type of varibale _hello is : ",type(_hello))

print("\n\n\n")

#duck typing
print("------------------------------------------------")
print ("Demonstrating duck typing")
print("------------------------------------------------")
_hello += 10
print (_hello)

_hello = float(_hello)
print ("Now Type of varibale _hello after type casting is : ",type(_hello))
```

```
156
157 print("-----------------------------------------------")
158 print ("Demonstrating recursion")
159 print("-----------------------------------------------")
160 print ("factorial by recursion ",fact_recur(5))
161 print("\n\n\n")
162
163 print("-----------------------------------------------")
164 print ("Demonstrating Iteration using For Loop")
165 print("-----------------------------------------------")
166 print ("factorial by iteration for loop", fact_iter_for(6))
167 print("\n\n\n")
168
169
170 print("-----------------------------------------------")
171 print ("Demonstrating Iteration using While Loop")
172 print("-----------------------------------------------")
173 print ("factorial by iteration while loop", fact_iter_while(7))
174 print("\n\n\n")
175
176
```

# OUTPUT

```
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $
jitendra@jitendra-Inspiron-5547 ~/Desktop/python $ python all_features.py
************************
Python Program Started
************************

------------------------------------------------
calculated using functional programing paradigm
------------------------------------------------
sum of 1 and 2 is :
3
------------------------------------------------
Demonstrating procedural programing paradigm
------------------------------------------------
Sum of  the  element  in the  list :
6
------------------------------------------------
Demonstrating use of set in python
set([1, 2, 3, 4, 5])
------------------------------------------------
------------------------------------------------
Demonstrating use of frozenset in python
frozenset([1, 2, 3, 4])
------------------------------------------------
------------------------------------------------
Demonstrating use   of list  in python
[1, 2, 3, 4, 5, 6, 7, 8, 9]
------------------------------------------------
```

```
------------------------------------------------
Demonstrating use  of string in python
M
i
True
False
False
True
True
------------------------------------------------
------------------------------------------------
Demostrating use of tuples in python
('tup1[0] : ', 'physics')
('tup2[1:5] : ', (2, 3, 4, 5))
------------------------------------------------
------------------------------------------------
Demonstrating  use of dictionary in python
dict['Age']:  8
dict['School']:  DPS School
------------------------------------------------
------------------------------------------------
demonstrating recursion
------------------------------------------------
factorial of 5 is :
120
------------------------------------------------

------------------------------------------------
Demonstrating Reflective programing paradigm
------------------------------------------------
('the sum from 1 to 50 is', 1225)
```

```
------------------------------------------------
Demonstrating Strongly Typed
------------------------------------------------
error due to strongly typed




------------------------------------------------
Demonstrating dynamically typed
------------------------------------------------
('Type of varibale _hello is : ', <type 'str'>)
('Now Type of varibale _hello is : ', <type 'int'>)




------------------------------------------------
Demonstrating duck typing
------------------------------------------------
15
('Now Type of varibale _hello after type casting is : ', <type 'float'>)
------------------------------------------------
Demonstrating recursion
------------------------------------------------
('factorial by recursion ', 120)




------------------------------------------------
Demonstrating Iteration using For Loop
------------------------------------------------
('factorial by iteration for loop', 720)
```

# THANK YOU !