PYTHON

- ✓ ***Python*** *is object-oriented, interpreted , dynamic & widely used high-level programming language for general-purpose programming.*
- ✓ ***Python*** *created by* ***Guido van Rossum*** *and first released in 1991 from python software foundation.*
- ✓ ***Python*** *is platform independent & open source language.*
- ✓ *File name extensions in python : .py .pyc .pyd .pyw .pyz*
- ✓ *Web site : www.python.org*
- ✓ *Python is a clear and powerful object-oriented programming language, comparable to Perl, Ruby, Scheme, or Java.*
- ✓ *Python is used to develop the different types of application such as web-apps, stand alone apps, enterprise apps, ERP and e-commerce application, scientific & numeric computing..etc.*

What is Python (Programming)?

Python is a general-purpose language. It has wide range of applications from Web development (like: Django and Bottle), scientific and mathematical computing (Orange, SymPy, NumPy) to desktop graphical user Interfaces (Pygame, Panda3D).

History of Python

Python is a fairly old language created by Guido Van Rossum. The design began in the late 1980s and was first released in February 1991.

Why Python was created?

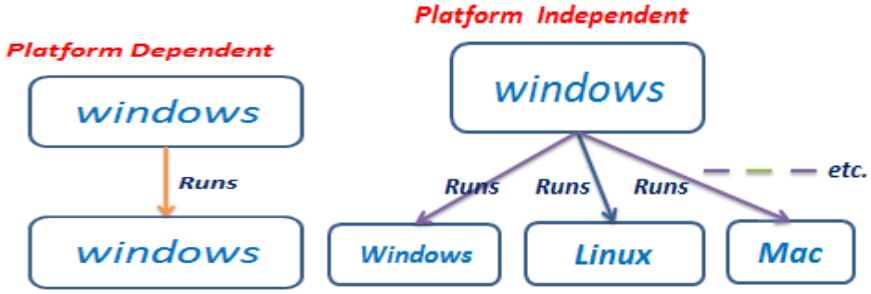
In late 1980s, Guido Van Rossum was working on the Amoeba distributed operating system group. He wanted to use an interpreted language like ABC (ABC has simple easy-to-understand syntax) that could access the Amoeba system calls. So, he decided to create a language that was extensible. This led to design of a new language which was later named Python.

Why the name Python?

No. It wasn't named after a dangerous snake. Rossum was fan of a comedy series from late seventies. The name "Python" was adopted from the same series "Monty Python's Flying Circus". Python was named for the BBC TV show Monty Python's Flying Circus.

Platform Independent vs. independent:-

- ✓ *Once we develop the application by using any one operating system(windows) that application runs only on same operating system is called platform dependency.*
  Ex :- C,CPP
- ✓ *Once we develop the application by using any one operating system(windows) that application runs on in all operating system is called platform independency.*
  Ex :- java



Python Version Release Date:

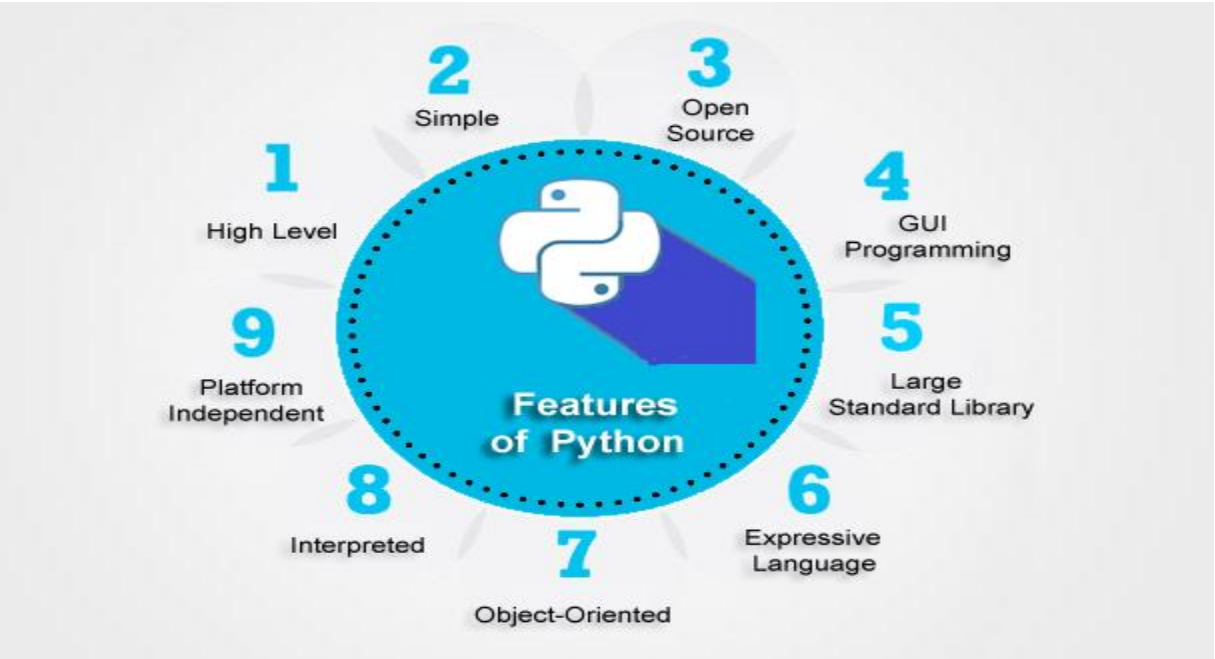
- *Python 1.0 - January 1994*
  - *Python 1.2 - April 10, 1995*
  - *Python 1.3 - October 12, 1995*
  - *Python 1.4 - October 25, 1996*
  - *Python 1.5 - December 31, 1997*
  - *Python 1.6 - September 5, 2000*

- *Python 2.0 - October 16, 2000*
  - *Python 2.1 - April 17, 2001*
  - *Python 2.2 - December 21, 2001*
  - *Python 2.3 - July 29, 2003*
  - *Python 2.4 - November 30, 2004*
  - *Python 2.5 - September 19, 2006*
  - *Python 2.6 - October 1, 2008*
  - *Python 2.7 - July 3, 2010*

- *Python 3.0 - December 3, 2008*
  - *Python 3.1 - June 27, 2009*
  - *Python 3.2 - February 20, 2011*
  - *Python 3.3 - September 29, 2012*
  - *Python 3.4 - March 16, 2014*
  - *Python 3.5 - September 13, 2015*
  - *Python 3.6 - December 23, 2016*

Features of python:

- ✓ *Easy to Use*
  - *It is a programmer friendly; it contains syntaxes just like English commands.*
  - *Uses an elegant syntax, making the programs you write easier to read.*
- ✓ *High Level Language*
  - *Python is a clear and powerful object-oriented programming language, comparable to Perl, Ruby, Scheme, or Java.*
- ✓ *Expressive Language*
  - *The code is easily understandable.*
- ✓ *Interpreted*
  - *The execution done in line by line format.*
- ✓ *Platform Independent*
  - *We can run this python code in all operating systems.*
- ✓ *Open Source*
  - *Python is free of cost, source code also available.*
- ✓ *Object-Oriented language*
  - *Python supports object-oriented programming with classes and multiple inheritance*
- ✓ *Huge Standard Library*
  - *Code can be grouped into modules and packages.*
- ✓ *GUI Programming*
  - *Graphical user interfaces can be developed using Python.*
- ✓ *Integrated*
  - *It can be easily integrated with languages like C, C++, JAVA etc.*
- ✓ *Extensible*
  - *Is easily extended by adding new modules implemented in a compiled language such as C or C++.*



Java vs. python:

1. ***Python simple langugae :***

I will write the codeto print Ratan world on screen in C, Java, Python. Decide your self which is simple.

Case 1: *printing Hello world…..*

In java

```
public class Test
{       public static void main(String[] args)
        {       System.out.println("ratan world");
        }
}
```

In python

```
Print ("ratan world")
```

Case 2: *performing addition operation….*

In java:

```
class Test
{       public static void main(String[] args)
        {       int a=10;
                int b=20;
                int c;
                c=a+b;
                System.out.println(c);
        }
}
```

In python :

```
a=10
b=20
c=a+b
print(c)
```

case 3: *Taking input from end-user…..*

in java:

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       Scanner s = new Scanner(System.in);
                System.out.println("etner a number");
                int n = s.nextInt();
                System.out.println(n);
        }
}
```

In python:

```
n = input("enter a number")
print(n)
```

2. Python is dynamically typed.

In java we must declare the type of the variables by using data type concept. But python is dynamically typed no need to declare the data type.

Case 1 : *data types declarations…*

In java : *byte short int long float double char boolean String class*

```
int eid=111;
String ename="ratan";
float esal=100000.34;
```

in python :

```
a=10        int
b=10.5      float
c=True      boolean
d="ratan"   String
```

3. Python Single line code

In python we have simple syntax so it will take less time to debug the syntax and more time programming.

```
Print  "ratan world"
```

In other languages it will take more lines of code but in python we can write the code in less number of lines.

Case 1 : *variable declaration single line of code in python.*

In java :

```
int eid=111;
String ename="ratan";
float esal=100000.34;
```

in python :

```
eid,ename,esal=111,"ratan",100000.34
```

case 2: *swaping two variables*

In java *:*

```
int a=10;
int b=20;
int temp;
temp=a;
a=b;
b=temp;
```

In python :

```
a,b=10,20
a,b=b,a
```

4. ***Python is opensource software***

   Free of cost & source code is open.

5. ***In Python English like commands (more readable)***

   in java *:* *String name="ratan";*
   System.out.println(name);

   In python : *name="ratan"*
   print(name)

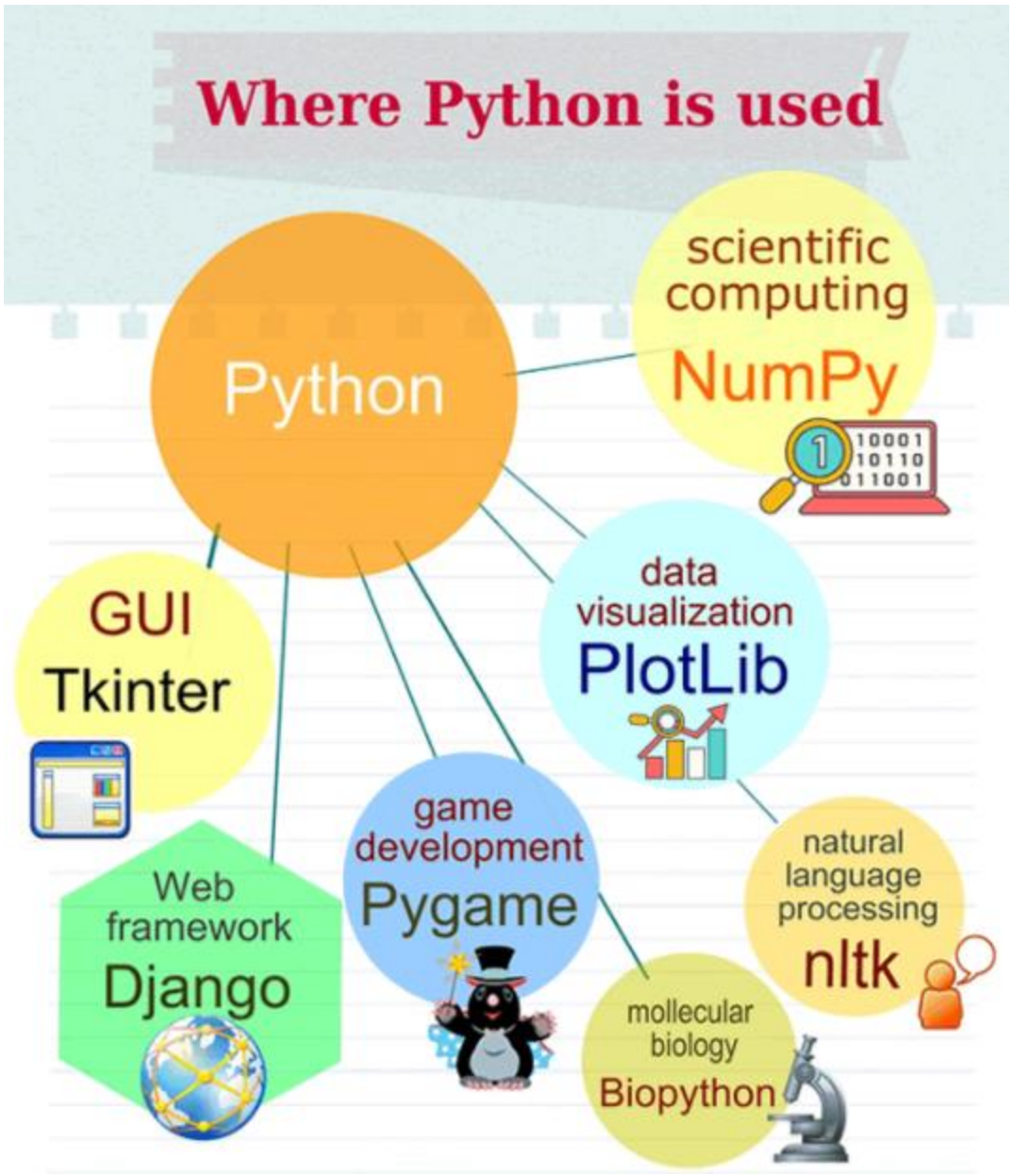
6. ***Importance of python & it is a general purpose language: build anything***

   Many big names such as Yahoo, IBM, Nokia, Google, Disney, NASA, Mozilla and much more rely on Python.

   Python is used to develop the different types of application such as web-apps, stand alone apps, enterprise apps, ERP and e-commerce application, scientific & numeric computing..etc.

7. ***Powerful data structures in python.***

   In java we have arrays & collections but in python we can handle the this data by using only List,Tuple,Set , Dictionaries.



*Web Technology uses **PYTHON** with **django***
*Mobile Apps uses **PYTHON** with **FLASK***
*Big Data Hadoop uses **PYTHON***

*Data Science Analytics uses **PYTHON***
*Amazon Web Services uses **PYTHON***
*Automation Tools DevOps uses **PYTHON***
*Reporting Tableau uses **PYTHON***
*Scripting and GUI Apps Uses **PYTHON***
*Scientific Applications uses **PYTHON** with **Numpy**, **Scipy***
Gaming, Networking, Embedded Systems, AI

Python has significant advantages in terms of

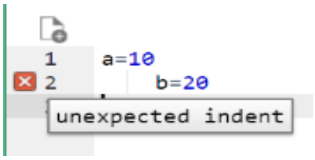
- ✓ *Pyramid, Flask, Bottle and Django are among the well known frameworks of Python.*
- ✓ *Pyramid is a small framework which makes real-world Web applications productive.*
- ✓ *Django is more mature and is one among the largest Web-based frameworks for Python But Django is very hard to customise and troubleshoot, in some situations, due to its all-encompassing nature. This is one of its drawbacks.*
- ✓ *Tornado is lighter in weight and has a few more features than Django. As I said earlier, Tornado is known for its high performance*
- ✓ *Building API services for Mobile (Flask)*
- ✓ *Doing Scientific computations (Numpy, Scipy)*
- ✓ *PlotLib would enable you to create data visualizations.*
- ✓ *Dealing with Data (Pandas)*
- ✓ *Numerous other things like scripting/ text processing/ machine learning/Natural Language Processing/ Text mining/ Web mining/ OpinionMining/ Sentiment analysis/ Big data system*

Python Coding Style:

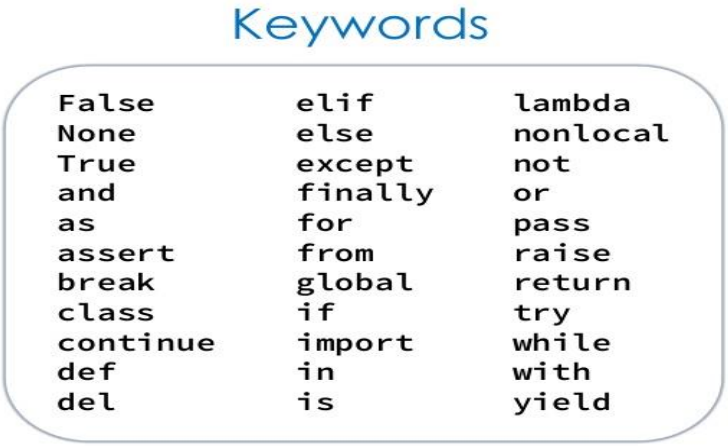
- ✓ *Use 4 spaces per indentation and no tabs.*
- ✓ *Do not mix tabs and spaces. Tabs create confusion and it is recommended to use only spaces.*
- ✓ *Maximum line length : 79 characters which help users with a small display.*
- ✓ *Use blank lines to separate top-level function and class definitions and single blank line to separate methods definitions inside a class and larger blocks of code inside functions.*
- ✓ *When possible, put inline comments (should be complete sentences).*
- ✓ *Use spaces around expressions and statements.*

Indentation:-

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



keywords *:(33 keywords in python) (Altered in different version of python))*



The above keywords are altered in different versions of python so some extra keywords are added or some might be removed. So it is possible to get the current version of keywords by typing fallowing code in the prompt.

*Example :**first.py***

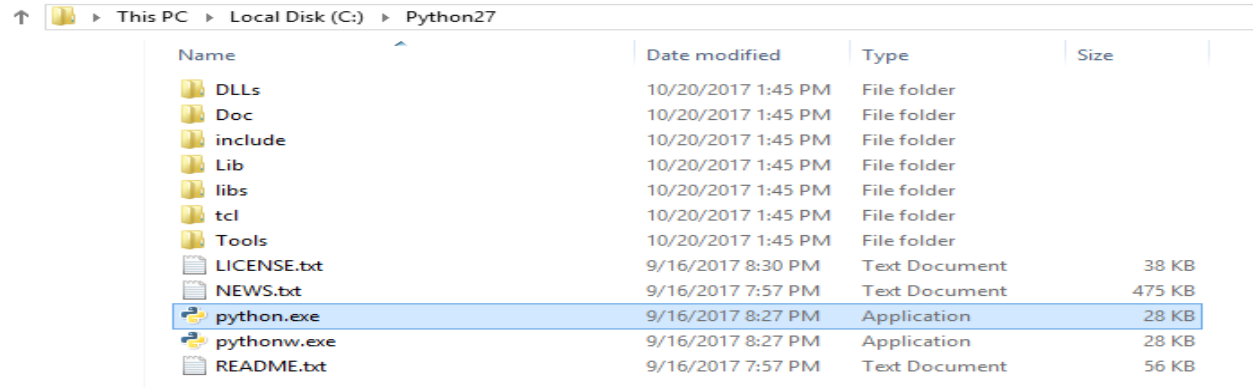
```
import keyword
print(keyword.kwlist)
```

G:\>python first.py

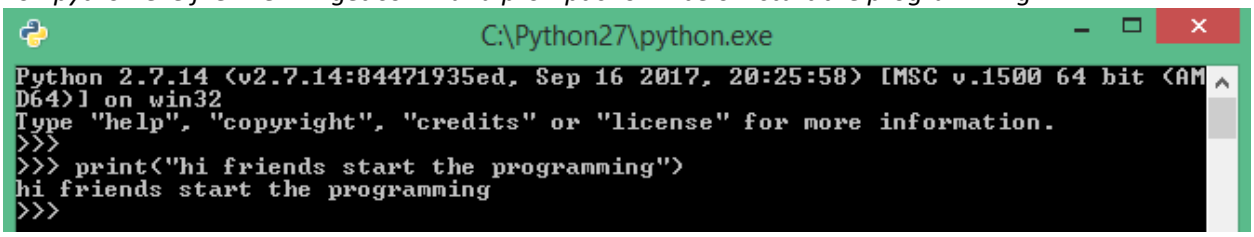
['and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'exec', 'finally', 'for', 'from', 'global', 'if', 'import', 'in','is', 'lambda', 'not', 'or', 'pass', 'print', 'raise', 'return', 'try', 'while', 'with', 'yield']

There are three different ways to run the python application

1. ***By using python command line***

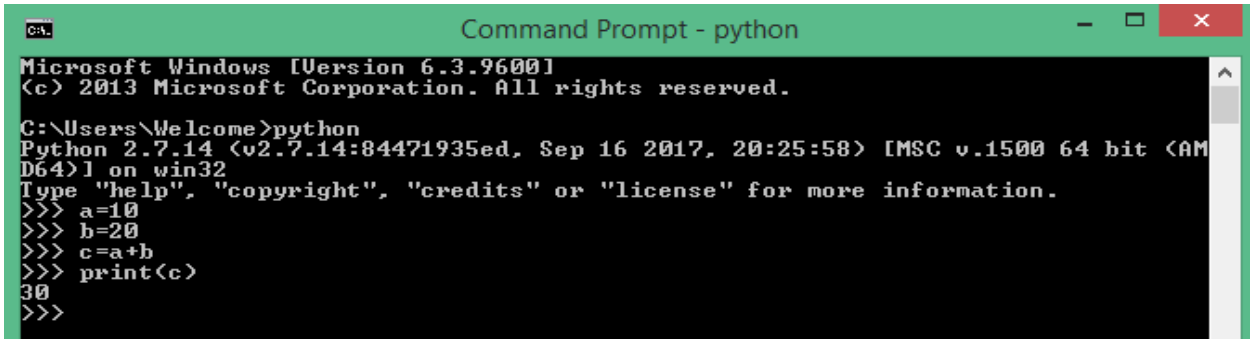


Just click on python.exe file w e will get command prompt shown below start the programming.



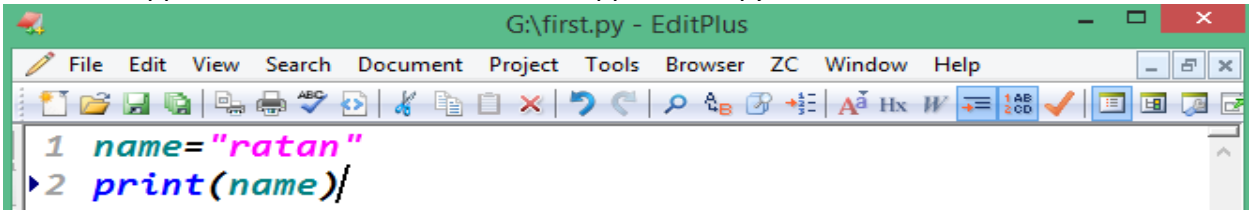
2. *By using system command prompt*

   Open the system command prompt type python command then we will python prompt.

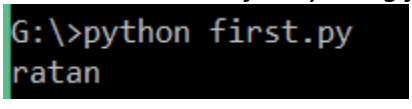


3. *Write the application in editor(notepad,editplus…etc) run from the command prompt.*

   Write the application in editor then save the application .py extension.



Execute the above file by using fallowing command.



4. *We can develop the application by using IDE like PyCharm.*

Comments:

- ✓ *Comments are used to write description about application logics to understand the logics easily.*
- ✓ *The main objective comments the code maintenance will become easy.*
- ✓ *The comments are non-executable code.*

There are two types of comments in python.

1. *Single line comments : write the description in single line & it starts with #*
   Syntax: # statement
2. *Multiline comments:*
   - *write the description in more than one line starts with “”” ends with : “””(triple quotes)*
   - *in python while writing the comments we can write double quote or single quote (“) or (‘)*

Syntax : “ ” ”
St-1
St-2
;;;;;;;;
St-n
“ ” ”

Syntax : ‘ ’ ’
St-1
St-2
;;;;;;;;
St-n
‘ ’ ’

```
print 10+20    #addition                single line comment
"""hi ratan sir                         multiline comments
how r u
start the python classes
"""
print 10*20
```

Python application with comments *Save the python application by **.py** extension.*

```
#first application in python2
print "Ratan world!"

#first application in python3
print ("Ratan world!" )
```

Example : Escape sequence character : *start with back slash*

```
# print("hi "ratan" sir how r u") SyntaxError: invalid syntax
print("hi \"ratan\" sir")
print("hi \'ratan\' sir")
print("hi \\ratan\\ sir")
print("hi\tratan\tsir")
print("hi\nratan\nsir")
```

Number data type : (int , float)

- ✓ *The integer numbers(2,4,30) have type int but python is dynamically typed no need to declare the type. integer can take a value of -1, 1, 0, etc*
- ✓ *The factional part (5.6, 4.7) have type float.And float can take 0.01, 1.2, etc*
- ✓ *Division always returns a floating point number.*
- ✓ *To represent the group of character use string data type.*

Example: Declaring a variable & printing a variables.

```
a=10
b=10.5
name='ratan'
print (a)
print (b)
print (name)
print (a,b,name)
```

Example: *Different ways to initialize the variables*

In python2

```
a=b=c=30;
i,j,k=10,20,30
x,y,z=10,10.5,"ratan"

print a+b+c
print i,j,k
print "{0} {1} {2}".format(x, y,z)
print "%d %g %s"%(x,y,z)
```

In python3

```
a=b=c=30;
i,j,k=10,20,30
x,y,z=10,10.5,"ratan"

print (a+b+c)
print  (i,j,k)
print ("{0} {1} {2}".format(x,y,z))
print ("%d %g %s"%(x,y,z))
```

Example :

```
eid,ename,esal=111,'ratan',10000.45
print ("Emp id=",eid)
print ("Emp name=",ename)
print ("Empsal=",esal)
print  (eid,ename,esal)
print ("{0} {1} {2}".format(eid,ename,esal))
print ("%d %s %g"%(eid,ename,esal))
print ("Emp id={0} \n Emp name={1} \n Empsal={2}".format(eid,ename,esal))
```

Example : *In python it is possible to combine same data type values*

Mixing operators between numbers and strings is not supported:

```
one = 1
two = 2
hello = "hello"
print(one + two + hello)
```

Example : *Strings are defined either with a single quote or a double quotes.*

```
mystring = 'hello'
print(mystring)
mystring = "hello"
print(mystring)
```

Example : *Swapping means interchanging the values of two variables.*
eg : if x is 10 and y is 5 then after swapping x will be 5 and y will be 10.

```
x = 10
y = 5
x,y = y,x
print (x)
print (y)
```

Re-declare a Variable: *You can re-declare the variable even after you have declared it once*

```
a=10
print (a)
a=100
print (a)
```

Output :
10
100

Deleting a variable *: You can also delete variable using the command* ***del*** *command.*

```
a=10
print(a)
del a    # deleting a variable
print(a)
E:\>python first.py
10
NameError: name 'a' is not defined
```

Finding type of value: *To know the type of the value use type function.*

```
print type('Hello World')
print type(8)
print type(8.0)
print type(True)
```

output :

```
E:\>python first.py
<type 'str'>
<type 'int'>
<type 'float'>
<type 'bool'>
```

```
a,b,c=10,10.5,'ratan'
print type(a)
print type(b)
print type(c)
print type(True)
```

output :

```
E:\>python first.py
<type 'int'>
<type 'float'>
<type 'str'>
<type 'bool'>
```

Formatting data with the % & {} :

%d *int* ***%s*** *string* ***%f/%g*** *floating point*

```
name,age,sal = "balu",24,10000.34
print("%s  age : %d  salary is %g"%(name, age, sal))
print("{} age :{}  salary is :{}".format(name, age, sal))
print("{0} age :{1}  salary is :{2}".format(name, age, sal))
```

We can represent int values in the following ways

1. *Decimal form*
2. *Binary form*
3. *Octal form*
4. *Hexa decimal form*

Decimal form(base-10): *It is the default number system in Python The allowed digits are: 0 to 9*

Eg: a =10

Binary form(Base-2): *The allowed digits are : 0 & 1 ,Literal value should be prefixed with 0b or 0B*

Ex:

```
a =0B1111
print(a)  # 15
a= 0b0101
print(a) #5
```

Octal Form(Base-8): *The allowed digits are : 0 to 7 Literal value should be prefixed with 0o or 0O.*

Ex:

```
a=0o123
print(a)
a=0O111
print(a)
```

Hexadecimal System: *Hexadecimal system is base 16 number system.*

Note : ***A number with the prefix '0b' is considered binary, '0o' is considered octal and '0x' as hexadecimal.***

ex: conversion of decimal to binary,octal, hexadecimal.

In this program, we have used built-in functions bin(), oct() and hex() to convert the given decimal number into respective number systems.

```
dec = int(input("Enter a decimal number: "))
print(bin(dec),"in binary.")
print(oct(dec),"in octal.")
print(hex(dec),"in hexadecimal.")
```

ex:

```
a,b,c,d=10,0o10,0X10,0B10
print(a,b,c,d)
```

ex: Finding ascii values.

```
c = input("Enter a character: ")
print("The ASCII value of '" + c + "' is",ord(c))
```

ex: Type cconversion process

```
print(int(123.987))
print(int(True))
print(int(False))
print(float(False))
print(str(False))

a=10
b=20
print(str(a)+str(b))

print(int("10"))
print(float("10"))
#print(int("10.6"))   #ValueError: invalid literal for int() with base 10: '10.6
```

Variables:

- ✓ *A variable is nothing but a reserved memory location to store values.*
- ✓ *Variables are used to store the data by using this data we will achieve project requirement.*
- ✓ *memory allocated when the values are stored in variables.*
- ✓ *Every variable must have some type i.e. ,*
  - *Number*
  - *String*
  - *List*
  - *Tuple*
  - *Dictionary….etc*

There are two types of variables in python,

1. *Local variables*
2. *Global variables*

- ✓ *In python when we want use variable rest of the application or module you declare it global.*
- ✓ *If you want to use the variable within the function use local variable.*

***ex** :*

- ✓ *The variables which are declared inside the function is called local variable.*
- ✓ *The variables which are declared outside of the function is called global variables.*

```
x,y=10,20        #global variables
def add():
        a,b=100,200      #local variables
        print (a+b)
        print(x+y)
def mul():
        a,b=100,200      #local variables
        print (a*b)
        print(x*y)
add()            # function calling
mul()            # function calling
print (x+y)
```

ex: *The scope of the local variable only within the function , if we are calling outside of the function we will get error message.*

```
def f():
        name = "ratan"
        print (s)
f()
print (s)    # NameError: name 's' is not defined
```

ex:

```
s = "sunny"
def f():
   s = "ratan
   print (s)
f()
print (s)
```

ex *: Using the keyword global, you can reference the a global variable inside a function*

```
str="ratan"      #global variables

def wish():
        global str        #reference global variable inside the function
        str="good morning"
        print str

wish()            # calling a method
print str
```

ex:

```
def a():
  global foo
  foo = 'A'

def b():
  global foo
  foo = 'B'

b()
a()
print (foo)
```

ex: ***inner functions*** *: declaring the function inside the another function.*

```
def ex4():
   var_outer = 'foo'
   def inner():
      var_inner = 'bar'
      print (var_outer)
      print (var_inner)

   inner()   #calling of inner function
   print (var_outer)

   ex4()
```

ex:

- ✓ *inside the inner function to represent outer function variable use **nonlocal** keyword**.***
- ✓ *inside the function to represent the global value use **global** keyword.*

```
def ex4():
   var_outer = 'ratan'
   def inner():
      nonlocal var_outer
      var_outer="anu"
      print (var_outer)

   inner()   #calling of inner function
   print (var_outer)

ex4()
```

ex:

```
name='ratan'
def ex4():
   var_outer = 'ratan'
   def inner():
      nonlocal var_outer
      global name
      name="ratanit"
      var_outer="anu"
      print (var_outer)

   inner()   #calling of inner function
   print (var_outer)

ex4()
print(name)
```

Ex: ***non-local vs global***

```
def disp():
    name="ratan"

    def localdisp():
        name = "anu"

    def nonlocaldisp():
        nonlocal name
        name = "durga"

    def globaldisp():
        global name
        name = "sunny"

    localdisp()
    print("local Name:", name)    #  ratan
    nonlocaldisp()
    print("nonlocal Name:", name) # durga
    globaldisp()
    print("global Name:", name)   # durga

disp()
print("global :", name)   # sunny
```

Ex: ***non-local vs global : Assignment : write the output***

```
def scope_test():
    spam = "test spam"

    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    do_local()
```

```
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)   # global spam
```

ex: Assignment : write the output

```
a = 10

# Uses global because there is no local 'a'
def f():
    print ('Inside f() : ', a)

# Variable 'a' is redefined as a local
def g():
    a = 20
    print ('Inside g() : ',a)

# Uses global keyword to modify global 'a'
def h():
    global a
    a = 30
    print ('Inside h() : ',a)

# Global scope
print ('global : ',a)
f()
print ('global : ',a)
g()
print ('global : ',a)
h()
print ('global : ',a)
```

ex: Assignment : write the output

```
a_var = 10
b_var = 15
e_var = 25
d_var = 100
def a_func(a_var):
   print("in a_func a_var =", a_var)
   b_var = 100 + a_var
   d_var = 2 * a_var
   print("in a_func b_var =", b_var)
   print("in a_func d_var =", d_var)
   print("in a_func e_var =", e_var)
   return b_var + 10

c_var = a_func(b_var)

print("a_var =", a_var)
print("b_var =", b_var)
print("c_var =", c_var)
print("d_var =", d_var)
```

ex: Assignment : write the output

```
def foo(x, y):
   global a
   a = 42
   x,y = y,x
   b = 33
   b = 17
   c = 100
   print (a,b,x,y)

a,b,x,y = 1,15,3,4
foo(17,4)
print (a,b,x,y)
```

Getting input from the end-user: in python 2.7

There are two ways to get the input,

1. *By using input function*
2. *By using raw_input function*

ex-1 :*Taking data from end-user by using* ***input*** *function.*

- ✓ *If the user gives an integer value, the input function returns integer value*
- ✓ *If the user gives float value, the input function returns float value*
- ✓ *If the user gives string value, the input function returns string value*

```
num1 = input('Enter first number: ')
num2 = input('Enter second number: ')
```

```
# Add two numbers
sum=num1+num2
# Display the sum
Print  sum
```

case 1 : int data

```
E:\>python First.py
Enter First Number :10
Enter Second Number :20
Addition= 30
```

case 2: float data

```
E:\>python First.py
Enter First Number :10.5
Enter Second Number :20.4
Addition= 30.9
```

case 3: String data

```
E:\>python First.py
Enter First Number :'ratan'
Enter Second Number :'anu'
Addition= ratananu
```

case 4: number data

```
E:\>python First.py
Enter First Number :10
Enter Second Number :10.5
Addition= 20.5
```

E:\>python First.py

```
Enter First Number :10
Enter Second Number :'ratan'
TypeError: unsupported operand type(s) for +:
'int' and 'str'
```

ex-2 : *Taking data from end-user by using* ***raw_input*** *function it return data only in String format.*

- ✓ *We entered 10 to raw_input which will return "10" (a string) and not 10 int.*
- ✓ *We entered 12.32 to raw_input which will return '12.32' (a string) and not 12.32 decimal .*

```
num1 = raw_input('Enter first number: ')
num2 = raw_input('Enter second number: ')
# Add two numbers
sum=num1+num2
# Display the sum
Print sum
```

```
E:\>python First.py
Enter first Number :100
Enter Second Number :200
addition=100200
```

```
E:\>python First.py
Enter first Number :10.5
Enter Second Number :20.6
addition=10.520.6
```

Type conversion in python:

Getting data from user by using raw_input function in the form of String then converting,

- ✓ *String to int*
  - *num1 = int(raw_input('Enter first number: '))*
- ✓ *String to float*
  - *num2 = float(raw_input('Enter second number: '))*

ex -3 : *Type conversion*

```
num1 = int(raw_input("Enter first Number :"))
num2 = int(raw_input("Enter Second Number :"))
add=num1+num2
print "addition=",add
```

ex- 4 : *Type conversion process*

```
num1 = raw_input("Enter first Number :")
num2 = raw_input("Enter Second Number :")
add= float(num1)+int(num2)
print "addition=",add
```

ex- 5 *:- Taking the data by using both functions* ***input & raw_input***

```
eid = input("Enter emp id:")
ename = raw_input("Enter emp name:")
esal = input("enter empsal:")
print "emp id=",eid
print "emp name=",ename
print "empsal=",esal
print eid,ename,esal
print ("Emp id={0}  Emp name={1}  Emp sal={2}").format(eid,ename,esal)
```

Getting input from the end-user python2 vs python3*:*

- ✓ *In python2 we have two functions to take the input from the end-use*
  - *Input function : Takes any type of data*
  - *raw_input function : Takes only string data*

- ✓ *In python3 we have only one function to take the input from the end-user.*
  - *Input function : Takes only string data*

Note *: In Python 3, 'raw_input()' is changed to 'input()'. Thus, 'input()' of Python 3 will behave as 'raw_input()' of Python 2.*

ex - 6 : *In python3*

```
num1 = input('Enter first number: ')       #10
num2 = input('Enter second number: ')   #10
sum= num1+ num
print ("addition of {0} , {1} is {3} =".format(num1,num2,sum))      #1010
```

ex -7: *Type conversion process*

```
num1 = int(input("Enter First Number:"))
num2 = int(input("Enter Second Number:"))
add=num1+num2
print ("Addition:",add)
```

ex - 8 : *Type conversion process*

```
num1 = input("Enter First Number:")
num2 = input("Enter Second Number:")
add=float(num1)+int(num2)
print ("Addition:",add)
```

ex-9: Type conversion process

```
a,b=10.5,20.6
sum = int(a)+int(b)
print (sum)
```

ex-10 : ValueError

```
num = input("Enter a Number:")
print (int(num))

Enter a Number: 10.4
ValueError: invalid literal for int() with base 10: '10.4'
```

example *: Display calendar*

```
import calendar
# Enter the month and year
yy = int(input("Enter year: "))
mm = int(input("Enter month: "))

# display the calendar
print(calendar.month(yy,mm))
```

none :-

- ✓ *none is a special constant in python to represent absence of value or null value.*
- ✓ *None is not a* ***0 , false, []***
- ✓ *Void functions that don't return anything will return a none object automatically.*

Case 1: Test.py

```
def add():
 a=10
 b=20
 c=a+b

x = add()
print x
```

```
G:\>python Test.py
None
```

case 2: Test.py

```
def add(a):
if (a % 2) == 0:
        return True

x = add(3)
print x
```

```
 G:\>python Test.py
None
```

pass *Statements :*

The pass statement does nothing. It can be used when a statement is required syntactically but the program requires no action. For example:

Case: *while True:*
pass

Case: *This is commonly used for creating minimal classes:*
class MyEmptyClass:
pass

Case: *Another place pass can be used is as a place-holder for a function or conditional body when you are working on new code, allowing you to keep thinking at a more abstract level. The pass is silently ignored.*

```
def initlog(*args):
    pass   # Remember to implement this!
```

Ex:

```
a = int(input("Enter first value:"))
b = int(input("Enter first value:"))
def absolute_value(n):
  if n < 0:
    n = -n
  return n
if absolute_value(a) == absolute_value(b):
  print("The absolute values of", a, "and", b, "are equal.")
else:
  print("The absolute values of", a, "and", b, "are different.")
```

Functions Def keyword:

Step 1: *Declare the function with the keyword **def** followed by the function name.*
Step 2: *Write the arguments inside the opening and closing parentheses of the function, and end the declaration with a colon.*
Step 3: *Add the program statements to be executed*
Step 4: *End the function with/without return statement.*

Syntax :

```
def  function_name(parameters):
        """doc string"""
        statement(s)
```

ex:

```
def userDefFunction (arg1, arg2, arg3 ...):
   program statement1
   program statement3
   program statement3
   ....
  return
```

There are two types of functions

1. *Built-in functions - Functions that are built into Python.*
2. *User-defined functions - Functions defined by the users themselves.*

Advantages of functions :

- ✓ *User-defined functions are reusable code blocks; they only need to be written once, then they can be used multiple times.*
- ✓ *These functions are very useful, from writing common utilities to specific business logic.*
- ✓ *The code is usually well organized, easy to maintain, and developer-friendly.*
- ✓ *A function can have dif types of arguments & return value.*

ex :
```
def disp():
  print("hi ratan sir")
  print("hi anu")

disp()   # function calling
```

ex: *To specify no body of the function use* ***pass statement.***
```
def disp():
  pass

disp()
```

ex: *one function is able to call more than one function.*
```
def happyBirthday(person):
  print("Happy Birthday dear ",person)

def main():
   happyBirthday('ratan')
   happyBirthday('anu')

main()
```

ex: inner functions *: declaring the function inside the another function.*
```
def ex4():
   var_outer = 'foo'
   def inner():
      var_inner = 'bar'
      print (var_outer)
      print (var_inner)

   inner()   #calling of inner function
   print (var_outer)
```

```
ex4()
```

ex:

- ✓ *inside the inner function to represent outer function variable use **nonlocal** keyword.*
- ✓ *inside the function to represent the global value use **global** keyword.*

```
def ex4():
    var_outer = 'ratan'
    def inner():
        nonlocal var_outer
        var_outer="anu"
        print (var_outer)

    inner()   #calling of inner function
    print (var_outer)

ex4()
```

ex:

```
name='ratan'
def ex4():
    var_outer = 'ratan'
    def inner():
        nonlocal var_outer
        global name
        name="ratanit"
        var_outer="anu"
        print (var_outer)

    inner()   #calling of inner function
    print (var_outer)

ex4()
print(name)
```

function vs arguments:-

1. *default arg*
2. *required arg*
3. *keyword argument*
4. *variable argument*

Default arguments:

When we call the function if we are not passing any argument the default value is assigned.

ex : *defempdetails(eid=1,ename="anu",esal=10000):*

```
        print ("Emp id =", eid)
        print ("Emp name = ", ename)
        print ("Empsal=", esal)
        print("**********")

    empdetails()
    empdetails(222)
    empdetails(333,"durga")
    empdetails(111,"ratan",10.5)
```

ex :

```
    def defArgFunc( empname, emprole = "Manager" ):
        print ("Emp Name: ", empname)
        print ("Emp Role ", emprole)

    print("Using default value")
    defArgFunc("Ratan")
    print("**********")
    print("Overwriting default value")
    defArgFunc("Anu","CEO")
```

Required arguments:

Required arguments are the mandatory arguments of a function. These argument values must be passed in correct number and order during function call.

```
        Def show(a,b):      #function declaration
        show(10,20)         #function calling
```

ex 1: *def reqArgFunc(empname):*

```
  print ("Emp Name: ", empname)

reqArgFunc("ratan")
```

ex:

```
def add(a,b):
   print(a+b)

add(10,20)
add(10.5,20.4)
add("ratan","anu")
add(10,10.5)
add("ratan",10)
```

ex 2:
```
def addition(x,y):
        print x+y

a,b=10,20
addition(100 ,200)
addition(a,b)
```

Keyword arguments / named arguments:

The keywords are mentioned during the function call along with their corresponding values. These keywords are mapped with the function arguments so the function can easily identify the corresponding values even if the order is not maintained during the function call.

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  empdetails(name,role):
empdetails ( name = "ratan", role = "Manager")          #function calling
empdetails (role = "developer" ,  name = "anu”)         #function calling
```

ex:
```
def msg(id,name):
        print id
        print name

msg(id=111,name='ratatn')
msg(name='anu',id=222)
```

ex:
```
def disp(eid,ename):
   print(eid,ename)

disp(eid=111,ename="anu")
disp(ename="ratan",eid=222)
disp(333,"durga")
disp(444,ename="sravya")
disp(eid=555,"aaa")  #SyntaxError: positional argument follows keyword argument
```

ex:
```
def emp(eid,ename="ratan",esal=10000):
  print(eid,ename,esal)

emp(111)
emp(222,ename="anu")
#emp(ename="anu",222)  #error SyntaxError: positional argument follows keyword
argument
```

```
emp(222,esal=100,ename="anu")
```

- ✓ *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.*
- ✓ *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.*

Variable number of arguments:

This is very useful when we do not know the exact number of arguments that will be passed to a function. Or we can have a design where any number of arguments can be passed based on the requirement.

```
def varlengthArgs(*varargs):
varlengthArgs(10,20,30,40)          #function calling
```

ex:

```
def disp(*var):
   for i in var:
     print("var arg=",i)

disp()
disp(10,20,30)
disp(10,20.3,"ratan")
```

Function return type:-

Example 1:

```
def addition(x,y):
        return a+y

a,b=10,20
sum=addition(a ,b)
print (sum)
```

ex : *If the functions not returns the value but if we are trying to hold the values it retuns none as a default value.*

```
def add():
 a=10
 b=20
 c=a+b
```

```
x = add()
print x

G:\>python Test.py
None
```

Ex:

```
def hello():
   print('Hello!')

def print_welcome(name):
   print('Welcome,', name)

def area(width, height):
   return width * height

def positive_input(prompt):
   number = float(input(prompt))
   while number <= 0:
      print('Must be a positive number')
      number = float(input(prompt))
   return number

name = input('Your Name: ')
hello()
print_welcome(name)
print('To find the area of a rectangle, enter the width and height below.')
w = positive_input('Width: ')
h = positive_input('Height: ')

print('Width =', w, ' Height =', h, ' so Area =', area(w, h))
```

Ex:

```
def factorial(n):
   if n <= 1:
      return 1
   return n * factorial(n - 1)
```

```
n = int(input("enter a number to perform factorial:"))
print("%d factorial is ="%(n),factorial(n))
```

Names can only contain upper/lower case digits (A-Z, a-z), numbers (0-9) or underscores _;
Names cannot start with a number;
Names cannot be equal to reserved keywords:

Python control flow statements

If-else statement :

- ✓ *if the condition true if block executed.*
- ✓ *if the condition false else block executed.*

Syntax :

```
if(condition):
    Statement(s)
else :
    statement(s)
```

ex:

```
a=10
if(a>10):
    print("if body")
else:
    print("else body")
```

ex : *In python 0=false 1=true*

```
if(1):
    print("hi ratan")
else:
    print("hi anu")
```

ex: *In python Boolean constants start with uppercase character**.***

```
if(False):
    print ("true body")
else:
    print ("false body")
```

ex:

```
year = 2000
if year % 4 == 0:
```

```
    print("Year is Leap")
else:
    print("Year is not Leap")
```

elif statement :

- ✓ *The keyword 'elif' is short for 'else if'*
- ✓ *An if … elif … elif … sequence is a substitute for the switch or case statements found in other languages*

Syntax:

```
if expression:
    statement(s)
elif expression:
    statement(s)
elif expression:
    statement(s)
...
else:
    statement(s)
```

Ex:

```
number = 23
guess = int(input("Enter an integer : "))
if guess == number:
    print("Congratulations, you guessed it.")
elif guess < number:
    print("No, it is a little higher number")
else:
    print("No, it is a little lower number")
print("rest of the app")
```

ex:

```
x = int(input("Please enter an integer: "))
if x > 0:
  print ("Positive")
elif x == 0:
  print ("Zero")
```

```
else:
  print ("Negative")
```

ex:

```
x = int(input("Please enter an integer: "))
if x < 0:
  print ("x is negative")
elif x % 2:
  print ("x is positive and odd")
else:
  print ("x is even and non-negative")
print("process is done")
```

for loop *:*

- ✓ *Used to print the data **n** number of times based on conation.*
- ✓ *If you do need to iterate over a sequence of numbers,use the built-in function range().*

syntax : for <temp-variable> in <sequence-data>:
statement(s)

range() function :

range(10)	*1-10*
range(5, 10)	*5 through 9*
range(0, 10, 3)	*0, 3, 6, 9*
range(-10, -100, -30)	*-10, -40, -70*

Syntax:

```
for iterator_name in range(10):
    …statements…

for iterator_name in range(start,end):
    …statements…

for iterator_name in range(start,stop,increment):
    …statements…
```

ex:

```
for x in range(10):
  print("ratan World",x)

for x in range(8,10):
  print("durga World",x)

for x in range(3,10,3):
  print("anu world",x)
```

```
for x in range(-20,-10):
  print("ratan sir",x)

for x in range(-20,-10,3):
  print("ratan sir",x)

for i in range(-10,-100,-15):
   print(i)

for i in range(10, 0, -2):
   print (i)
```

Loops with else block :

ex: ***else*** *is always executed if the loop executed normally termination.*

```
for i in range(1, 5):
   print(i)
else:
   print('The for loop is over')
```

else block is not executed in two cases

case 1: if the exception raised in loop else block not executed.

```
for x in range(10):
  print("ratan world",x)
  print(10/0)
else:
  print("else block")
```

case 2: In loop when we use break statement the else block not executed.

```
for x in range(10):
  print("ratan sir",x)
  if(x==4):
    break
else:
  print("else block")
```

ex:

```
sum=0
for i in range(1,100):
        sum=sum+i
print sum
```

ex:

```
for i in range(1,10):
        if i%2==0:
                print("even number=",i)
        else:
```

```
                print("odd number",i)
```

ex:

```
words = ["cat", "apple", "ratanit","four"]
for w in words:
  print(w, len(w))

words = ["cat", "apple", "ratanit","four"]
for w in words[1:3]:
  print(w, len(w))
```

While loop:

```
   while <expression>:
        Body
```

ex :

```
a=0
while(a<10):
        print ("hi ratan sir")
        a=a+1
```

ex: ***else*** *is always executed after the for loop is over unless a break statement is encountered.*

```
a=0
while(a<10):
        print ("hi ratan sir",a)
        a=a+1
else:
  print("else block after while");
print("process done")
```

ex: *in below example else not executed.*

```
a=0
while(a<10):
        print ("hi ratan sir",a)
        a=a+1
        if(a==2):
          break
else:
  print("else block after while");
print("process done")
```

ex: *The below example represent infinite times.*

```
while(True):
        print ("hi ratan sir")
```

ex:

```
while True:
  eid = input("Enter emp id")
  ename=input("enter emp name")
  if(ename=="ratan"):
    print("login success")
    break
  else:
    print("login fail try with valid name")
```

Break & continue:

- ✓ *Break is used to stop the execution.*
- ✓ *Continue used to skip the particular iteration.*

ex:

```
for i in range(1,10):
  if(i==4):
    break
  print(i)
```

ex:

```
for i in range(1,10):
  if(i==4):
    continue
  print(i)
```

ex:

```
while 1:
   n = input("Please enter 'hello':")
   if n.strip() == 'hello':
      break
   else:
     print("u entered wrong input")
```

ex:

```
while True:
  n = input("enter some name")
  if(n=='exit'):
    break
  elif(len(n)<3):
    print("name is very small...")
  print("you entered good name....")
```

ex:

```
for letter in 'ratan':
   if letter == 'a' or letter == 'r':
      continue
```

```
    print ('Current Letter :', letter)
```

ex:

```
for letter in 'ratanit':
    if letter == 'a' or letter == 'x':
        continue
    elif letter=='i':
         break
    print ('Current Letter :', letter)
```

ex:

```
# Iterating over a List data
print("List Iteration")
l = ["ratan", "anu", "sunny"]
for i in l:
    print(i)

# Iterating over a tuple
print("Tuple Iteration")
t = ("aaa", "bbb", "ccc")
for i in t:
    print(i)

# Iterating over a String
print("String Iteration")
s = "ratanit"
for i in s :
    print(i)
# Iterating over dictionary
print("Dictionary Iteration")
d = dict()
d['xyz'] = 123
d['abc'] = 345
for i in d :
    print("%s  %d" %(i, d[i]))
```

ex:

```
result4 = [  ]
l=[10,20,30,40]
for x in l:
    if x>20:
        result4.append(x+1)
print(result4)
```

ex:

```
politicleParties=['tdp','ysrcp','congress','bjp']
electionYear=["2014","2019","2005","2001"]
countryStatus=["worst","developing","developed"]
corruptionStatus=["Max","Normal","Min"]

for party in politicleParties:
   year=input("enter a year")

   if year in electionYear:
      if year == "2014":
         print("tdp Wins!")
         print("Country status: "+countryStatus[2])
         print("Corruption Status: "+corruptionStatus[0])
         break

      elif year == "2019":
         print("ysrcp Won")
         print("Country status: "+countryStatus[0])
         print("Corruption Status: "+corruptionStatus[0])
         break

      elif year == "2005":
         print("congress won!")
         print("Country status: "+countryStatus[0])
         print("Corruption Status: "+corruptionStatus[0])
         break

      elif year == "2001":
         print("bjp won!")
         print("Country status: "+countryStatus[0])
         print("Corruption Status: "+corruptionStatus[0])
         break
```

```
else:
    print("Wrong year of election!")
```

Data types in Python

- *Numbers*
  - *Int / float/complex : type*
  - *Describes the numeric value & decimal value*
  - *These are immutable modifications are not allowed.*

- *Boolean*
  - *bool : type*
  - *represent True/False values.*
  - *0 =False & 1 =True*
  - *Logical operators and or not return value is Boolean*

- ✓ *Strings*
  - *str : type*
  - *Represent group of characters*
  - *Declared with in single or double or triple quotes*
  - *It is immutable modifications are not allowed.*

- ✓ *Lists*
  - ✓ *list : type*
  - ✓ *group of heterogeneous objects in sequence.*
  - ✓ *This is mutable modifications are allowed*
  - ✓ *Declared with in the square brackets **[ ]***

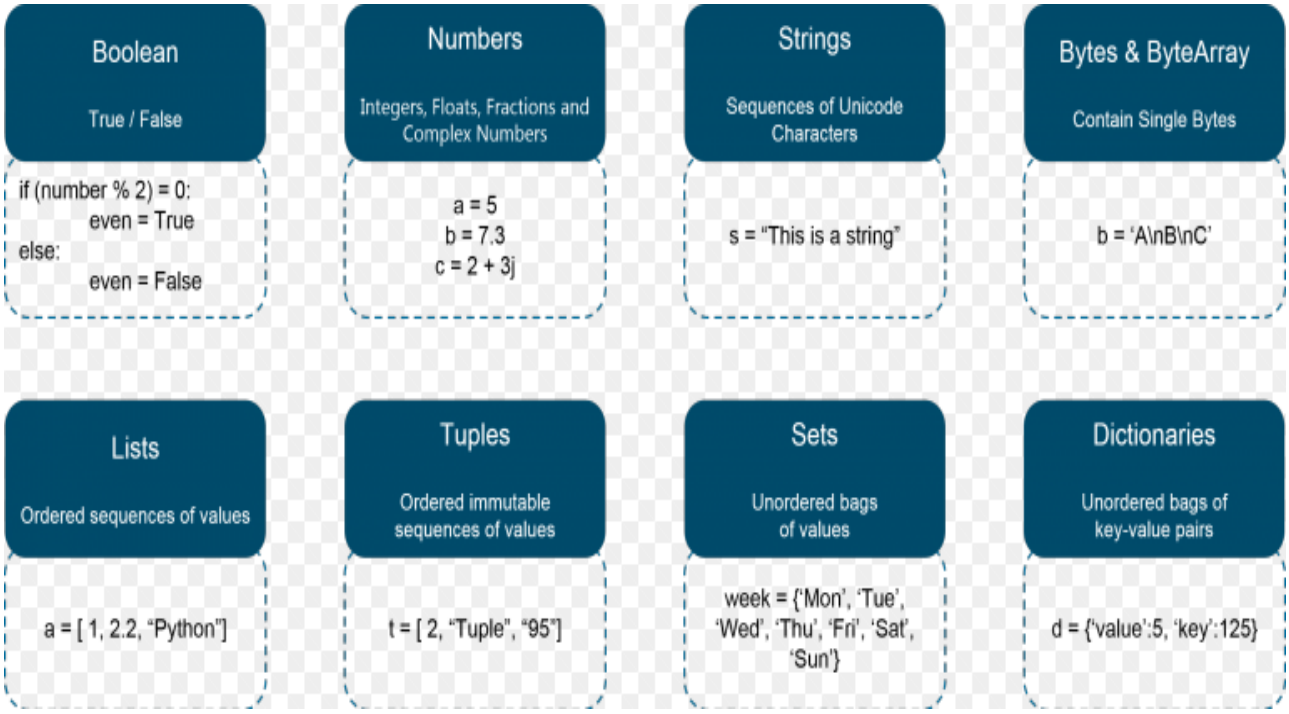
- ✓ *Tuples*
  - ✓ *tuple : type*
  - ✓ *group of heterogeneous objects in sequence*
  - ✓ *this is immutable modifications are not allowed.*
  - ✓ *Declared within the parenthesis ( )*

- ✓ *Sets*
  - ✓ *set : type*
  - ✓ *group of heterogeneous objects in unordered*
  - ✓ *this is mutable modifications are allowed*

- ✓ *declared within brasses { }*

- ✓ *Dictionaries*
  - ✓ *dict : type*
  - ✓ *it stores the data in key value pairs format.*
  - ✓ *Keys must be unique & value*
  - ✓ *It is mutable modifications are allowed.*
  - ✓ *Declared within the curly brasses {key:value}*

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	



Boolean data type :- (bool)

- ✓ *true& false are result values of comparison operation or logical operation in python.*
- ✓ *true & false in python is same as 1 & 0 1=true 0=false*
- ✓ *except zero it is always True.*
- ✓ *while writing true & false first letter should be capital otherwise error message will be generated.*
- ✓ *Comparison operations are return Boolean values.*

ex:

```
a = bool(1)
b = bool(0)
c = bool(10)
d = bool(-5)
e = int(True)
f = int(False)
print("a: ", a, " b: ", b, " c: ", c, " d: ", d , " e: ", e, " f: ", f)
```

ex:

```
T,F = True,Flase
print ("T: ", T, " F:", F)
print ("T and F: ", T and F) #False
print ("T and F: ", F and T) #False
print ("T and T: ", T and T) #True
print ("F and F: ", F and F) #False
print ("not T: ", not T) # False
print ("not F: ", not F) # True
print ("T or F: ", T or F) # True
print ("T or F: ", F or T) # True
print ("T or T: ", T or T) # True
print ("F or F: ", F or F) # False
```

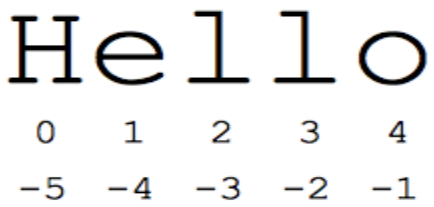
ex:

```
print (1 == 1) #true
print (5 > 3) #true
print (True or False) #true
print (3> 7) #false
print (True and False) #false
```

```
print ("ratan"=="ratan") #true
print ('a'=="a") #true
print (1==True) #true
print (False==0) #true
print (True+True) #2
print (False+False) #2
```

Strings Data type : (str)

- ✓ *A string is a list of characters in order enclosed by single quote or double quote.*
- ✓ *Python string is immutable modifications are not allowed once it is created.*
- ✓ *In java String data combine with int data it will become String but not in python.*
- ✓ *String index starts from 0, trying to access character out of index range will generate IndexError.*
- ✓ *In python it is not possible to add any two different data types, possible to add only same data type data.*

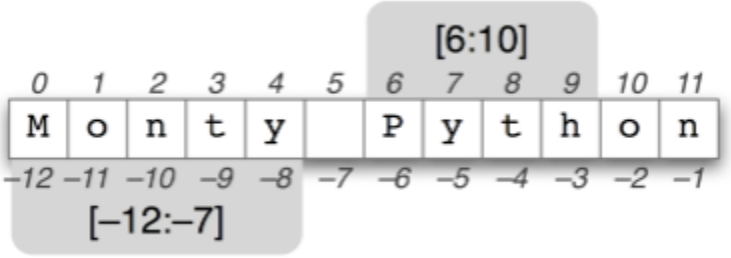


Slice Notation

<string_name>[startIndex:endIndex],
<string_name>[:endIndex],
<string_name>[startIndex:]

s[1:4] is 'ell' -- chars starting at index 1 and extending up to but not including index 4
s[1:] is 'ello' -- omitting either index defaults to the start or end of the string
s[:] is 'Hello' -- omitting both always gives us a copy of the whole thing
s[1:100] is 'ello' -- an index that is too big is truncated down to the string length
s[-1] is 'o' -- last char (1st from the end)

s[-4] is 'e' -- 4th from the end
s[:-3] is 'He' -- going up to but not including the last 3 chars.
s[-3:] is 'llo' -- starting with the 3rd char from the end and extending to the end of the string.



Example :

```
str="ratanit"
print(str[3])          #a
print(str[1:3])        #at
print(str[3:])         #anit
print(str[:4])         #ratan
print(str[:])          #ratanit

print(str[-3])             #n
print(str[-6:-4])          #at
print(str[-3:])            #nit
print(str[:-5])            #ra
print(str[:])              #ratanit
```

Method	***Functionality***
s.find(t)	*index of first instance of string t inside s (-1 if not found)*
s.rfind(t)	*index of last instance of string t inside s (-1 if not found)*
s.index(t)	*like s.find(t) except it raises ValueError if not found*
s.rindex(t)	*like s.rfind(t) except it raises ValueError if not found*
s.join(text)	*combine the words of the text into a string using s as the glue*
s.split(t)	*split s into a list wherever a t is found (whitespace by default)*
s.splitlines()	*split s into a list of strings, one per line*
s.lower()	*a lowercased version of the string s*
s.upper()	*an uppercased version of the string s*
s.title()	*a titlecased version of the string s*
s.strip()	*a copy of s without leading or trailing whitespace*
s.replace(t, u)	*replace instances of t with u inside s*

Example: upper() , lower() , capitalize() , replace() , split()

```
text = "hi ratan sir python is very good"
print ("upper =>", text.upper())
print ("lower =>", text.lower())
print ("capitalize =>", text.capitalize())
print ("split =>", text.split())
print ("join =>", "+".join(text.split()))
print ("replace =>", text.replace("python", "java"))
```

Example :- len() strip() enumerate() count() functions

- ✓ *The enumerate() function returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration.*
- ✓ *Similarly, len() returns the length (number of characters) of the string.*
- ✓ *Split () function used to split the data, the default splitting character is space.*

```
str1="  ratan  "
print(str1)
print(len(str1))
print(str1.strip())
print(len(str1.strip()))

str2="ho si.r how r u"
print (str2.split()) #default splitting character is space
print (str2.split("."))

str4 = 'ratanit'
# enumerate()
print(tuple(enumerate(str4)))
print(list(enumerate(str4)))

print ("ratanit".count('a'))
print ("ratanit".count('a',3,6))
print ("ratanit".count('a',5,7))

msg = "welcome to to ratanit"
substr1 = "to"
print (msg.count(substr1, 4, 16))
substr2 = "t"
print (msg.count(substr2))
```

String Formatting in Python :

String Formatting with the % Operator:

%d *int* ***%s*** *string* ***%f/%g*** *floating point*

```
x = "apples"
y = "lemons"
a=10
b=10.5
z = "In the basket are %s  %d  and %s %f " % (x,a,y,b)
print (z)
```

String Formatting with the { } Operators:

```
fname = "balu"
sal=10000.34
```

```
age = 24
print ("{} age :{}  salary is :{}".format(fname, age, sal))
print ("{0} age :{1}  salary is :{2}".format(fname, age, sal))
```

Example :

'10' + '20'	*"1020"*
"ratan"+"anu"	*"ratananu"*
"ratan"+"10"	*"ratan10"*
"ratan"+10	*TypeError: cannot concatenate 'str' and 'int' objects*

Replication Operator: (*)

- ✓ *Replication operator uses two parameter for operation. One is the integer value and the other one is the String.*
- ✓ *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.*

```
str1 = 'Hello'
str2 ='World!'
# using +
print(str1 + str2)
# using *
print(str1 * 3)
print(2 * str1 )
```

There are two types of Membership operators:

in: *"in" operator return true if a character or the entire substring is present in the specified string, otherwise false.*

not in: ***"****not in" operator return true if a character or entire substring does not exist in the specified string, otherwise false.*

```
str1="ratanit"
str2="durgasoft"
str3="ratan"
str4="durga"
print(str3 in str1)              # True
print(str4 in str2)              # True
print(str3 in str2)              # False
print("ratan"  in "ratanit")     #true
```

```
print("ratan"  in "durgasoft")  #False

print(str3 not in str1)          # False
print(str4 not in str2)          # False
print(str3 not in str2)          # True
print("ratan" not in "ratanit") # false
print("ratan" not in "anu")     # true
```

Example : startswith() & endswith() functions.

```
string1="Welcome to ratanit";
print (string1.endswith("ratanit"))
print (string1.endswith("to",2,16))
print (string1.startswith("Welcome"))
print (string1.startswith("come",3,10))
```

ex : index() & find() functions

```
str="Welcome to ratanit";
print (str.find("ratanit"))
print (str.index("come"))
print (str.index("t",10,15))
```

Example : swapcase() , lstrip() , rstrip() functions

```
string1="Welcome To Ratan It"
print (string1.swapcase())

string2="    Hello Python"
print (string2.lstrip())

string3="welcome to Ratanit@@@@@@@@@@"
print (string3.rstrip('@'))
```

ex : isalnum() , isalpha() , isdigit() functions

```
str="Welcome to ratanit"
print (str.isalnum()) #false

str1="Python36"
print (str1.isalnum()) #true

string2="HelloPython"
print (string2.isalpha()) #true

string3="This is Python3.1.6"
print (string3.isalpha()) #false

string4="HelloPython";
```

```
print (string4.isdigit()) #false

string5="98564738"
print (string5.isdigit()) #true
```

ex : islower() , isupper() , isspace() functions

```
string1="Ratanit";
print (string1.islower()) #false
string2="ratanit"
print (string2.islower()) #true
string3="ratanit";
print (string3.isupper()) #false
string4="RATANIT"
print (string4.isupper()) #true
string5="WELCOME TO WORLD OF PYT"
print (string5.isspace()) #false
string6=" ";
print (string6.isspace()) #true
```

Example: python program perform sorting of words in given String

```
mystr = input("Enter a string: ")
# breakdown the string into a list of words
words = mystr.split()
# sort the list
words.sort()
# display the sorted words
for word in words:
  print(word)
```

Relational Operators:

All the comparison operators i.e., (<,><=,>=, ==,! =, <>) are also applicable to strings. The Strings are compared based on dictionary Order.

```
print("ratan"=="ratan")
print("ratan">="Ratan")
print("Ratan"<="ratan")
print("Ratan"!="anu")
print("Ratan"!="Ratan")
```

ex :

```
speed=100
if speed >= 80:
        print ('You are so busted')
else:
```

```
        print ('Have a nice day')
```

ex :

Using for loop we can iterate through a string. Here is an example to count the number of 'a' in a string.

```
count=0
for i in "ratanit":
        if(i=="a"):
                count=count+1
print("a charcater occur:",count)
```

Errors in Strings :

NameError:

```
str = "hi sir"
print substring(2,4)
```

```
        G:\>python first.py
        NameError: name 'substring' is not define
```

IndexError:

```
str = "hi sir"
printstr[10]
```

```
        G:\>python first.py
        IndexError: string index out of range
```

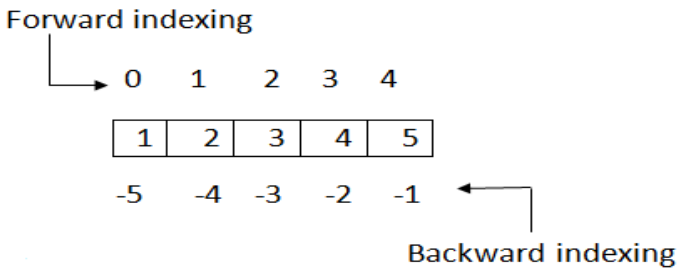
TypeError:

```
a=10;
str="ratan"
print a+str
```

```
        G:\>python first.py
        TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

List data type : (list)

- ✓ *List is used to store the group of values & we can manipulate them, in list the values are stores in index format starts with 0;*
- ✓ *List is mutable object so we can do the manipulations.*
- ✓ *A python list is enclosed between square([]) brackets.*
- ✓ *In list insertion order is preserved it means in which order we inserted element same order output is printed.*
- ✓ *A list can be composed by storing a sequence of different type of values separated by commas.*
  <list_name>=[value1,value2,value3,...,valuen];
- ✓ *The list contians forward indexing & backword indexing.*



ex : List data

```
data1=[1,2,3,4]                    # list of integers
data2=['x','y','z']                # list of String
data3=[12.5,11.6];                 # list of floats
data4=[]                           # empty list
data5=['ratan',10,56.4,'a']    # list with mixed datatypes

print (data1)
print (data2)
print (data3)
print (data4)
print (data5)
print(type(data1))
```

ex : Accessing List data

```
data1=[1,2,3,4];
data2=['ratan','anu','durga'];
print (data1[0])
print (data1[0:3])
print (data2[-3:-1])
print (data1[0:])
print (data2[:2])
print (data2[:])
```

ex:

```
x = list()
print(x)    # []

y = list("ratan")
print(y)   # ['r', 'a', 't', 'a', 'n']
print(len(y))

l1 = list("ratan")
print(l1)

l2 = [10,20,30]+[40,50]
print(l2)

l3 = [10,20,30]*3
print(l3)

print(len(l1))
print(len(l2))
print(len(l3))
```

ex:

```
marks= [60,70,80]
print(type(marks))
m1,m2,m3 = marks
print(m1, m2, m3)
print(type(m1),type(m2),type(m3))

a = [10,10.5,"ratan"]
x,y,z=a
print(x,y,z)
print(type(x),type(y),type(z))

i=[10,20,30]
a,b=i  #ValueError: too many values to unpack
```

Nested List :-

```
my_list = ["mouse", [1,2,3], ['a','b'],"anu"]
print(my_list)

a,b,c=my_list
print(a)
print(b)
```

```
print(c)
print(type(a),type(b),type(c))
# unpack the data

matrix = [[1,2,3],[4,5,6],[7,8,9]]
print(matrix[1])
print(matrix[1][1])
print(matrix[2][0])
```

ex:

- ✓ *The method **list.append(x)** will add an item (x) to the end of a list.*
- ✓ *The **list.insert(i,x)** method takes two arguments, where i = index x= item .*
- ✓ *we can use list.copy() to make a copy of the list.*
- ✓ *We can combine the list by using + operator.*
- ✓ *If we want to combine more than one list, we can use the **list.extend(L)** method.*

```
# adding the data by using append() function
animal = ['rat','tiger']
animal.append('cat')
print(animal)

#adding the data by using insert()
animal.insert(2,'lion')
animal.insert(0,'cow')
print(animal)

#adding the data by using index values
animal[1]="xxx"
animal[1:3]=["ratan","anu","live"]
print(animal)
animal[2:4]=[]
print(animal)

#adding the data by using + operator
l1=[1,2,3]
l2=[4,5,6]
l3 = l1+l2
print(l3)

#adding same data multiple times
x=[1,2,3]*3
print(x)
```

```
#copy the data by using copy method
fruit=["apple","orange","grapes","banana"]
fruitcopy = fruit.copy()
print(fruitcopy)

#adding the data by using extend()
a=[10,20,30]
b=[10,40,50]
a.extend(b)
print (a)
```

ex :

- ✓ *When we need to remove an item from a list, we'll use the **list.remove(x)** method which removes the first item in a list whose value is equivalent to x.*
- ✓ *If you pass an item in for x in list.remove() that does not exist in the list, you'll receive the following error: list.remove(x): x not in list*
- ✓ *We can use the **list.pop([i])** method to return the item at the given index position from the list and then remove that item.*
- ✓ *The square brackets around the i for index tell us that this parameter is optional, so if we don't specify an index (as in fish.pop()), the last item will be returned and removed.*
- ✓ *The del statement can also be used to remove slices from a list or clear the entire list*
- ✓ *By using clear() function we can clear the all the elements of the list.*

```
# removing elements by using remove()
a=[10,20,30]
b=[10,40,50]
a.remove(10)
print(a)
#a.remove('ratan')  #ValueError: list.remove(x): x not in list

#removing element by using pop() function
fruit=["apple","orange","grapes","banana","orange"]
print(fruit.pop())
print(fruit.pop(0))
#print(fruit.pop(10))  # IndexError: pop index out of range
print(fruit)

# deleting element by using del
a = [-1, 1, 66.25, 333, 333, 1234.5,"ratan"]
del a[0]
print(a)
del a[2:4]
print(a)
del a[:2]
```

```
print(a)
del a[1:]
print(a)
del a[:]
print(a)

#clearing element by using clear()  function
fruit=["apple","orange","apple","banana"]
print(fruit)
fruit.clear()
print(fruit)
```

ex:

- ✓ ***List.index()*** *used to prepare the index of element. If the eelement is not available generates error : ValueError: 'banana' is not in list.*
- ✓ *The reverse() function is used to reverse the order of items.*
- ✓ *We can use the list.sort() method to sort the items in a list.*
- ✓ *The list.count(x) method will return the number of times the value x occurs within a specified list.*

```
fruit=["apple","orange","grapes","orange"]
print(fruit.index("orange"))
print(fruit.index("orange",2))
#print(fruit.index("banana"))                #ValueError: 'banana' is not in list

fruit.reverse()
print(fruit)

fruit=["apple","orange","apple","banana"]
print(fruit.count("apple"))
fruit.sort()
print(fruit)
```

ex:

- ✓ ***==*** *operator will check the data in the list not memory address.*
- ✓ *To check the memory address use* ***is*** *&* ***is not*** *operator.*
- ✓ *To check the data is available or not use* ***in*** *&* ***not*** *in operators.*

```
l1=[10,20,30]
l2=[10,20,30]
l3=l1
print(id(l1))
print(id(l2))
print(id(l3))
print(l1==l2)
print(l1==l3)
```

```
print( l1 is l2)
print(l1 is not l2)
print(l1 is l3)
print(l1 is not l3)

print(10 in l1)
print(100 in l1)
print(10 not in l1)
print(100 not in l1)
```

ex: iterating list data by using for loop.

```
l1=[1,2,4]
for i in l1:
  print(i)
for i in [10,20,30]:
  print(i)
for i in list("ratan"):
  print(i)
fruit=["apple","orange","graps","banana"]
for i in fruit[1:3]:
  print(i,len(i))
```

ex:

```
x = [i for i in range(10,20)]
print(x)
y = [i*i for i in range(10)]
print(y)
z= [i+1 for i in range(10,20,2)]
print(z)
```

ex:

```
line = "hi ratan sir how are you"
words =  line.split()
print(words)
print(type(words))
x = [[w.upper(),w.lower(),len(w)] for w in words]
print(x)
y = ['AnB','ABN','and','bDE']
print(sorted([i.lower() for i in y]))
print(sorted([i.lower() for i in y],reverse=True))
```

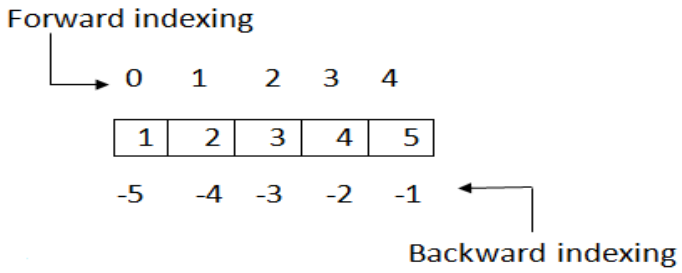
Ex:

```
def append_to_sequence (myseq):
```

```
        myseq += (9,9,9)
        return myseq

    tuple1 = (1,2,3) # tuples are immutable
    list1 = [1,2,3] # lists are mutable
    tuple2 = append_to_sequence(tuple1)
    list2 = append_to_sequence(list1)
    print ('tuple1 = ', tuple1) # outputs (1, 2, 3)
    print ('tuple2 = ', tuple2) # outputs (1, 2, 3, 9, 9, 9)
    print ('list1 = ', list1) # outputs [1, 2, 3, 9, 9, 9]
    print ('list2 = ', list2) # outputs [1, 2, 3, 9, 9, 9]
ex:
l1=[['a',2],['b',4],['c',6]]

l2=list()
l3=list()

for i,j in l1:
   l2.append(i)
   l3.append(j)

print(l2)
print(l3)
```

Tuple data type : (tuple)

- ✓ *tuple : type*
- ✓ *group of heterogeneous objects in sequence*
- ✓ *this is immutable modifications are not allowed.*
- ✓ *Declared within the parenthesis ( )*
- ✓ *Insertion order is preserved it means in which order we inserted the objects same order output is printed.*
- ✓ *The tuple contains forward indexing & backward indexing.*



ex:

```
tup1 = ('ratan', 'anu', 'durga')
tup2 = (1, 2, 3, 4, 5 )
tup3 = "a", "b", "c", "d"      #  valid not recommended
tup4=()
tup5 = (10)
tup6 = (10,)
tup7=(1,2,3,"ratan",10.5)

print(tup1)
print(tup2)
print(tup3)
print(tup4)
print(type(tup5))   #<class 'int'>
print(type(tup6))   #<class 'tuple'>
print(tup7)
```

Syntactically, a tuple is a comma-separated list of values:

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

Although it is not necessary, it is common to enclose tuples in parentheses to help us quickly identify tuples when we look at Python code:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

Ex:

```
t1=(10,20,30,40,50)
print(t1[0])
print(t1[2:4])
print(t1[:4])
print(t1[2:])
print(t1[:])
print(t1[2:10])
#print(t1[10])  #IndexError: tuple index out of range

print(t1[-3])
print(t1[-4:-2])
print(t1[:-3])
print(t1[-3:])
print(t1[:])
#print(t1[-9]) #IndexError: tuple index out of range
```

ex: un packing tuple data

```
t1 = tuple()
print(t1)

t2 = tuple("ratan")
print(t2)

t3 =(10,1.5,'ratan')
a,b,c=t3
print(a,b,c)
print(type(a),type(b),type(c))

t4=(10,20)
a,b,c=t4   #ValueError: not enough values to unpack (expected 3, got 2)
print(a,b,c)

m = ( 'have', 'fun' )
x = m[0]
y = m[1]
print(x,y)
```

ex:

```
t1=(10,20,30)
t2=(10,20,30)
t3=t1

print(id(t1))
print(id(t2))
print(id(t3))
```

 # checking the data only but not address

```
print(t1==t2)
print(t1==t3)
print(t3==t2)

# checking the memory address
print(t1 is t2)
print(t1 is not t2)
print(t1 is t3)
print(t1 is not t3)

# checking data is available or not
print(10 in t1)
print(100 in t2)
print(100 not in t2)
```

ex: ***Nested Tuple***

```
my_tuple = ("mouse", (1,2,3), ('a','b'),"anu")
print(my_tuple)

matrix = ((1,2,3),(4,5,6),(7,8,9))
print(matrix[1])
print(matrix[1][1])
print(matrix[2][0])
```

ex:

```
tup1 = (10, 20,30)
tup2 = ('ratan', 'anu')
tup3 = tup1 + tup2
print(tup3)
print(len(tup3))

tup4 = tup1*3
print(tup4)
print(len(tup4))

# tuple is immutable so modifications are not allowed
t1=(10,20,30)
#t1[1]=100          TypeError: 'tuple' object does not support item assignment
#t1.insert(1,100)   AttributeError: 'tuple' object has no attribute 'insert'
print(t1)

#conversion of tuple to list data do the modifications
t1=(10,20,30)
t2 = t1+(40,50)
print(t2)

l1 = list(t2)
l1.append(60)
```

```
l1.insert(2,6)
t3 = tuple(l1)
print(t3)

#conversion of list to tuple
l= [10,20,30]
t= tuple(l)
print(t)

# conversion of tuple to String
tup = ('r','a','t','a','n')
str =  ''.join(tup)
print(str)
```

ex:

```
from copy import deepcopy
#create a tuple
tuplex = ("HELLO", 5, [], True)
print(tuplex)
#make a copy of a tuple using deepcopy() function
tuplex_clone = deepcopy(tuplex)
tuplex_clone[2].append(50)
print(tuplex_clone)
print(tuplex)
```

ex:

```
tuplex = (2, 4, 5, 6, 2, 3, 4, 4, 7)
print(tuplex)
count = tuplex.count(4)
print(count)

tuplex = tuple("ratan")
print(tuplex)

index = tuplex.index("r")
print(index)

#define the index from which you want to search
index = tuplex.index("a", 3)
print(index)
```

```
#define the segment of the tuple to be searched
index = tuplex.index("t", 1,3 )
print(index)
#if item not exists in the tuple return ValueError Exception
#index = tuplex.index("y")

t1=(10,20,30)
print(min(t1))
print(max(t1))

t1=('ratan','anu','durga')
print(min(t1))
print(max(t1))
```

Ex:

```
txt = “hi rattan sir what about python”
words = txt.split()
t = list()
for word in words:
  t.append((len(word), word))

t.sort(reverse=True)   # decending order
#t.sort()  ascending order

print(t)
res = list()
for length, word in t:
   res.append(word)
print (res)

res = list()
for length, word in t:
   res.append(length)

print (res)
```

Python set:
Python also includes a data type for sets. A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the set() function can be used to create sets. Note: to create an empty set you have to use set(), not {}; the latter creates an empty dictionary, a data structure that we discuss in the next section.

Ex:

```
# set of integers
my_set = {1, 2, 3}
print(my_set)
print(type(my_set))

#creates empty set
s=set()
print(s)

# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)

# set do not have duplicates
# Output: {1, 2, 3, 4}
my_set = {1,2,3,4,3,2}
print(my_set)
```

ex:

```
# set of integers
my_set = {1, 2, 3}
my_set.add(9)
print(my_set)

#add list data
my_set.update([6,7,8])
print(my_set)

# add list and set
my_set.update([4,5], {1,6,8})
print(my_set)
```

Ex:

*We can test if an item exists in a set or not, using the keyword **in , not in***

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket)    # show that duplicates have been
print('orange' in basket)
print('orange' not in basket)
print('mango' in basket)
print('mango' not in basket)
```

ex:

```
a= set("ratan")
print(a)

b= set("ratansoft")
print(b)

print(b-a)     # letters in b but not in a
print(a|b)     # letters in a or b or both
print(a&b)     # letters in both a and b
print(a^b)     # letters in a or b but not both

basket1 = {'orange', 'apple', 'pear','banana'}
basket2 = {'pear', 'orange', 'banana'}
print(basket1-basket2)
print(basket1|basket2)
print(basket1&basket2)
print(basket1^basket2)
```

ex:

```
engineers = set(['ratan', 'anu', 'durga'])
programmers = set(['Jack', 'Sam', 'Susan'])
managers = set(['Jane', 'Jack', 'Susan'])

employees = engineers | programmers | managers
print(employees)
```

ex: creating set by passing more than one list data.

```
l1 = ['ratan', 'anu', 'durga']
l2 = ['Jack', 'Sam', 'Susan']
l3 = ['Jane', 'Jack', 'Susan']

engineers = set(l1+l2+l3)
print(engineers)
```

Ex:

A particular item can be removed from set using methods, discard() and remove().
we can remove and return an item using the pop() method.
Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary.
We can also remove all items from a set using clear().

```
my_set = {1, 2, 3, 4, 4, 5, 5, 6}
print(my_set)
print(len(my_set))

#copy the set data
a= my_set.copy()
print(a)

my_set.discard(4)
print(my_set)

my_set.remove(5)
print(my_set)

my_set.discard(7)
print(my_set)

#pop takes random element
print(my_set.pop())
print(my_set.pop())

print(my_set)
```

```
# my_set.remove(7)
# print(my_set)     # KeyError: 7

my_set.clear()
print(my_set)
```

ex: iterating by using for loop

```
for letter in {"apple","banana"}:
    print(letter)

for letter in set("apple"):
    print(letter)
```

Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned. While tuples are immutable lists, frozensets are immutable sets.
Frozensets can be created using the function frozenset().

This datatype supports methods like copy(), difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric_difference() and union(). Being immutable it does not have method that add or remove elements.

Dictionary data type

- ✓ *List,tupple ,set data types are used to represent individual objects as a single entity.*
- ✓ *To store the group of objects as a key-value pairs use dictionary.*
- ✓ *A dictionary is a data type similar to arrays, but works with keys and values instead of indexes.*
- ✓ *Each value stored in a dictionary can be accessed using a key, which is any type of object (a string, a number, a list, etc.) instead of using its index to address it.*
- ✓ *The keys must be unique keys but values can be duplicated.*

Ex:

```
phonebook = {}
phonebook["ratan"] = 935577566
phonebook["anu"] = 936677884
phonebook["durga"] = 9476655551
print(phonebook)
```

ex: Alternatively, a dictionary can *be initialized with the same values in the following notation:*

```
phonebook = { "ratan" : 935577566, "anu" : 936677884, "durga" : 9476655551}
print(phonebook)
```

ex :

```
phonebook = { "ratan" : 938477566,"anu" : 938377264,"durga" : 947662781}
del phonebook["durga"]
phonebook.pop("anu")
print(phonebook)
```

- ✓ Dictionaries can be created using pair of curly braces ({}). Each item in the dictionary consist of key, followed by a colon, which is followed by value. And each item is separated using commas (,) .
- ✓ An item has a key and the corresponding value expressed as a pair, key: value.
- ✓ in dictionary values can be of any data type and can repeat,.
- ✓ keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

ex:

```
# empty dictionary
my_dict = {}
print(my_dict)

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}
print(my_dict)

# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}
print(my_dict)
```

```
friends = {'tom':'111-222-333','jerry':'666-33-111'}
print(friends)
print(friends['tom'])
print(friends.get('jerry'))
print(friends.get(1))  #none
print(friends['ratan'])  #KeyError: 'ratan'
```

While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the get() method.

ex: Dictionary are mutable. We can add new items or change the value of existing items using assignment operator.

Adding & deleting

```
# empty dictionary
my_dict = {}
print(my_dict)

#adding item
my_dict['ratan']=1234
print(my_dict)

#updating value
my_dict['ratan']=6666
print(my_dict)

#deleting keys
del my_dict['ratan']

print(my_dict)
```

ex:

```
d1={1:"ratan",2:"anu",4:"durga",3:"aaa"}

print(d1.keys())
print(d1.values())

print(list(d1.keys()))
print(list(d1.values()))

print(tuple(d1.keys()))
print(tuple(d1.values()))

print(set(d1.keys()))
```

```
print(set(d1.values()))

print(sorted(d1.keys()))
print(sorted(d1.values()))
ex:
d1={1:"ratan",2:"anu",4:"durga",3:"aaa"}

print(sorted(d1.keys()))
print(sorted(d1.values()))

squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for i in squares:
    print(i)

for i in squares:
    print(squares[i])

ex:
in or not in operators
in  and not in  operators to check whether key exists in the dictionary.

my_disc={1:"ratan",2:"anu"}

#in   not in
print(1 in my_disc)
print(2 not in my_disc)


ex:
d1={1:"ratan",2:"anu"}
d2={111:"ratan",222:"anu"}
d3=d1

print(id(d1))
print(id(d2))
print(id(d3))

print(d1 is d2)
print(d1 is d3)
print(d1 is not d2)
print(d1 is not d3)

print(d1==d2)
print(d1==d3)
```

```
print(d1!=d2)
print(d2!=d3)

print(1 in d1)
print(11 in d1)
print(1 not in d1)
print(11 not in d1)

ex:
l1=[10,20,30,40]
l2=["ratan","anu","durga","aaa"]
x = zip(l1,l2)
d = dict(x)
print(d)

l1=(10,20,30,40)
l2=("ratan","anu","durga","aaa")
x = zip(l1,l2)
d = dict(x)
print(d)

l1={10,20,30,40}
l2={"ratan","anu","durga","aaa"}
x = zip(l1,l2)
d = dict(x)
print(d)

ex:
d1={10:"ratan",20:"anu"}
d2={1:"aaa",2:"bbb"}

l1 = list(d1.keys())
l2 = list(d2.values())

x = zip(l1,l2)
d = dict(x)
print(d)


ex:
Equality Tests in dictionary
==  and !=  operators tells whether dictionary contains same items not.

d1={1:"ratan",2:"anu"}
```

```
d2={111:"ratan",222:"anu"}
d3=d1

print(id(d1))
print(id(d2))
print(id(d3))

print(d1==d2)
print(d1==d3)
print(d1!=d2)
print(d2!=d3)




d1={1:"ratan",2:"anu",3:"durga",4:"aaa"}
d2 = d1.copy()
d3=d1;
print(d2)
print(d3)

print(list(d1.keys()))
print(tuple(d1.keys()))
print(list(d1.values()))
print(tuple(d1.values()))

print(d1.items())        #we could turn this into a list with two-tuples



ex:
hits = {"home": 125, "sitemap": 27, "about": 43}
keys = hits.keys()
values = hits.values()
print(len(hits))

print("Keys:")
print(keys)
print(len(keys))

print("Values:")
print(values)
print(len(values))
```

ex:
popitem() Returns randomly select item from dictionary and also remove the selected item.

clear() Delete everything from dictionary

keys() Return keys in dictionary as tuples

values() Return values in dictionary as tuples

get(key) Return value of key, if key is not found it returns None, instead on throwing KeyError exception

pop(key) Remove the item from the dictionary, if key is not found KeyError will be thrown

```
d1={1:"ratan",2:"anu","durga":3,4:"aaa"}
print(d1.popitem())

print(d1.keys())
print(d1.values())
print(d1.get(1))
print(d1.pop(2))

print(d1)
d1.clear()
print(d1)
```

update() merges the keys and values of one dictionary into another, overwriting values of the same key:

```
d1={1:"ratan",2:"anu",3:"durga"}
d2={111:"ratan",222:"anu",3:"xxx"}
d1.update(d2)
print(d1)
```

Dictionaries from Lists

```
l1=[1,2,3]
l2=["apl1e","orange","grapes",]

d=zip(l1,l2)
d1 = dict(d)
print(d1)


# Create a dictionary.
data = {"a": 1, "b": 2, "c": 3}
# Loop over items and unpack each item.
for k, v in data.items():
    # Display key and value.
    print(k, v)




original = {"box": 1, "cat": 2, "apple": 5}

# Create copy of dictionary.
modified = original.copy()

# Change copy only.
modified["cat"] = 200
modified["apple"] = 9

# Original is still the same.
print(original)
print(modified)



d1 = {1:"ratan",3:"anu"}
d2 = {2:"aaa",4:"bbb"}
x = {**d1,**d2}
print(x)



# A list of keys.
keys = ["bird", "plant", "fish"]
```

```
# Create dictionary from keys.
d = dict.fromkeys(keys, 5)

# Display.
print(d)


ex:
pairs = [("cat", "meow"), ("dog", "bark"), ("bird", "chirp")]

# Convert list to dictionary.
lookup = dict(pairs)
print(lookup)

print(lookup.items())
```

ex:

```
v1 = int(2.7) # 2
v2 = int(-3.9) # -3
v3 = int("2") # 2
v4 = int("11", 16) # 17, base 16
v5 = long(2)
v6 = float(2) # 2.0
v7 = float("2.7") # 2.7
v8 = float("2.7E-2") # 0.027
v9 = float(False) # 0.0
vA = float(True) # 1.0
vB = str(4.5) # "4.5"
vC = str([1, 3, 5]) # "[1, 3, 5]"
vD = bool(0) # False; bool fn since Python 2.2.1
vE = bool(3) # True
vF = bool([]) # False - empty list
vG = bool([False]) # True - non-empty list
vH = bool({}) # False - empty dict; same for empty tuple
vI = bool("") # False - empty string
vJ = bool(" ") # True - non-empty string
vK = bool(None) # False
vL = bool(len) # True
```

Python operators

- ✓ *Python Arithmetic Operators.*
- ✓ *Python Comparison/relational Operators*
- ✓ *Python Logical Operators*
- ✓ *Python Assignment Operators*
- ✓ *Python Identity Operators*
- ✓ *Python Membership Operators*
- ✓ *Python Bitwise Operators*

✓ ***Arithemtic operators***

Arithmetic Operators

Operator	Meaning	Example
+	Addition	4 + 7 ⟶ 11
-	Subtraction	12 - 5 ⟶ 7
*	Multiplication	6 * 6 ⟶ 36
/	Division	30 / 5 ⟶ 6
%	Modulus	10 % 4 ⟶ 2
//	Quotient	18 // 5 ⟶ 3
**	Exponent	3 ** 5 ⟶ 243

ex : *x ,y= 15,4*

```
# Output: x + y = 19
print('x + y =',x+y)

# Output: x - y = 11
print('x - y =',x-y)

# Output: x * y = 60
print('x * y =',x*y)

# Output: x / y = 3.75
print('x / y =',x/y)

# Output: x // y = 3
print('x // y =',x//y)

# Output: x ** y = 50625
print('x ** y =',x**y)
```

Python Mixed-Mode Arithmetic

The calculation which done both integer and floating-point number is called mixed-mode arithmetic. When each operand is of a different type.

9/2.0 ---> 4.5

✓ ***Relational operators***

Relational Operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

```
x = 5
y = 2

# Output: x > y is False
print('x > y  is',x>y)

# Output: x < y is True
print('x < y  is',x<y)

# Output: x == y is False
print('x == y is',x==y)

# Output: x != y is True
print('x != y is',x!=y)

# Output: x >= y is False
print('x >= y is',x>=y)

# Output: x <= y is True
print('x <= y is',x<=y)
```

- ✓ ***Logical operator :***

Logical operators

```
>>> a, b, c = 10, 20, 30

>>> (a > b) and (b < c)
False

>>> (a < b) and (b < c)
True

>>> (a > b) or (b < c)
True
```

Operator	Description
a **and** b	Logical AND If both operands are True than it returns True
a **or** b	Logical OR If one of the operands is True then it returns True
not	Logical NOT

Example :

```
x = True
y = False

# Output: x and y is False
print('x and y is',x and y)

# Output: x or y is True
print('x or y is',x or y)

# Output: not x is False
print('not x is',not x)
```

✓ ***Assignment operator :***

Operator	Description
=	x=y , y is assigned to x
+=	x+=y is equivalent to x=x+y
-=	x-=y is equivalent to x=x-y
*=	x*=y is equivalent to x=x*y
/=	x/=y is equivalent to x=x/y
**=	x**=y is equivalent to x=x**y

Assignment operators in Python

Operator	Example	Equivatent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

✓ ***Identity operators***

Operator	Meaning
is	True if the operands are identical (refer to the same object)
is not	True if the operands are not identical (do not refer to the same object)

x1 , y1 = 5 ,5
x2 , y2 = "ratan",”ratan”
x3 , y3= [1,2,3],[1,2,3]

print(id(x1)) ***#20935504***
print(id(y1)) ***#20935504***
print(x1 is y1)
print(x1 is not y1)

print(id(x2)) ***#21527296***
print(id(y2)) ***#21527296***
print(x2 is y2)
print(x2 is not y2)

print(id(x3)) ***#45082264***
print(id(y3)) ***#45061624***
print(x3 is y3)
print(x3 is not y3)

ex:
a=10
b=15

x=a
y=b
z=a

print(x is y)
print(x is a)
print(y is b)
print(x is not y)
print(x is not a)
print(x is z)

- ✓ ***Membership operators:***

  in: *"in" operator return true if a character or the entire substring is present in the specified string, otherwise false.*

  not in: ***"****not in" operator return true if a character or entire substring does not exist in the specified string, otherwise false.*

```
str1="ratanit"
str2="durgasoft"
str3="ratan"
str4="durga"
print(str3 in str1)                 # True
print(str4 in str2)                 # True
print(str3 in str2)                 # False
print("ratan"  in "ratanit")      #true
print("ratan"  in "durgasoft")  #False

print(str3 not in str1)             # False
print(str4 not in str2)             # False
print(str3 not in str2)             # True
print("ratan" not in "ratanit") # false
print("ratan" not in "anu")      # true
```

✓ ***Bitwise operator: -***

Operator	Description
|	Perform binary OR operation
&	Perform binary AND operation
~	Perform binary XOR operation
^	Perform binary one's Complement operation
<<	Left shift operator, left side operand bit is moved left by numeric number specified in right side
>>	Right shift operator, left side operand bit is moved right by numeric number specified in right side

& operator :

print(3&7)

0011
0111
0011

print(15&15)

1111
1111
1111

print(9&6)

1001
0101
0000

print(0&0)

0000
0000
0000

| operator :

print(3|7)

0011
0111
0111

print(15|15)

1111
1111
1111

print(9|6)

1001
0101
1111

print(0|0)

0000
0000
0000

Python Class concept

- ✓ *Class is a logical entity grouping functions and data as a single unit where as object is the physical entity represent memory.*
- ✓ *Class is a blue print it decides object creation.*
- ✓ *Based on single class possible to create multiple objects but every object occupies memory.*
- ✓ *In python define the class by using class keyword.*

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

Classic class declaration not inheriting form any class. Python 2.7

```
class MyClass:                    # Don't inherit from anything.
```

While new-style classes inherit from either object or some other class. Python 3.x

The three declarations are valid & same:

```
class MyClass(object):          # Inherit from object, new-style class.
class MyClass:
class MyClass():
```

Creating of object of a class:

```
c = myClass()
```

- ✓ *The default super class in python is object in python3.*
- ✓ *The class cantinas group of Functions , constructors, variables.*
- ✓ *The self-argument represent the method is belongs to particular class.*
- ✓ ***Self*** *is the name preferred by convention by Pythons to indicate the first parameter of instance methods in Python.*
- ✓ ***Self*** *It is part of the Python syntax to access members of objects.*
- ✓ *To declare the empty class without functions use* ***pass*** *statement.*

```
class AClass:
  pass
```

*Ex**: In python 2.7** by default our class not extending object class*

```
class MyClass:
  pass
print(type(MyClass))
print(issubclass(MyClass,object))

output :
<type 'classobj'>
False
```

python 3.x *: the default super class is object*

```
#three declarations are valid & same
class MyClass:
    pass

class MyClass():
    pass

class MyClass(object):
    pass
```

ex:

```
class MyClass:
  pass
print(type(MyClass))
print(issubclass(MyClass,object))

class MyClass():
  pass
print(type(MyClass))
print(issubclass(MyClass,object))

class MyClass(object):
  pass
print(type(MyClass))
print(issubclass(MyClass,object))

output :
<class 'type'>
True
<class 'type'>
True
<class 'type'>
True
```

ex :

```
class Myclass():
    def m1(self):
      print("m1 function")
    def m2(self):
      print("m2 function")

c = Myclass()
c.m1()
c.m2()
```

ex: *For the single class possible to create multiple objects every object occupies memory.*

```
class MyClass:
        def add(self,a,b):
                print(a+b)
        def mul(self,x,y):
                print(x*y)

c1 = MyClass()
c1.add(10,20)
c1.mul(3,4)

c2 = MyClass()
c2.add(100,200)
c2.mul(4,5)
```

ex:

```
class MyClass:
        a,b=10,20
        def m1(self):
                print(self.a)
                print(self.b)
        def m2(self):
                print(self.a)
                print(self.b)

c = MyClass()
c.m1()
c.m2()
#printing class variables outside of the class
print(c.a)
print(c.b)
```

ex:

```
class MyClass():
    a,b = 10,20
#first object creation
c1 = MyClass()
print(c1.a+c1.b)
#second object creation
c2 = MyClass()
print(c2.a+c2.b)
```

ex:

```
class MyClass:
    a=10

c1 = MyClass()
c2 = MyClass()
c2.a=100
print(c1.a)
print(c2.a)
```

ex:

```
class MyClass:
    a,b=10,20
    def add(self,a,b):
        print(a+b)
        print(self.a+self.b)
    def mul(self,a,b):
        print(a*b)
        print(self.a*self.b)

c = MyClass()
c.add(100,200)
c.mul(3,4)
```

ex: Named object vs. Name-less object

```
class MyClass():
    def show(self):
        print("ratan world")
# named object
c1 = MyClass()
c1.m1()

# name less object
MyClass().m1()
```

Object ID:

- ✓ *Every object has an identity, a type and a value. An object's identity never changes once it has been created, you may think of it as the object's address in memory.*
- ✓ *To get the id of the object use id() function it returns an integer representing its identity.*
- ✓ *The 'is' , 'is not' operators compares the identity of two objects.*

```
class MyClass():
        def show(self):
                print("ratan world")
c1 = MyClass()
c2 = c1
c3 = MyClass()

print(id(c1))
print(id(c2))
print(id(c3))

print(c1 is c2)         #True
print(c1 is c3)         #False
print(c1 is not c2)     #Flase
print(c1 is not c3)     #True
```

Ex:

```
class Myclass:
  def show(self):
    print("ratanit")

# class object
c = Myclass()
print(c)
print(c.show)

#function object
print(Myclass.show)

output :
<Myclass object at 0x7feb2a4d13c8>
<bound method Myclass.show of <Myclass object at 0x7feb2a4d13c8>>
<function Myclass.show at 0x7feb2a1ec400>
```

Constructor: ***__init__ ()***

- ✓ *Constructors are used to write the logics these logics are executed during object creation.*
- ✓ *Constructors are used to initialize the values to variables during object creation.*
- ✓ *The constructor arguments are local variables.*
- ✓ *To make the local variables to global variables use self keyword. (self.eid=eid)*

ex:

```
class MyClass:
  def __init__(self):
    print("constructor")

a = MyClass()
a.show()
```

ex:

```
class MyClass:
        a,b=100,200
        def __init__(self,a,b):
                print(a+b)
                print(self.a+self.b)

c = MyClass(10,20)
```

ex :

```
class Student:
   def __init__(self, rollno, name):
     self.rollno = rollno
     self.name = name

   def displayStudent(self):
            print ("rollno : ",self.rollno )
            print("name: ",self.name)

student1 = Student(111, "ratan")
student1.displayStudent()

student2 = Student(222, "anu")
student2.displayStudent()
```

ex: ***__init__()*** *executed when we create the object.*
__str__() *executed when we print reference variable it returns always String only.*

```
class MyClass:
    pass
c = MyClass()
print(c)

class Test:
    def __str__(self):
        return "ratanit.com"
t = Test();
print(t)

output :    <MyClass object at 0x7f89d1115438>
            ratanit.com
```

ex:

```
class Emp:
    def __init__(self,eid,ename,esal):
        self.eid=eid
        self.ename=ename
        self.esal=esal

    def disp(self):
        print("emp id=",self.eid)
        print("emp name=",self.ename)
        print("emp esal=",self.esal)

e1 = Emp(111,"ratan",10000)
e1.disp();
Emp(222,"anu",20000).disp()
```

ex:

```
class Emp:
    def __init__(self,eid,ename,esal):
        self.eid=eid
        self.ename=ename
        self.esal=esal
    def __str__(self):
        return "emp id=%d Emp name=%s Emp sal=%g"%(self.eid,self.ename,self.esal)

e1 = Emp(111,"ratan",100000.45)
print(e1)
e2 = Emp(111,"anu",200000.46)
print(e2)
```

ex :
```
class Greeting:
        msg = 'morning'
        def __init__(self,name):
            self.name=name;
r = Greeting('ratan')
a = Greeting('anu')

print( r.msg , r.name)
print (a.msg , a.name)
```

ex :
```
class Pet(object):
   def __init__(self, name, species):
      self.name = name
      self.species = species

   def getName(self):
      return self.name

   def getSpecies(self):
      return self.species

a = Pet("durga", "human")
name =a.getName()
print("Name=",name)

print("species=",a.getSpecies())
```

ex:
```
class Emp:
  def __init__(self,eid,ename):
    self.eid=eid
    self.ename=ename

  def getEid(self):
    return self.eid

  def getEname(self):
    return self.ename;

e1 = Emp(111,"ratan")
print(e1.getEid(),e1.getEname())

e2 = Emp(222,"anu")
print(e2.getEid(),e2.getEname())
```

ex :

```
class Employee:
  #Common variable for all employees
  empCount = 0

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

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

  def __str__(self):
    return "Name : {0}  Salary: {1}".format(self.name,self.salary)

emp1 = Employee("ratan", 20000)
print(emp1)
emp2 = Employee("anu", 5000)
print(emp2)

emp2.displayCount()
```

Destructors in python:

- ✓ *To destroy the object use* ***del***
- ✓ *Destructors are called when an object gets destroyed, it is opposite to constructors.*
- ✓ *When we destroy the object the* ***__del__()*** *function executed.*
- ✓ *When multiple reference variables are pointing to same object , if all reference variable count will become zero then only __del__() will be executed.*

ex:

```
class Vachile:
        def __init__(self):
                print("Vachile object created")
        def __del__(self):
                print("Vachile object destroyed")
v = Vachile()
del v
```

ex:

```
class MyClass:
  def __del__(self):
    print("object destroyed")

c1 = MyClass()
c2= MyClass()

print(id(c1))
print(id(c2))

del c1
del c2
```

ex:

- ✓ *When multiple reference variables are pointing to same object, if all reference variable count will become zero then only __del__() will be executed.*

```
class MyClass:
  def __del__(self):
    print("object destroyed")

c1 = MyClass()
c2= c1
c3 = c1

del c1
del c2
del c3
```

ex: *If any exceptions are raised in __del__() these are ignored object destroyed*

```
class Vachile:
        def __del__(self):
                print("Vachile object destroyed")
                print(10/0)
v = Vachile()
del v
```

```
E:\>python first.py
Vachile object destroyed
Exception ZeroDivisionError: 'integer division or modulo by zero' in <bound meth
od Vachile.__del__ of <__main__.Vachile instance at 0x02901260>> ignored
```

ex:

```
class MyClass:
        a=10
        def m1(self):
                print("m1 function")
        def __init__(self):
                print("constructor")
        def __str__(self):
                return "ratanit.com"
        def __del__(self):
                print("object destoryed....")

c = MyClass()           # constructor  __init__()  executed
print(c.a)              # variables are printed
c.m1();                 # method executed
print(c)                #  __str__()  executed
del c                   #  __del__()  executed
```

ex:

```
class Customer:
  def __init__(self,name,bal=0.0):
    self.name=name
    self.bal=bal

  def deposit(self,amount):
    self.bal=self.bal+amount

  def withdraw(self,amount):
    if amount>self.bal:
      raise RuntimeError("withdraw  amount is more than balance")
    else:
     self.bal=self.bal-amount

  def remaining(self):
    return self.bal;

c = Customer("ratan",10000)
damt = int(input("enter amount to deposit"))
c.deposit(damt)

amt = int(input("enter amount to withdraw"))
c.withdraw(amt)

print(c.remaining())
```

Class Attribute:

Hasattr() : *To check the attribute present in the class or not. If the attribute present in the class it returns true otherwise false.*

```
class Student:
  def __init__(self, rollno, name):
    self.rollno = rollno
    self.name = name

Student1 = Student(111, "ratan")
Student2 = Student(222, "anu")
print(hasattr(Student1,"rollno"))
print(hasattr(Student1,"name"))
print(hasattr(Student1,"age"))
```

Ex:

```
class Student:
  def __init__(self, rollno, name):
    self.rollno = rollno
    self.name = name

Student1 = Student(111, "ratan")

print(hasattr(Student1,"age"))
setattr(Student1,"age",25)
print(hasattr(Student1,"age"))
print(getattr(Student1,"age"))
delattr(Student1,"age")
print(hasattr(Student1,"age"))
```

name	*public*	*can be accessed from inside and outside.*
_name	*protected*	*like public member, but should not be directly accessed from outside*
__name	*private*	*can't be seen and accessed from outside.*

- ✓ *All member variables and methods are public by default in Python*

Inheritance

- ✓ *The process of acquiring properties from parent to child class is called inheritance.*

Classic class declaration not inheriting form any class.

```
class MyClass:                    # Don't inherit from anything.
```

While new-style classes inherit from either object or some other class.

```
class MyClass(object):          # Inherit from object, new-style class.
```

Single inheritance: *is when each class inherits from exactly one parent, or super class*

```
>class A(object): pass
 > class B(A): pass
```

object
|
A

Multilevel inheritance

```
>class A(object): pass
> class B(A): pass
> class C(B): pass
```

object

```
  |                                   B
  A                                   |
  |                                   C
```

hierarchical inheritance

```
>class A(): pass                      A
> class B(A): pass                   / \
> class C(A): pass                  B   C
```

multiple inheritance *:*

at least one of the classes involved inherits from two or more super classes. Here's a particularly simple example:

```
> class A(object): pass                object
> class B(A): pass                       |
> class C(A): pass                       A
> class D(B, C): pass  # multiple superclasses   / \
                                          B   C
                                           \ /
                                            D
```

ex:

```
class Parent:
  pass
class Child(Parent):
  pass

p = Parent()
c = Child()

print(isinstance(p,object))
print(isinstance(p,Parent))
print(isinstance(c,Child))
print(isinstance(c,object))
print(isinstance(p,Child))

print(issubclass(Parent,object))
print(issubclass(Child,object))
print(issubclass(Child,Parent))
print(issubclass(Parent,Child))
```

ex:

```
class Parent:
  def m1(self):
    print("Parent class m1")

class Child(Parent):
  def m2(self):
    print("Child class m2")

p = Parent()
p.m1()

c = Child()
c.m1()
c.m2()
```

ex:

```
class A:
    def m1(self):
        print("m1 of A called")

class B(A):
    def m2(self):
        print("m2 of B called")

class C(B):
    def m3(self):
        print("m3 of C called")

c = Child()
c.m1()
c.m2()
c.m3()
```

ex:

In below example 1-arg constructor not present in B class so parent class(A) constructor will be executed.

```
class A:
  def __init__(self,name):
     print("A class cons")
     self.name=name

class B(A):
   def disp(self):
```

```
        print(self.name)

    b = B("ratan")
    b.disp()
```

ex:

In below example 1-arg constructor present in B class so class B constructor will be executed.

```
    class A:
      def __init__(self,name):
        print("A class cons")
        self.name=name

    class B(A):
      def __init__(self,name):
        print("B class cons")
        self.name=name
      def disp(self):
        print(self.name)

    b = B("ratan")
    b.disp()
```

ex:

```
    class Parent:
      def m1(self):
        print("parent m1()")

    class Child(Parent):
      def m1(self):
        print("child m1()")
      def disp(self):
        self.m1()              # Current class function calling
        super().m1()           # parent class function calling

    c = Child()
    c.disp()
```

ex:

```
    class Parent:
      a,b=10,20

    class Child(Parent):
      def disp(self):
```

```
        print(self.a+self.b)

    c = Child()
    c.disp()
```

ex:

```
    class A(object):
        def save(self):
          print("class A saves")
    class B(A):
        def save(self):
            print("B saves stuff")
            super().save()
            #A.save(self)  # call the parent class method too
    class C(A):
        def save(self):
            print("C saves stuff")
            A.save(self)

    class D(B, C):
        def save(self):
            print ("D saves stuff")
            # make sure you let both B and C save too
            B.save(self)
            C.save(self)
```

```
        d = D()
        d.save()
```

Ex :

```
        class A(object):
            def save(self):
                print("class A saves")
        class B(A):
            def save(self):
                print("B saves stuff")
                super(B, self).save()
        class C(A):
            def save(self):
              print("C saves stuff")
              super(C, self).save()
        class D(B, C):
            def save(self):
                print ("D saves stuff")
                super(D, self).save()

        d = D()
        d.save()
```

ex:

```
class A:
   def m(self):
     pass

class B(A):
   def m(self):
      print("m of B called")
      super().m()
class C(A):
   def m(self):
      print("m of C called")
      super().m()

class D(B,C):
   pass
D().m()
```

Ex:

```
class A:
   def m(self):
      print("m of A called")
```

```
class B(A):
   def m(self):
      print("m of B called")

class C(A):
   def m(self):
      print("m of C called")
```

Case -1 : ***m of B called***

```
class D(B,C):
   pass
D().m()
```

Case-2 : ***m of C called***

```
class D(C,B):
   pass
D().m()
```

Ex:

```
class A:
   def m(self):
      print("m of A called")

class B(A):
   pass

class C(A):
   def m(self):
      print("m of C called")

class D(B,C):
   pass

x = D()
x.m()
```

output : ***"m of C called"***

Ex:

```
class A:
   def m(self):
      print("m of A called")

class B(A):
   def m(self):
      print("m of B called")

class C(A):
   pass

class D(C,B):
   pass

x = D()
x.m()
```

output : ***"m of B called"***

ex:

```
class A:
   def __init__(self):
```

```
        print("A.__init__ ")

class B(A):
    def __init__(self):
        print("B.__init__ ")
        super().__init__()

class C(A):
    def __init__(self):
        print("C.__init__ ")
        super().__init__()


class D(B,C):
    def __init__(self):
        print("D.__init__ ")
        super().__init__()
A()
B()
C()
D()
```

Ex:

```
class A:
    def m(self):
        print("m of A called")

class B(A):
    def m(self):
        print("m of B called")

class C(A):
    def m(self):
        print("m of C called")

class D(B,C):
    def m(self):
        B.m(self)
        C.m(self)
        A.m(self)
D().m()
```

Ex:

```
class A:
    def m(self):
```

```
        print("m of A called")

class B(A):
    def m(self):
        print("m of B called")
        A.m(self)

class C(A):
    def m(self):
        print("m of C called")
        A.m(self)

class D(B,C):
    def m(self):
        print("m of D called")
        B.m(self)
        C.m(self)
D().m()
```

Ex:

```
class Person:
    def __init__(self, personName, personAge):
        self.name = personName
        self.age = personAge

    def showName(self):
        print(self.name)

    def showAge(self):
        print(self.age)

class Student:
    def __init__(self, studentId):
        self.studentId = studentId

    def getId(self):
        return self.studentId


class Resident(Person, Student):
    def __init__(self, name, age, id):
        Person.__init__(self, name, age)
```

```
        # super().__init__(name,age)    # Another way to call super class members
        Student.__init__(self, id)


# Create an object of the subclass
resident1 = Resident('John', 30, '102')
resident1.showName()
resident1.showAge()
print(resident1.getId())
```

```
ex:
class A:
    def __init__(self):
        self.name = 'John'
    def getName(self):
        return self.name

class B:
    def __init__(self):
        self.name = 'Richard'
    def getName(self):
        return self.name

class C(A,B):
    def __init__(self):
        A.__init__(self)
        B.__init__(self)

    def getName(self):
        return self.name

C1 = C()
print(C1.getName())
```

ex:

```
class Person(object):
   def __init__(self, first, last):
      self.firstname = first
      self.lastname = last
   def Name(self):
      return self.firstname + " " + self.lastname

class Employee(Person):
   def __init__(self, first, last, staffnum):
      super().__init__(first,last)
      #Person.__init__(self,first, last)
      self.staffnumber = staffnum
   def GetEmployee(self):
      return self.Name() + ", " +  self.staffnumber

x = Person("ratan", "addanki")
y = Employee("ratan", "addanki", "111")

print(x.Name())
print(y.GetEmployee())
```

The __init__ method of our Employee class explicitly invokes the __init__method of the Person class. We could have used super instead. super().__init__(first, last) is automatically replaced by a call to the superclasses method, in this case __init__:

```
 def __init__(self, first, last, staffnum):
      super().__init__(first, last)
      self.staffnumber = staffnum
```

ex :

```
class Person:
  def __init__(self, first, last):
    self.firstname = first
    self.lastname = last

  def __str__(self):
    return self.firstname + " " + self.lastname

class Employee(Person):
  def __init__(self, first, last, id):
    super().__init__(first, last)
    self.id = id

x = Person("ratan", "addanki")
y = Employee("ratan", "addanki", "111")

print(x)
print(y)
```

ex: ***We have overridden the method __str__ from Person in Employee***

```
class Person:
  def __init__(self, first, last):
    self.firstname = first
    self.lastname = last

  def __str__(self):
    return self.firstname + " " + self.lastname

class Employee(Person):
  def __init__(self, first, last, id):
    super().__init__(first, last)
    self.id = id
  def __str__(self):
   return super().__str__()+" "+self.id

x = Person("ratan", "addanki")
y = Employee("ratan", "addanki", "111")

print(x)
print(y)
```

Example : ***method overriding***

```
class Animal:
   def eat(self):
     print ('Animal Eating...')

class Dog(Animal):
  def eat(self):
     print ('Dog eating...')

d = Dog()
d.eat()
```

Exception handling

Two distinguishable kinds of errors:

1. *syntax errors*
2. *Exceptions.*

- ✓ *An exception is an error that happens during the execution of a program.*
- ✓ *Whenever the exception raised program terminated abnormally rest of the application not executed.*
- ✓ *To overcome above problem to get the normal termination & to execute the remaining code handle the exception.*

 # Application without try-except block

Whenever the exception raised program terminated abnormally rest of the application not executed.

```
print("Hello")
print(10/0)
print("rest of the app……")
```

output :

```
Hello
ZeroDivisionError: division by zero
```

Disadvantages:

- *Program terminated abnormally.*

- *Rest of the application not executed.*

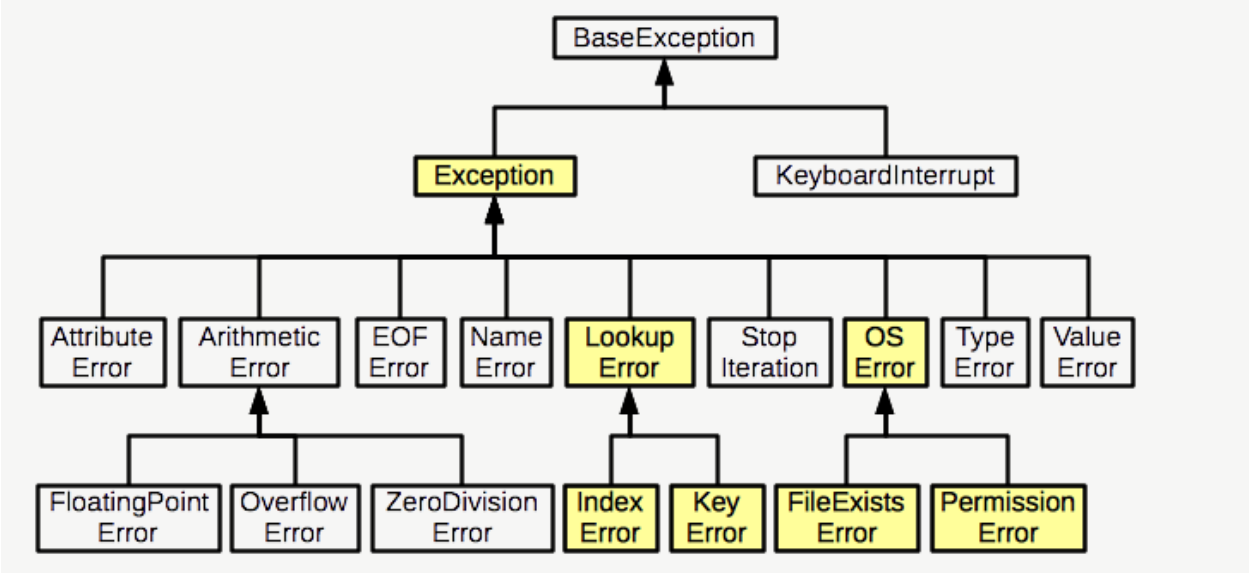
 # Application with try-except block

```
try:
   a=10/0
except ArithmeticError:
      print ("This statement is raising an exception")
print("rest of the Application")
```

output : *This statement is raising an exception*
rest of the Application

Advantages:

- *Rest of the application executed.*
- *Program terminated normally.*



Exception AssertionError : *Raised when an assert statement fails.*
ZeroDivisionError : *Occurs when a number is divided by zero.*
NameError : *It occurs when a name is not found. It may be local or global.*
IndentationError : *If incorrect indentation is given.*
IOError : *It occurs when Input Output operation fails.*
EOFError : *occurs when end of file is reached and yet operations are being performed*
IndexError : *Raised when a sequence subscript is out of range.*

ImportError	***:***	*Raised when an import statement fails to find the module definition*
KeyError	***:***	*Raised when a dictionary key is not found in the set of existing keys.*
MemoryError	*:*	*Raised when an operation runs out of memory*
OSError	***:***	*It is raised when a function returns a system-related error*
SyntaxError	***:***	*Raised when the parser encounters a syntax error.*
IndentationError	*:*	*Base class for syntax errors related to incorrect indentation.*
TabError	***:***	*Raised when indentation contains an inconsistent use of tabs and spaces*

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |    |    +-- FloatingPointError
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
      |    +-- EnvironmentError
      |    |    +-- IOError
      |    |    +-- OSError
      |    |         +-- WindowsError (Windows)
      |    |         +-- VMSError (VMS)
      |    +-- EOFError
      |    +-- ImportError
      |    +-- LookupError
      |    |    +-- IndexError
      |    |    +-- KeyError
      |    +-- MemoryError
      |    +-- NameError
      |    |    +-- UnboundLocalError
      |    +-- ReferenceError
      |    +-- RuntimeError
      |    |    +-- NotImplementedError
      |    +-- SyntaxError
      |    |    +-- IndentationError
      |    |         +-- TabError
      |    +-- SystemError
      |    +-- TypeError
      |    +-- ValueError
      |         +-- UnicodeError
      |              +-- UnicodeDecodeError
      |              +-- UnicodeEncodeError
      |              +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
```

Exception handling: *In python we can handle the exceptions by using try-except blocks.*

Syntax :

```
try:
   exceptional code
except Exception1:
   execute code if exception raised
except Exception2:
   execute code if exception raised
....
....
except ExceptionN:
   execute code if exception raised
except :
   code execute if other except blocks are not matched.
else:
   In case of no exception, execute the else block code.
```

Important points :

- ✓ *The try block contains exceptional code it may raise an exception or may not.*
- ✓ *If the exception raised in try block the matched except block will be executed.*
- ✓ *If the try block contains exception, If the except block is not matched program terminated abnormally.*
- ✓ *If no exception occurs then code under except clause will be skipped.*
- ✓ *Once the exception raised in try block remaining code of the try block is not executed.*
- ✓ *If the except blocks are not matched then default except block will be executed.*
- ✓ *The default except block must be last statement.*
- ✓ *The else block is executed if there is no exception in try block.*

Ex:

```
st-1
st-2
try:
        st-1
        st-2
except Exception1:
        st-1
        st-2
except Exception2:
        st-1
        st-2
except:
        st-1
        st-2
else :
        st-1
        st-2
st-1
st-2
```

```
ex :    print("ratan sir python is good")
        try:
                n = int(input("enter a number:"))
                print(10/n)
        except ZeroDivisionError:
                print(10/2)
        print("rest of the app")

ex:   except block is not matched so program terminated abnormally
        print("ratan sir python is good")
        try:
                n = int(input("enter a number:"))
                print(10/n)
        except TypeError:
                print(10/2)
        print("rest of the app")

ex:    exception with reference variable.
        print("ratan sir python is good")
        try:
                n = int(input("enter a number:"))
                print(10/n)
        except ArithmeticError as a:
                print ("exception =",a)
        print("rest of the app")

ex:
        print("ratan sir python is good")
        try:
                n = int(input("enter a number:"))
                print(10/n)
                print(10+"ratan")
        except ArithmeticError as a:
                print("ratanit.com")
        except TypeError as e:
                print("operations are not supported",e)
        except:
                print("durgasoft")
        else:
                print("no exception in app")
        print("rest of the app")

ex:     default 'except:' must be last
        except:
                print("durgasoft")
        except TypeError:
                print("ratanit.com")
SyntaxError: default 'except:' must be last
```

ex :

```
print("Application Started")
try:
   n = int(input("enter Your number: "))
   print(10/n)
except ValueError as e:
   print ("Enter only integers",e)
except ZeroDivisionError as a:
   print ("Infinity : entered value is zero",a)
except:
        print("durgasoft")
else:
        print("no exception in app")
print("Rest of the Application.....")
```

```
E:\>python first.py
        Application Started
        Your number: 2
        5
        Rest of the Application.....

E:\>python first.py
        Application Started
        Your number: 0
        'Infinity : entered value is zero', ZeroDivisionError:'integer division or modulo by zero',
        Rest of the Application.....

E:\>python first.py
        Application Started
        Your number: 'ratan'
        'Enter only integers', ValueError("invalid literal for int() with base 10: 'ratan'
        Rest of the Application.....
```

Ex:

```
print("ratan sir python is good")
try:
        n = int(input("enter a number:"))
        print(10/n)
        print(10+"ratan")
except (ArithmeticError,TypeError) as a:
        print("ratanit.com")
except:
        print("durgasoft")
else:
        print("no exception in app")
print("rest of the app")
```

ex: *Handling more than one exception by using single except block.*

```
try:
        n = int(input("enter a number"))
        print(10/n)
        str="ratan"
        print(str[10])
except (IndexError, ZeroDivisionError) as e:
        print("An ZeroDivisionError or a IndexError occurred :",e)
```

ex: ***The Exception , BaseException class is able to handle all exceptions.***

```
try:
   x = int(input("Your number: "))
   inverse = 10 / x
   print(inverse)
except Exception as e:
   print(e);

print("Rest of the Application.....")
```

case 1: *Your number: ratan*
could not convert string to float: 'ratan'
Rest of the Application.....

Case 2: *Your number: 0*
float division by zero
Rest of the Application.....

ex: *We learn from the below result that the function catches the exception.*

```
def f():
   x = int("four")

try:
   f()
except ValueError as e:
   print("got it : ", e)

print("Rest of the Application")
```

output : *got it : invalid literal for int() with base 10: 'four'*
Rest of the Application

Ex: *The exception will be caught inside of the function and not in the caller's exception*

```
def f():
    try:
        x = int("four")
    except ValueError as e:
        print("got it in the function :-) ", e)

try:
    f()
except ValueError as e:
    print("got it :-) ", e)

print("Rest of the Application")
```

output : *got it in the function :-) invalid literal for int() with base 10: 'four'*
Rest of the Application

Ex:

```
def m1():
        n = int(input("enter a numner:"))
        print(10/n)

def m2():
        print(10+"ratan")

try:
        m1()
        m2()
except ArithmeticError as a:
        print("ratanit.com")
except:
        print("durgasoft")
else:
        print("no exception")
print("rest of the app")
```

Finally block :

- ✓ *The try block contains exceptional code it may raise an exception or may not.*
- ✓ *If the exception raised in try block the matched except block will be executed.*
- ✓ *If the except blocks are not matched then default except block will be executed.*
- ✓ *The default except block must be last statement.*
- ✓ *The else block is executed if there is no exception in try block.*
- ✓ *Finally block is always executed irrespective of try-except blocks.*
- ✓ *The finally block is used to write the resource releasing code.*

```
try:
   <exceptional code>
except <ExceptionType1>:
   <handler1>
except <ExceptionTypeN>:
   <handlerN>
except:
   <handlerExcept>
else:
   <process_else>
finally:
   <process_finally>
```

Ex:

```
try:
  n = int(input("enter a number"))
  print(10/n)
except ArithmeticError as e:
  print("ratanit.com")
else:
  print("no xception")
finally:
  print("finally block")
```

ex:

```
try:
  print(10+"ratan")
except ArithmeticError as e:
  print("ratanit.com")
finally:
  print("finally block")
```

ex:

```
try:
  print("try block")
finally:
  print("finally block")
```

ex : Invalid : else must be with presence of except bock.

```
try:
 print("try block")
else:
  print("no exception")
finally:
  print("finally block")
```

In two cases finally block is not executed

- *if the control is not entered in try block*
- *when we use os._exit(0)*

case 1: The control is not entered in try block so finally block is not executed.

```
print(10/0)
try:
        print("try block")
finally:
        print("finally block")
```

case 2: when we use os._exit(0) virtual machine is shutdown so finally block is not executed.

```
import os
try:
        print("try block")
        os._exit(0)
finally:
        print("finally block")
```

ex: If the try,except , finally block contains exceptions : it display finally block exception

```
try:
  print(10/0)
except ArithmeticError as e:
   print("ratan"+10)
finally:
  s="ratan"
  print(s[10])
```

ex: If the try,except , finally block contains return statement : it display finally block return

```
def m1():
 try:
  return 10
 except ArithmeticError as e:
  return 20
 finally:
  return 30

print(m1())
```

ex :
```
try:
   num1, num2 = eval(input("Enter two numbers, separated by a comma : "))
   result = num1 / num2
   print("Result is", result)
except ZeroDivisionError:
   print("Division by zero is error !!")
except SyntaxError:
   print("Comma is missing. Enter numbers separated by comma like this 1, 2")
except:
   print("Wrong input")
else:
   print("No exceptions")
finally:
   print("This will execute no matter what")
```

case 1: *Enter two numbers, separated by a comma : 10,2*
Result is 5.0
No exceptions
This will execute no matter what

Case 2: *Enter two numbers, separated by a comma : 10,0*
Division by zero is error !!
This will execute no matter what

Case 3: *Enter two numbers, separated by a comma : 10.6*
Wrong input
This will execute no matter what

Nested try-except block:

- ✓ *In outer try block if there is no exception then outer else block will be executed.*
- ✓ *In inner try block if there is no exception then inner else block will be executed.*

```
Ex:     try:
          print("outer try block")
          n = int(input("enter a number"))
          print(10/n)
                  try:
                          print("inner try")
                          print("anu"+"ratan")
                  except TypeError:
                          print("ratanit.com")
                  else:
                          print("inner no exception")
          except ArithmeticError:
                  print(10/5)
          else:
                  print("outer no excepiton")
          finally:
                  print("finally block")
```

ex:

```
n = int(input("enter a number:"))
try:
  print("outer try block")
  try:
    print("Inner try block")
    print(10/n)
  except NameError:
    print("Inner except block")
  finally:
    print("Inner finally block")
except ZeroDivisionError:
  print("outer except block")
else:
   print("else block execute")
finally:
  print("outer finally block")
print("Rest of the Application")
```

enter a number: 0

```
outer try block
Inner try block
Inner finally block
outer except block
outer finally block
Rest of the Application
```

enter a number: 2

```
outer try block
Inner try block
5.0
Inner finally block
else block execute
outer finally block
Rest of the Application
```

ex :

```
st-1
st-2
try:
        st-3
        st-4
        try:
                st-5
                st-6
        except:
                st-7
                st-8
except:
        try:
                st-9
                st-10
        except:
                st-11
                st-12
else:
        st-13
        st-14
finally:
        st-15
        st-16
st-17
st-18
```

case 1: *No exception in above example.*
Case 2: *Exception raised in st-2*
Case 3: *exception raised in st-3 the except block is matched.*
Case 4: *exception raised in st-4 the except block is not matched.*
Case 5: *exception raised in st-5 the except block is matched.*
Case 6: *exception raised in st-6 the inner except block is not matched but outer except block is matched .*

Case 7 : *Exception raised in st-5 the except block is matched while executing except block exception raised in st-7 the inner except block executed while executing inner except block exception raised in st-9 , the inner except block executed while executing inner except exception raised in st-11.*

Case 8 : *Exception raised in st-6 the except block is matched while executing except block exception raised in st-8 the inner except block executed while executing inner except block exception raised in st-10 , the inner except block executed while executing inner except exception raised in st-12.*

Case 9 : *exception raised in st-13*
Case 10 : *exception raised in st-15*
Case 11 *: exception raised in st-18*

There are two types of exceptions.

1. *predefined exception : ArithmeticError*
2. *user defined exceptions : InvalidAgeError*

- ✓ *By using raise keyword we can raise predefined exceptions & user defined exceptions.*
- ✓ *By using raise keyword it is not recommended to raise predefined exceptions because predefined exceptions contains some fixed meaning(don't disturb the meaning).*

  raise ArithmeticError("name is not good")

- ✓ *By using raise keyword it is recommended to raise user defined exceptions.*

  raise InvalidAgeError("age is not good")

raise keyword :

✓ *To raise your exceptions from your own methods you need to use raise keyword.*

```
raise ExceptionClassName("Your information")
```

ex :

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

ex1:

```
def status(age):
   if age < 0:
      raise ValueError("Only positive integers are allowed")

   if age>22:
      print("eligible for mrg")
   else:
      print("not eligible for mrg try after some time")

try:
   num = int(input("Enter your age: "))
   status(num)
finally:
  print("finally block....")
```

```
Enter your age:  23
finally block....
ValueError: Only positive integers are allowed
```

Ex:
```
def status(age):
    if age < 0:
        raise ValueError("Only positive integers are allowed")
    if age>22:
        print("eligible for mrg")
    else:
        print("not eligible for mrg try after some time")

try:
    num = int(input("Enter your age: "))
    status(num)
except ValueError:
    print("Only positive integers are allowed you ......")
finally:
  print("finally block....")
```

User defined exceptions :

ex :
```
class NegativeAgeException(RuntimeError):
    def __init__(self, age):
        super().__init__()
        self.age = age

def status(age):
    if age < 0:
        raise NegativeAgeException("Only positive integers are allowed")
    if age>22:
        print("Eligible for mrg")
    else:
        print("not Eligible for mrg....")

try:
    num = int(input("Enter your age: "))
    status(num)
except NegativeAgeException:
    print("Only positive integers are allowed")
except:
    print("something is wrong")
```

```
ex :    class YoungException(Exception):
          def __init__(self,age):
            self.age=age

        class OldException(Exception):
         def __init__(self,age):
           self.age=age

        age=int(input("Enter Age:"))
        if age<18:
           raise YoungException("Plz wait some time ")
        elif age>65:
           raise TooOldException("Your age  too old")
        else:
           print("we will find one girl soon”)
```

Enter Age: 12

Check the output

Enter Age: 89

Check the output

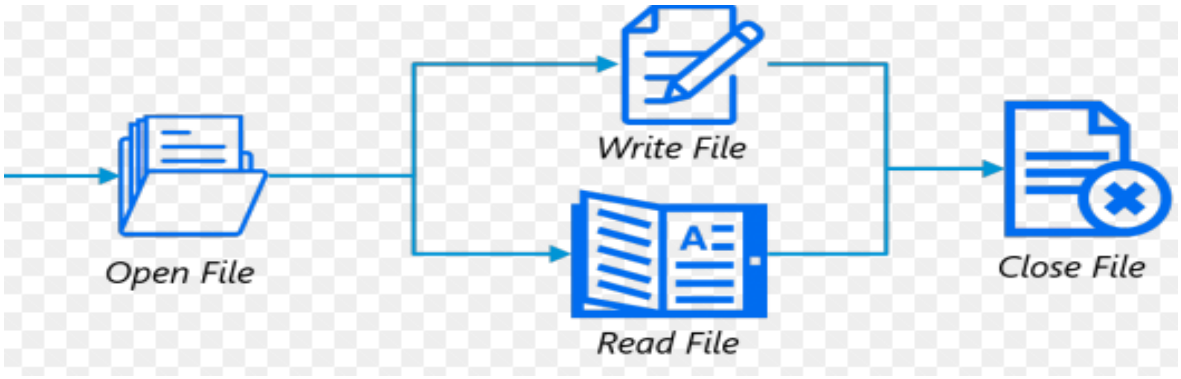
```
ex :    class TooYoungException(Exception):
          def __init__(self,age):
            self.age=age

        class TooOldException(Exception):
         def __init__(self,age):
           self.age=age

        try:
                age=int(input("Enter Age:"))
                if age<18:
                   raise YoungException("Plz wait some time ")
                elif age>65:
                   raise TooOldException("Your age  too old")
                else:
                   print("we will find one girl soon”)
        except   YoungException as e:
          print("Plz wait some time ")
        except  OldException as e:
          print("Your age  too old ")
```

File Handling in Python

- ✓ *Python can be used to read and write data. Also it supports reading and writing data to Files.*
- ✓ *File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).*

To perform file handling, we need to perform these steps:

Open File
Read / Write File
Close File



Opening a File:

Before working with Files you have to open the File. To open a File, Python built in function open() is used. It returns an object of File which is used with other functions. Having opened the file now you can perform read, write, etc. operations on the File.

The open function takes two arguments, the name of the file and the mode of operation.
The default file operations is read mode

```
f = open("test.txt")                    # open file in current directory
f = open("C:/Python33/ratan.txt")       # specifying full path
```

Writing to a File: *write() method is used to write a string into a file.*

Reading from a File: *read() method is used to read data from the File.*

The read functions contains different methods, read(),readline() and readlines()

```
read()          #return one big string
readline        #return one line at a time
readlines       #returns a list of lines
```

Closing a File:

Once you are finished with the operations on File at the end you need to close the file. It is done by the close() method. close() method is used to close a File.

Python File Modes

Mode	Description
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

ex *: writing the data to file*

We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased.

```
text = "hi ratan sir \n welcome to ratanit"
f = open("sample.txt","w")
f.write(text)
f.close()
print("Operations are completed....")
```

ex : *To read the data from file , the file is mandatory otherwise we will get* ***IOError****.*

```
try:
        f1=open("sample.txt","r")
        s=f1.read()
        print (s)
except IOError as e:
        print(e)
finally:
        f1.close()
print("rest of the application")
```

ex: in below example file is not available so the except block will be executed.

```
try:
        f1=open("sdss.txt","r")
        s=f1.read()
        print (s)
except IOError as e:
        print(e)
print("rest of the application")

E:\>python first.py
'exception raised',No such file or directory: 'sdss.txt'
rest of the application
```

ex:

```
obj=open("abc.txt","w")
obj.write("Welcome to ratanit")
obj.close()

f1=open("abc.txt","r")
s=f1.read()
print (s)
f1.close()

f2=open("abc.txt","r")
s1=f2.read(10)
print (s1)
f2.close()
```

ex:

```
try:
        f1=open("sample.txt","r")
        s=f1.read(5)
        print (s)
        print(f1.tell())
except IOError as e:
        print("exception raised",e)
print("rest of the application")
```

ex :

- ✓ *When we read the data from file after reading the data the cursor is present in same location.*
- ✓ *To overcome above problem to move the cursor to particular location use seek() function.*

```
f = open("sample.txt","r")
s = f.read()                # entire file data
print (s)

f.seek(0)
```

```
s1 = f.read(2)          # read only 2-characters
print (s1)

f.seek(5)
s1 = f.read(3)          # read only 3-characters
print (s1)

f.close()
print ("operations are completed")
```

ex:

we can read the data in three ways

1. *Read ()*
2. *Readline()*
3. *Readlines()*

```
#write operations
f = open("aaa.txt", "w")
f.write("hi ratan sir\nhow r u")
f.close()

#Read operations ; read complete String
f1 = open("aaa.txt","r")
s = f1.read()
print(s)

f1.seek(0)
#To read one line at a time, use:
print (f1.readline())

f1.seek(0)
#To read a list of lines use
print (f1.readlines())
```

ex :

```
f1 = open("sample.txt")
f2 = open("abc.txt","w")

for line in f1:
        f2.write(line)
f1.close()
f2.close()
```

ex : *when we declare the file by using with statement he file is automatically closed once the execution completed .*

```
with open("sample.txt", "w") as fh:
   fh.write("hi friends\nwelcome to ratanit!\n")
```

Append () :

- ✓ *The append function is used to append to the file instead of overwriting it.*
- ✓ *To append to an existing file, simply open the file in append mode ("a").*

```
#write operations
f = open("aaa.txt", "w")
f.write("hi ratan sir\nhow r u")
f.close()

#Append operations
fh = open("aaa.txt", "a")
fh.write("\nHello World")
fh.close()

#Read operations
f1 = open("aaa.txt","r")
s = f1.read()
print(s)
```

ex:

- ✓ ***r+*** *opening a file both reading & writing operations.*
- ✓ *While performing operations on the file if it is not available we will get error message.*
- ✓ *In this mode while performing write operations the data will be replaced with existing data.(check the below example output)*

```
f = open("sample.txt","r+")
x = f.read()
print(x)
```

```
f.seek(0)

f.write("aaa")
f.seek(0)
x = f.read()
print(x)

E:\>python first.py
ratanit
aaaanit
```

ex:

- ✓ *W+ opening a file both read & writes operations.*
- ✓ *When we open the file, if the file is not available then it will create the file.*
- ✓ *When we open the file, if the file is available then it will erase the file data then creates empty file.*
- ✓ *In this mode while performing write operations the data will be completely replaced with existing data.*

```
f = open("sample.txt","w+")
f.write("aaa")
f.seek(0)
x = f.read()
print(x)
```

ex:

- ✓ *A+ opening a file both read & append mode.*
- ✓ *When we open the file, if the file is not available then it will create the file.*
- ✓ *When we open the file, if the file is available then the cursor is present in end of the file.*

```
fh = open("sample.txt", "a+")
fh.write("\nHello World")
#read operataions
fh.seek(0)
s = fh.read()
print(s)
fh.close()
```

ex : ***Name*** *Returns the name of the file.*
Mode *Returns the mode in which file is being opened.*
Closed *Returns Boolean value. True, in case if file is closed else false.*

```
obj = open("data.txt", "w")
print (obj.name)
print (obj.mode)
print (obj.closed)
obj.close()
print (obj.closed)
```

file operations module ***: os***

- ✓ *There is a module "os" defined in Python that provides various functions which are used to perform various operations on Files.*
- ✓ *To use these functions 'os' needs to be imported by using import keyword.*

rename() *: It is used to rename a file. It takes two arguments, existing_file_name and new_file_name.*
remove() *: It is used to delete a file. It takes one argument.*
mkdir() : *It is used to create a directory. A directory contains the files. It takes one argument which is the name of the directory.*
chdir() *: It is used to change the current working directory. It takes one argument which is the name of the directory.*
getcwd() *It gives the current working directory.*
rmdir() *It is used to delete a directory. It takes one argument which is the name of the directory.*
tell() *It is used to get the exact position in the file.*

ex : rename() remove() functions

```
import os
os.rename("sample.txt","ratan.txt")
os.remove("Test.java")
```

ex :

```
import os
os.mkdir("new")
os.chdir("hadoop")
print(os.getcwd())
```

ex : *In order to delete a directory, it should be empty. In case directory is not empty first delete the files*

```
import os
os.rmdir("new")
```

Abstraction :

- ✓ *The abstract class contains one or more abstract methods.*
- ✓ *The abstract method contains only method declaration but not implementation.*
- ✓ *In Python comes with a module which provides the infrastructure for defining Abstract Base Classes (ABCs)s not possible to create the object of abstract classes.*
- ✓ *A class that is derived from an abstract class cannot be instantiated unless all of its abstract methods are overridden.*

Ex:

```
from abc import ABC, abstractmethod
class Test(ABC):
  @abstractmethod
  def m1(self):
    pass

class Test1(Test):
  def m1(self):
    print("implementation here")

#t = Test()    # abstract class object creation not allowed
t1 = Test1()
t1.m1()
```

ex:

```
from abc import ABCMeta, abstractmethod
class Animal(object):
 __metaclass__=ABCMeta
 @abstractmethod
 def eat(self):
    pass
class Tiger(Animal):
  def eat(self):
    print("Tiger implementation ...")
Tiger().eat()
```

ex:

```
from abc import ABC, abstractmethod
class Animal(ABC):
  @abstractmethod
  def eat(self):
    pass

class Tiger(Animal):
  def eat(self):
    print("Tiger implementation ...")
```

```
class Lion(Animal):
  def eat(self):
    print("Lion implementation here...")

t = Tiger()
t.eat()
l = Lion()
l.eat()
```

ex: *Abstract class constructor.*

```
from abc import ABC, abstractmethod

class AbstractClassExample(ABC):

    def __init__(self, value):
        self.value = value
        super().__init__()

    @abstractmethod
    def do_something(self):
        pass

class DoAdd(AbstractClassExample):
    def do_something(self):
        return self.value + 42

class DoMul(AbstractClassExample):

    def do_something(self):
        return self.value * 42

x = DoAdd(10)
y = DoMul(10)
print(x.do_something())
print(y.do_something())
```

ex:
ex 1:

```
from abc import ABC, abstractmethod
class Animal(ABC):
  @abstractmethod
  def eat1(self):
    pass
@abstractmethod
  def eat2(self):
    pass
```

```
class Tiger(Animal):
  def eat1(self):
    print("Tiger implementation ...")
  def eat2(self):
    print("Tiger implementation ...")

t = Tiger()
t.eat1()
t.eat2()
```

ex 2:

```
from abc import ABC, abstractmethod
class Animal(ABC):
  @abstractmethod
  def eat1(self):
    pass
  @abstractmethod
  def eat2(self):
    pass
class Tiger(Animal):
  def eat1(self):
    print("Tiger implementation ...")

t = Tiger()
t.eat1()
```

TypeError: Can't instantiate abstract class Tiger with abstract methods eat2

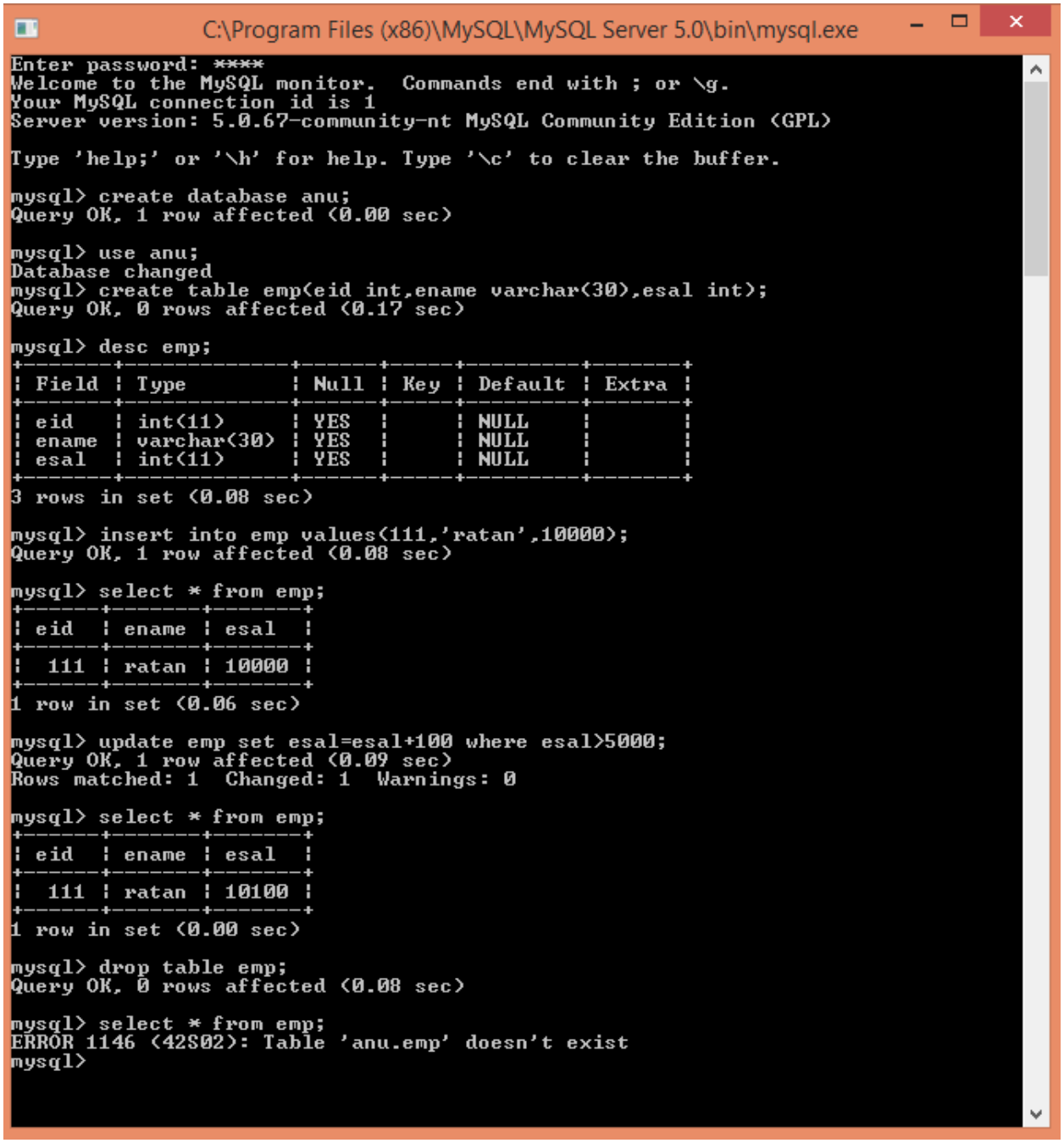
ex 3 :

```
from abc import ABC, abstractmethod
class Animal(ABC):
  @abstractmethod
  def eat1(self):
    pass
  @abstractmethod
  def eat2(self):d
    pass
class Tiger(Animal):
  def eat1(self):
    print("Tiger implementation ...")
class lion(Tiger):
  def eat2(self):
    print("lion implementation ...")

t = lion()
t.eat1()
t.eat2()
```

ex:
The pass statement does nothing. It can be used when a statement is required syntactically but the
while True:
  pass ...
This is commonly used for creating minimal classes:
class MyEmptyClass:
  pass

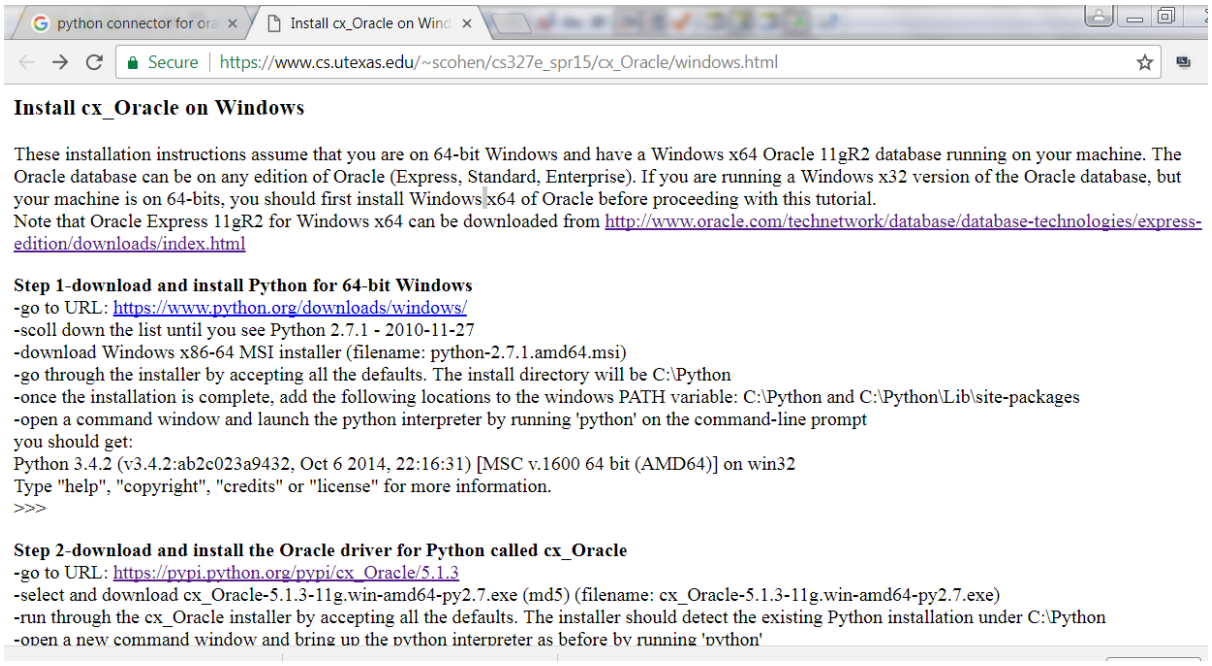
def initlog(*args):
        pass   # Remember to implement this!

Mysql :


C:\Program Files (x86)\MySQL\MySQL Server 5.0\bin\mysql.exe
Enter password: ****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.0.67-community-nt MySQL Community Edition (GPL)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> create database anu;
Query OK, 1 row affected (0.00 sec)
mysql> use anu;
Database changed
mysql> create table emp(eid int,ename varchar(30),esal int);
Query OK, 0 rows affected (0.17 sec)
mysql> desc emp;
+-------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
eid	int(11)	YES		NULL	
ename	varchar(30)	YES		NULL	
esal	int(11)	YES		NULL	
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.08 sec)
mysql> insert into emp values(111,'ratan',10000);
Query OK, 1 row affected (0.08 sec)
mysql> select * from emp;
+------+-------+-------+
| eid | ename | esal |
+------+-------+-------+
| 111 | ratan | 10000 |
+------+-------+-------+
1 row in set (0.06 sec)
mysql> update emp set esal=esal+100 where esal>5000;
Query OK, 1 row affected (0.09 sec)
Rows matched: 1 Changed: 1 Warnings: 0
mysql> select * from emp;
+------+-------+-------+
| eid | ename | esal |
+------+-------+-------+
| 111 | ratan | 10100 |
+------+-------+-------+
1 row in set (0.00 sec)
mysql> drop table emp;
Query OK, 0 rows affected (0.08 sec)
mysql> select * from emp;
ERROR 1146 (42S02): Table 'anu.emp' doesn't exist
mysql>

Ex:

```
import mysql.connector
try:
    con=mysql.connector.connect(host="localhost",user="root",password="root",database="ratan")
        print("Connection successfull")
        cursor=con.cursor()
        cursor.execute("create table emp(eid int,ename varchar(20),esal int)")
        print("table created successfully....")
        cursor.close()
        con.close()
except:
        print("operations are fail....")
```



python connector for ora | Install cx_Oracle on Win | Secure | https://www.cs.utexas.edu/~scohen/cs327e_spr15/cx_Oracle/windows.html

Install cx_Oracle on Windows

These installation instructions assume that you are on 64-bit Windows and have a Windows x64 Oracle 11gR2 database running on your machine. The Oracle database can be on any edition of Oracle (Express, Standard, Enterprise). If you are running a Windows x32 version of the Oracle database, but your machine is on 64-bits, you should first install Windows x64 of Oracle before proceeding with this tutorial.
Note that Oracle Express 11gR2 for Windows x64 can be downloaded from http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html

Step 1-download and install Python for 64-bit Windows
-go to URL: https://www.python.org/downloads/windows/
-scoll down the list until you see Python 2.7.1 - 2010-11-27
-download Windows x86-64 MSI installer (filename: python-2.7.1.amd64.msi)
-go through the installer by accepting all the defaults. The install directory will be C:\Python
-once the installation is complete, add the following locations to the windows PATH variable: C:\Python and C:\Python\Lib\site-packages
-open a command window and launch the python interpreter by running 'python' on the command-line prompt
you should get:
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>

Step 2-download and install the Oracle driver for Python called cx_Oracle
-go to URL: https://pypi.python.org/pypi/cx_Oracle/5.1.3
-select and download cx_Oracle-5.1.3-11g.win-amd64-py2.7.exe (md5) (filename: cx_Oracle-5.1.3-11g.win-amd64-py2.7.exe)
-run through the cx_Oracle installer by accepting all the defaults. The installer should detect the existing Python installation under C:\Python
-open a new command window and bring up the python interpreter as before by running 'python'

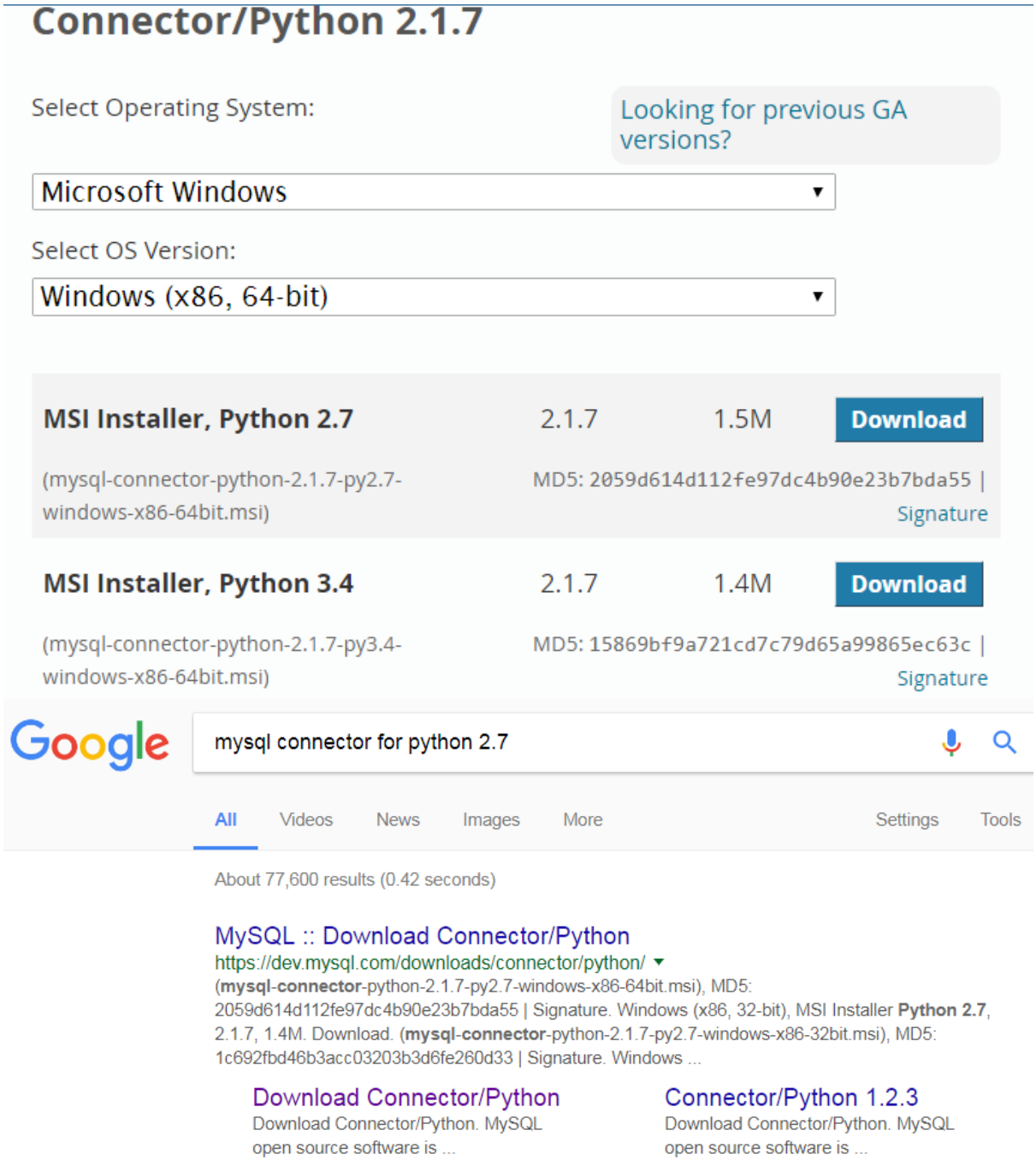
ex: Table creation & insertion & updating

```
import mysql.connector
```

```
import time
try:
    con=mysql.connector.connect(host="localhost",user="root",password="root",database="ratan")
        print("Connection successfull")
        cursor=con.cursor()
        cursor.execute("create table emp(eid int,ename varchar(20),esal int)")
        print("table created successfully....")
        time.sleep(1)

        cursor.execute("insert into emp values('%d','%s','%g')" %(111,"ratan",10000))
        cursor.execute("insert into emp values('%d','%s','%g')" %(222,"anu",20000))
        cursor.execute("insert into emp values('{0}','{1}','{2}')".format(333,"durga",30000))
        print("values are inserted successfully....")
        time.sleep(1)

        cursor.execute("update emp set esal=esal+{} where esal>{}".format(100,10000))
        print("values are updated successfully....")
        time.sleep(1)

        cursor.close()
        con.commit()
        con.close()
except:
        print("operataions are fail.....")


ex:
import mysql.connector
try:
        con=mysql.connector.connect(host="localhost",user="root",password="root",database="ratan")
        print("Connection successfull")

        cursor=con.cursor()
        cursor.execute("select * from emp")
        results = cursor.fetchall()
         # Fetch all the rows
        for row in results:
                eid = row[0]
                ename = row[1]
                esal = row[2]
                # Now print fetched result
                print "eid=%d ,ename=%s ,esal=%g" %(eid,ename,esal)

        cursor.close()
        con.close()
except:
        print("operataions are fail.....")
```

```
try:
    connection = getConnection(database)
    cursor = connection.cursor()
    cursor.execute("some query")
except:
    log.error("Problem.")
    raise
finally:
    cursor.close()
    connection.close()ra
```

```
connection = None
cursor = None

try:
    connection = getConnection(database)
    cursor = connection.cursor()
    cursor.execute("some query")
except:
    log.error("Problem.")
    raise
finally:
    if cursor is not None:
        cursor.close()
    if connection is not None:
        connection.close()
```


Connector/Python 2.1.7
Select Operating System:
Looking for previous GA versions?
Microsoft Windows
Select OS Version:
Windows (x86, 64-bit)
MSI Installer, Python 2.7
2.1.7
1.5M
Download
(mysql-connector-python-2.1.7-py2.7-windows-x86-64bit.msi)
MD5: 2059d614d112fe97dc4b90e23b7bda55 | Signature
MSI Installer, Python 3.4
2.1.7
1.4M
Download
(mysql-connector-python-2.1.7-py3.4-windows-x86-64bit.msi)
MD5: 15869bf9a721cd7c79d65a99865ec63c | Signature

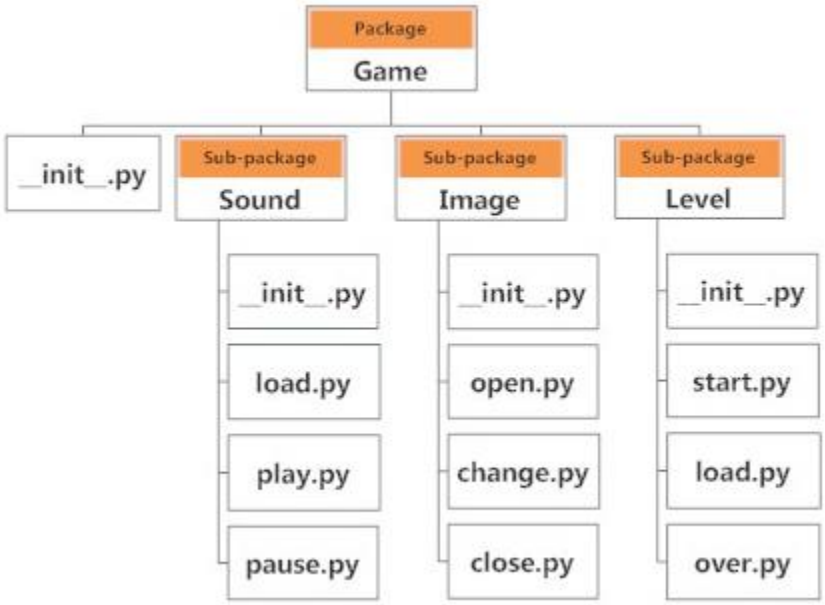
Google
mysql connector for python 2.7
All
Videos
News
Images
More
Settings
Tools
About 77,600 results (0.42 seconds)
MySQL :: Download Connector/Python
https://dev.mysql.com/downloads/connector/python/
(mysql-connector-python-2.1.7-py2.7-windows-x86-64bit.msi), MD5: 2059d614d112fe97dc4b90e23b7bda55 | Signature. Windows (x86, 32-bit), MSI Installer Python 2.7, 2.1.7, 1.4M. Download. (mysql-connector-python-2.1.7-py2.7-windows-x86-32bit.msi), MD5: 1c692fbd46b3acc03203b3d6fe260d33 | Signature. Windows ...
Download Connector/Python
Download Connector/Python. MySQL open source software is ...
Connector/Python 1.2.3
Download Connector/Python. MySQL open source software is ...

- ✓ *If you quit from the Python interpreter and enter it again, the definitions you have made (functions and variables) are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead. This is known as creating a script.*
- ✓ *As your program gets longer, you may want to split it into several files for easier maintenance*

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module
as a directory can contain sub-directories and files, a Python package can have sub-packages and modules.
Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory. Analogous to this, Python has packages for directories and modules for files.
definitions from a module can be imported into other modules or into the main module
A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended.
Python modules are .py files that consist of Python code. Any Python file can be referenced as a module.

By using The from...import Statement we can import the specific function from the particular module.
from mod-name import name1, name2, ... nameN

import <module-name>
from <module-name> import <name>
*from <module-name> import **



Example :

fibo.py

```
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
```

```
        print b,
        a, b = b, a+b
```

first.py

case 1:
```
import fibo
 fibo.fib(100)
```

case 2:
```
from fibo import fib
fib(100)
```

case 3:
```
from fibo import *
fib(100)
```

Example :

arithmetic.py
```
def add(x, y):
   return x + y
def multiply(x, y):
   return x * y
```

first.py
```
import arithmetic

print (arithmetic.add(5, 8))
print (arithmetic.multiply(12, 6))
```

user defined module functions

ex:
```
import fibo
print(dir(fibo))
```

Working with two different packages :
Accessing Modules from Another Directory
Modules may be useful for more than one programming project, and in that case it makes less sense to keep a module in a particular directory that's tied to a specific project.

If you want to use a Python module from a location other than the same directory where your main program is, you have a few options.

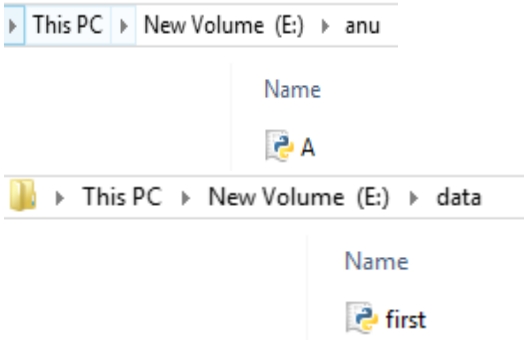
Appending Paths
One option is to invoke the path of the module via the programming files that use that module. This should be considered more of a temporary solution that can be done during the development process as it does not make the module available system-wide.

To append the path of a module to another programming file, you'll start by importing the sys module alongside any other modules you wish to use in your main program file.

The sys module is part of the Python Standard Library and provides system-specific parameters and functions that you can use in your program to set the path of the module you wish to implement.

For example, let's say we moved the hello.py file and it is now on the path /usr/sammy/ while the main_program.py file is in another directory.

In our main_program.py file, we can still import the hello module by importing the sys module and then appending /usr/sammy/ to the path that Python checks for files.



First.py:

E:\data\first.py - EditPlus

```
import sys
sys.path.append('E:/anu')

import A
A.m1()
```

Example *:*
step 1: create the package directory.
Step 2: create the modules in that directory (python files .py)
Step 3: create the client file import that modules

Fruits.py

```
class MyFruits:
        def disp(self):
                print("I like Orange & Apple")
```

Foods.py :

```
class Food:
        def disp(self):
                print("i like idly")
```

first.py file in different ways

first.py

```
import Fruits
import Foods

m = Fruits.MyFruits()
m.disp()
f = Foods.Food()
f.disp()
```

first.py

```
from Fruits import MyFruits
from Foods import Food
m = MyFruits()
m.disp()
f = Food()
f.disp()
```

first.py

```
from Fruits import *
from Foods import *
m = MyFruits()
m.disp()
f = Food()
f.disp()
```

```
E:\data>python first.py
I like Orange & Apple
i like idly
```

first.py

Example :

```
import Fruits
c = dir(Fruits)
print(c)

from Fruits import MyFruits
c = dir(MyFruits)
print(c)
```

ex:

Employee.py

```
class Emp:
  def __init__(self, eid, ename):
    self.eid = eid
    self.ename = ename
  def disp(self):
          print ("Emp id : ",self.eid , "Emp name: ",self.ename)

class MyClass:
        def disp(self):
                print("ratan sir good")
```

Student.py

```
class Stu:
  def __init__(self, rollno, name):
    self.rollno = rollno
    self.name = name

  def disp(self):
          print ("rollno : ",self.rollno , "name: ",self.name)

class A:
        def disp(self):
                print("Anu is good")
```

client.py

```
import Employee
import Student

e = Employee.Emp(111,"ratan")
e.disp()
m = Employee.MyClass()
m.disp()

s = Student.Stu(1,"anu")
s.disp()
c = Student.A()
c.disp()
```

Predefined modules:

math module :

ex:

```
import math
content = dir(math)
print (content)
```

```
E:\data>python first.py
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Example :

```
import math
print(math.ceil(30.3))
print(math.fabs(10))
print(math.factorial(4))
print(math.floor(30.9))
print(math.pow(3,4))
print(math.sqrt(4))
print(math.sin(90))
print(math.cos(90))
print(math.pi)
print(math.e)
```

math.pi : *The mathematical constant π = 3.141592…, to available precision.*
math.e: *The mathematical constant e = 2.718281…, to available precision.*
math.cos(x):*Return the cosine of x radians.*
math.sin(x): *Return the sine of x radians.*
math.pow(x, y):*Return x raised to the power y.*
math.sqrt(x) : *Return the square root of x.*
math.ceil(x) :*Return ceiling of x as a float, the smallest integer value greater than or equal to x*
math.fabs(x) : *Return the absolute value of x.*
math.factorial(x) : *Return x factorial. Raises ValueError if x is not integral or is negative.*
math.floor(x) : *Return the floor of x as a float, the largest integer value less than or equal to x.*

ex:

```
from math import sqrt
print(sqrt(4))
```

os module :

ex:

```
import os
content = dir(os)
print (content)
```

```
E:\data>python first.py
['F_OK', 'O_APPEND', 'O_BINARY', 'O_CREAT', 'O_EXCL', 'O_NOINHERIT', 'O_RANDOM',
 'O_RDONLY', 'O_RDWR', 'O_SEQUENTIAL', 'O_SHORT_LIVED', 'O_TEMPORARY', 'O_TEXT',
 'O_TRUNC', 'O_WRONLY', 'P_DETACH', 'P_NOWAIT', 'P_NOWAITO', 'P_OVERLAY', 'P_WAI
T', 'R_OK', 'SEEK_CUR', 'SEEK_END', 'SEEK_SET', 'TMP_MAX', 'UserDict', 'W_OK', '
X_OK', '_Environ', '__all__', '__builtins__', '__doc__', '__file__', '__name__', '__package__',
'_copy_reg', '_execvpe', '_exists', '_exit', '_get_exports_list', '_make_stat_result',
'_make_statvfs_result', '_pickle_stat_result', '_pickle_statvfs_result', 'abort', 'access', 'altsep',
'chdir', 'chmod', 'close', 'closerange', 'curdir', 'defpath', 'devnull', 'dup', 'dup2', 'environ', 'errno',
'error', 'execl', 'execle', 'execlp', 'execlpe', 'execv', 'execve', 'execvp', 'execvpe', 'extsep', 'fdopen',
'fstat', 'fsync', 'getcwd', 'getcwdu', 'getenv', 'getpid', 'isatty', 'kill', 'linesep', 'listdir', 'lseek', 'lstat',
'makedirs', 'mkdir','name', 'open', 'pardir', 'path', 'pathsep', 'pipe', 'popen', 'popen2', 'popen3'
, 'popen4', 'putenv', 'read', 'remove', 'removedirs', 'rename', 'renames', 'rmdir', 'sep', 'spawnl',
'spawnle', 'spawnv', 'spawnve', 'startfile', 'stat', 'stat_float_times', 'stat_result', 'statvfs_result',
'strerror', 'sys', 'system', 'tempnam', 'times', 'tmpfile', 'tmpnam', 'umask', 'unlink', 'unsetenv',
'urandom', 'utime', 'waitpid', 'walk', 'write']
```

Example:

```
import  os
os.rename("sample.txt","ratan.txt")
os.remove("ratan.txt")

os.mkdir("new")
os.chdir("data")
print(os.getcwd())
os.rmdir("new")
```

Example:

```
from os import  remove
remove("ratan.txt")
```

random Module :

ex:

```
import random
c = dir(random)
print(c)
```

```
E:\data>python first.py
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST', 'System
Random', 'TWOPI', 'WichmannHill', '_BuiltinMethodType', '_MethodType', '__all__'
, '__builtins__', '__doc__', '__file__', '__name__', '__package__', '_acos', '_c
eil', '_cos', '_e', '_exp', '_hashlib', '_hexlify', '_inst', '_log', '_pi', '_ra
ndom', '_sin', '_sqrt', '_test', '_test_generator', '_urandom', '_warn', 'betava
riate', 'choice', 'division', 'expovariate', 'gammavariate', 'gauss', 'getrandbi
ts', 'getstate', 'jumpahead', 'lognormvariate', 'normalvariate', 'paretovariate'
, 'randint', 'random', 'randrange', 'sample', 'seed', 'setstate', 'shuffle', 'tr
iangular', 'uniform', 'vonmisesvariate', 'weibullvariate']
```

ex:

```
import random

print(random.randint(1,100))
print(random.choice( ['red', 'black', 'green']))

myList = [2, 10.5, False,"ratan","anu"]
print(random.choice(myList))

for i in range(3):
   print random.randrange(0, 101, 5)
```

ex:

```
from random import randint
print(randint(1,100))
```

time module :

Example *:*

```
import time
content = dir(time)
print (content)
```

output :

```
E:\data>python first.py
['__doc__', '__name__', '__package__', 'accept2dyear', 'altzone', 'asctime', 'cl
ock', 'ctime', 'daylight', 'gmtime', 'localtime', 'mktime', 'sleep', 'strftime',
 'strptime', 'struct_time', 'time', 'timezone', 'tzname']
```

Example :

```
import time
print("hi sir")
time.sleep(1)
print(time.strftime('%X %x %Z'))
print("hi sir")
```

Directive	Meaning	Notes
%a	Locale's abbreviated weekday name.	
%A	Locale's full weekday name.	
%b	Locale's abbreviated month name.	
%B	Locale's full month name.	
%c	Locale's appropriate date and time representation.	
%d	Day of the month as a decimal number [01,31].	
%H	Hour (24-hour clock) as a decimal number [00,23].	
%I	Hour (12-hour clock) as a decimal number [01,12].	
%j	Day of the year as a decimal number [001,366].	
%m	Month as a decimal number [01,12].	
%M	Minute as a decimal number [00,59].	
%p	Locale's equivalent of either AM or PM.	(1)
%S	Second as a decimal number [00,61].	(2)
%U	Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0.	(3)
%w	Weekday as a decimal number [0(Sunday),6].	
%W	Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0.	(3)
%x	Locale's appropriate date representation.	
%X	Locale's appropriate time representation.	
%y	Year without century as a decimal number [00,99].	
%Y	Year with century as a decimal number.	
%Z	Time zone name (no characters if no time zone exists).	
%%	A literal '%' character.	

sys module :

```
import sys
content = dir(sys)
print (content)
```

```
E:\data>python first.py
['__displayhook__', '__doc__', '__excepthook__', '__name__', '__package__', '__stderr__',
'__stdin__', '__stdout__', '_clear_type_cache', '_current_frames', '_getframe', '_git', 'api_version',
'argv', 'builtin_module_names', 'byteorder', 'call_tracing', 'callstats', 'copyright', 'displayhook',
'dllhandle', 'dont_write_bytecode', 'exc_clear', 'exc_info', 'exc_type', 'excepthook', 'exec_prefix',
'executable', 'exit', 'flags', 'float_info', 'float_repr_style', 'getcheckinterval', 'getdefaultencoding',
'getfilesystemencoding', 'getprofile', 'getrecursionlimit', 'getrefcount', 'getsizeof', 'gettrace',
'getwindowsversion', 'hexversion', 'long_info', 'maxint', 'maxsize', 'maxunicode', 'meta_path',
'modules', 'path', 'path_hooks', 'path_importer_cache', 'platform', 'prefix', 'py3kwarning',
'setcheckinterval', 'setprofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout', 'subversion',
'version', 'version_info', 'warnoptions', 'winver']
```

Regular Expression

- ✓ *The term "regular expression", sometimes also called regex or regexp for string pattern matching.*
- ✓ *I n python the regular expression are available in **re** module*

The most common uses of regular expressions are:

Search a string (search and match)
Finding a string (findall)
Break string into a sub strings (split)
Replace part of a string (sub)

Methods in Regular Expression :

re.match()
re.search()
re.findall()
re.split()
re.sub()
re.compile()

ex:

This method finds match if it occurs at start of the string
If the match is not available it return none.
re.match(pattern, string):

```
import re
result = re.match(r'ratan', 'ratan sir good')
print (result)
print (result.group(0))
print (result.start())
print (result.end())

result = re.match(r'durga', 'hi sir')
print (result)  #None
```

ex:
It is similar to match() but it doesn't restrict us to find matches at the beginning of the string only.
re.search(pattern, string):

```
import re
result = re.search(r'ratan', 'hi ratan sir')
print(result)
print(result.group(0))
print(result.start())
print(result.end())

result = re.search(r'durga', 'hi ratan sir')
print(result)
```

Ex:

```
import re
print(re.search("ratan","hi welcome to ratanit"))
print(re.search("ratan","hi welcome to hyderabad"))
```

output :

```
<_sre.SRE_Match object; span=(14, 19), match='ratan'>
None
```

Ex:

```
import re
print(re.search(r"ratan","hi welcome to ratanit"))
result= re.search(r"ratan","hi welcome to ratanit")
print(result.group(0))
print(result.start())
print(result.end())
print(re.search("ratan","hi welcome to hyderabad"))
```

Ex:

```
import re
if re.search("cat","A cat and a rat can't be friends."):
    print ("Some kind of cat has been found :")
else:
    print ("No cat has been found :")
```

ex:

- ✓ *It helps to get a list of all matching patterns. It has no constraints of searching from start or end*
- ✓ *use re.findall() always, it can work like re.search() and re.match() both.*

re.findall (pattern, string):

```
import re
result = re.findall(r'ratan', 'hi ratan sir ratanit')
print(result)
print(tuple(result))
print(set(result))

result = re.findall(r'durga', 'hi ratan sir')
print(result)
```

ex: This methods helps to split string by the occurrences of given pattern.

re.split(pattern, string, [maxsplit=0]):

Method split() has another argument "maxsplit". It has default value of zero.

```
import re
result=re.split(r'a','ratanit')
print(result)    #['r', 't', 'nit']
```

ex:

```
import re
result=re.split(r'a','ratanitratanit',maxsplit=2)
print(result)    #['r', 't', 'nitratanit']
```

ex:

It helps to search a pattern and replace with a new sub string. If the pattern is not found, string is returned unchanged.

re.sub(pattern, repl, string):

```
import re
result=re.sub(r'durga',r'ratan','durga world durga in India')
print(result)

result=re.sub(r'sunny',r'ratan','durga world durga in India')
print(result)
```

ex:

We can combine a regular expression pattern into pattern objects, which can be used for pattern matching. It also helps to search a pattern again without rewriting it.

```
import re
pattern=re.compile('ratan')

result=pattern.findall('hi ratan sir welcome to ratanit')
print (result)

result2=pattern.findall('ratanit is good')
print (result2)

result3=pattern.findall('durgasoft is good')
print (result3)
```

ex:

```
import re
result=re.findall(r'.','hi welcome to ratan it')
print (result)
```

ex: finding the index of the string

```
import re
s = "hi i like beer and beer is good"

for i in re.finditer("beer",s):
  x = i.span()
  print(x)
```

ex:

\d *Matches with digits [0-9]*

[..] *Matches any single character in a square bracket*

```
import re
nameage="Balu age is 40 Chiru age is 65"

age=re.findall(r"\d{1,3}",nameage)
names = re.findall(r"[A-Z][a-z]*",nameage)

my_dict={}

x=0
```

```
for name in names:
  my_dict[name]=age[x]
  x=x+1

print(my_dict)
```

regularexpressions

Anchors

^	Start of string
\A	Start of string
$	End of string
\Z	End of string
\b	Word boundary
\B	Not word boundary
\<	Start of word
\>	End of word

Character Classes

\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\x	Hexadecimal digit
\O	Octal digit

POSIX

[:upper:]	Upper case letters
[:lower:]	Lower case letters
[:alpha:]	All letters
[:alnum:]	Digits and letters
[:digit:]	Digits
[:xdigit:]	Hexadecimal digits
[:punct:]	Punctuation
[:blank:]	Space and tab
[:space:]	Blank characters
[:cntrl:]	Control characters
[:graph:]	Printed characters
[:print:]	Printed characters and spaces
[:word:]	Digits, letters and underscore

Assertions

?=	Lookahead assertion
?!	Negative lookahead
?<=	Lookbehind assertion
?!= or ?<!	Negative lookbehind
?>	Once-only Subexpression
?()	Condition [if then]
?()|	Condition [if then else]
?#	Comment

Quantifiers

*	0 or more
+	1 or more
?	0 or 1
{3}	Exactly 3
{3,}	3 or more
{3,5}	3, 4 or 5

Quantifier Modifiers

"x" below represents a quantifier

x?	Ungreedy version of "x"

Escape Character

\	Escape Character

Metacharacters (must be escaped)

^	[	.
$	{	*
(	\	+
)	|	?
<	>	

Special Characters

\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	Form feed
\xxx	Octal character xxx
\xhh	Hex character hh

Groups and Ranges

.	Any character except new line (\n)
(a|b)	a or b
(...)	Group
(?:...)	Passive Group
[abc]	Range (a or b or c)
[^abc]	Not a or b or c
[a-q]	Letter between a and q
[A-Q]	Upper case letter between A and Q
[0-7]	Digit between 0 and 7
\n	nth group/subpattern

Note: Ranges are inclusive.

Pattern Modifiers

g	Global match
i	Case-insensitive
m	Multiple lines
s	Treat string as single line
x	Allow comments and white space in pattern
e	Evaluate replacement
U	Ungreedy pattern

String Replacement (Backreferences)

$n	nth non-passive group
$2	"xyz" in /^(abc(xyz))$/
$1	"xyz" in /^(?:abc)(xyz)$/
$`	Before matched string
$'	After matched string
$+	Last matched string
$&	Entire matched string

Sample Patterns

Pattern	*Will Match*
([A-Za-z0-9-]+)	Letters, numbers and hyphens
(\d{1,2}\/\d{1,2}\/\d{4})	Date (e.g. 21/3/2006)
([^\s]+(?=\.(jpg|gif|png))\.\2)	jpg, gif or png image
(^[1-9]{1}$\|^[1-4]{1}[0-9]{1}$|^50$)	Any number from 1 to 50 inclusive
(#?([A-Fa-f0-9]){3}(([A-Fa-f0-9]){3})?)	Valid hexadecimal colour code
((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,15})	String with at least one upper case letter, one lower case letter, and one digit (useful for passwords).
(\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})	Email addresses
(\<(/?[^\>]+)\>)	HTML Tags

Note: These patterns are intended for reference purposes and have not been extensively tested. Please use with caution and test thoroughly before use.

Ex:

```
import re
s = "rat mat bat cat durga"

x = re.findall("[rmbc]at",s)

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

ex:

```
import re
s = "rat mat bat cat durga"

x = re.findall("[rmbc]at",s)
y = re.findall("[a-f]at",s)
z = re.findall("[A-Z]at",s)
print(x)
print(y)
print(z)

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

ex:

```
import re
s = "rat mat bat cat"

x = re.findall("[a-k]at",s)

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

ex :

```
import re
s = "rat mat bat cat"

x = re.findall("[^a-k]at",s)

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

abc… *Letters*
123… *Digits*

\d	*Any Digit*
\D	*Any Non-digit character*
.	*Any Character*
\.	*Period*
[abc]	*Only a, b, or c*
[^abc]	*Not a, b, nor c*
[a-z]	*Characters a to z*
[0-9]	*Numbers 0 to 9*
\w	*Any Alphanumeric character*
\W	*Any Non-alphanumeric character*
{m}	*m Repetitions*
{m,n}	*m to n Repetitions*
***	*Zero or more repetitions*
+	*One or more repetitions*
?	*Optional character*
\s	*Any Whitespace*
\S	*Any Non-whitespace character*
^…$	*Starts and ends*

(…) Capture Group
(a(bc)) Capture Sub-group
(.) Capture all*
(abc|def) Matches abc or def

Ex:

```
import re

s=" hi sir \n please stop \n the class"
print(s)

r = re.compile("\n")
x = r.sub(" ",s)
print(x)
```

ex:

```
import re

num = "12345"
print("Matches : ",len(re.findall("\d",num)))
print("Matches : ",len(re.findall("\D",num)))
```

ex :

```
import re
```

```
num = "12345 6789 657 78  909090"
print(len(re.findall("\d{5}",num)))
print(len(re.findall("\d{4,7}",num)))
```

MultiThreading

ex:

```
# importing the threading module
import threading

def print_cube(num):
   print("Cube: {}".format(num * num * num))

def print_square(num):
   print("Square: {}".format(num * num))

# creating thread
t1 = threading.Thread(target=print_square, args=(10,))
t2 = threading.Thread(target=print_cube, args=(10,))
t1.start()  # starting thread 1
t2.start()  # starting thread 2
```

ex:

```
from threading import Thread
def print_cube(num):
   print("Cube: {}".format(num * num * num))
t1 = Thread(target=print_cube,args=(4,))
t1.start()
```

ex:

```
import threading
import time
def f1():
   print("thread starting",threading.currentThread().getName())
   time.sleep(1)
   print("thread-1 ending")

def f2():
   print("thread starting",threading.currentThread().getName())
   time.sleep(1)
```

```
   print("thread-2 ending")

def f3():
  print("thread starting",threading.currentThread().getName())
  time.sleep(1)
  print("thread-3 ending")

t1 = threading.Thread(target=f1, name="t1")
t2 = threading.Thread(target=f2,name="t2")
t3 = threading.Thread(target=f3,name="t3")
t1.start()
t2.start()
t3.start()
```

ex:

```
from threading import Thread
import threading
from time import sleep
def f1():
   print("thread starting",threading.currentThread().getName())
   sleep(1)
   print("thread-1 ending")

t1 = Thread(target=f1, name="t1")
t1.start()
```

ex:

```
import threading
import time
def f1():
   print("hi ratan sir staeted")
   time.sleep(3)
   print(threading.currentThread())
   print("hi ratan sir completed")

t1 = threading.Thread(target=f1, name="t1")
t2 = threading.Thread(target=f1,name="t2")
t3 = threading.Thread(target=f1,name="t3")
t1.start()
t2.start()
t3.start()
```

ex:

```
import threading
import time
def f1():
   print("hi ratan sir staeted")
   time.sleep(3)
```

```
        print(threading.currentThread())
        print("hi ratan sir completed")

    for i in range(1,4):
      t1 = threading.Thread(target=f1, name="t1")
      t1.start()
```

ex:

```
# importing the threading module
import threading
import time

def thread1():
   for i in range(1,10):
     print("Thread-1 running")
     time.sleep(1)

def thread2():
   for i in range(1,10):
     print("Thread-2 running")
     time.sleep(1)

# creating thread
t1 = threading.Thread(target=thread1)
t2 = threading.Thread(target=thread2)
t1.start()  # starting thread 1
t2.start()  # starting thread 2
```

case : Observation

```
# creating thread
t1 = threading.Thread(target=thread1)
t2 = threading.Thread(target=thread2)
t1.start()  # starting thread 1
t1.join()    # t1.join(2)
t2.start()  # starting thread 2
```

Ex:

```
# importing the threading module
import threading
import time

def thread1():
   for i in range(1,10):
     print("Thread-1 running")
```

```
    time.sleep(1)

def thread2():
   for i in range(1,10):
     print("Thread-2 running")
     time.sleep(1)

# creating thread
t1 = threading.Thread(target=thread1)
t2 = threading.Thread(target=thread2)
t1.start()  # starting thread 1
t1.join()
t2.start()  # starting thread 2
t2.join()
print("Rest of the app")

case : Observation
# creating thread
t1 = threading.Thread(target=thread1)
t2 = threading.Thread(target=thread2)
t1.start()  # starting thread 1
t2.start()  # starting thread 2
t1.join()
t2.join()
print("Rest of the app")
```

ex:

```
import threading

# creating thread
t1 = threading.Thread()
print(t1)
t1.start()
print(t1)
print(t1)
```

```
E:\>python first.py
<Thread(Thread-1, initial)>
<Thread(Thread-1, started 5776)>
<Thread(Thread-1, stopped 5776)>
```

ex:

```
import threading

# creating thread
t1 = threading.Thread()
```

```
t1.start()
t1.start()

E:\>python first.py
raise RuntimeError("threads can only be started once")
RuntimeError: threads can only be started once
```

It is highly recommended to store complete application flow and exceptions information to a file. This process is called logging.
The main advantages of logging are:
Depending on type of information,logging data is divided according to the following 6 levels in Python.
table
1. CRITICAL==>50==>Represents a very serious problem that needs high attention
2. ERROR===>40===>Represents a serious error
3. WARNING==>30==>Represents a warning message ,some caution needed.it is alert to the programmer
4. INFO===>20===>Represents a message with some important information
5. DEBUG===>10==>Represents a message with debugging

To perform logging,first we required to create a file to store messages and we have to specify which level messages we have to store.
We can do this by using basicConfig() function of logging module.

```
logging.basicConfig(filename='log.txt',level=logging.WARNING)
```

The above line will create a file log.txt and we can store either WARNING level or higher level messages to that file.

After creating log file,we can write messages to that file by using the following methods.
```
logging.debug(message)
logging.info(message)
logging.warning(message)
logging.error(message)
logging.critical(message)
```

ex:

```
import logging
logging.basicConfig(filename='log.txt',level=logging.WARNING)
print("Logging Module Demo")
logging.debug("This is debug message")
logging.info("This is info message")
logging.warning("This is warning message")
logging.error("This is error message")
logging.critical("This is critical message")
```

Note:
In the above program only WARNING and higher level messages will be written to log file.
If we set level as DEBUG then all messages will be written to log file.

Ex:

```
import logging
logging.basicConfig(filename='ratan.txt',level=logging.INFO)
logging.info("Request processing started:")
try:
    x=int(input("Enter First Number: "))
    y=int(input("Enter Second Number: "))
    print(x/y)
except ZeroDivisionError as msg:
    print("ZeroDivisionError occurred")
    logging.exception(msg)
except ValueError as msg:
    print("Enter only integer values")
```

```
            logging.exception(msg)
        logging.info("Request processing completed:")
```

ex:

```
def squareIt(x):
    return x**x

assert squareIt(2)==4,"The square of 2 should be 4"
assert squareIt(3)==9,"The square of 3 should be 9"
assert squareIt(4)==16,"The square of 4 should be 16"
print(squareIt(2))
print(squareIt(3))
print(squareIt(4))
```

ex:

```
def squareIt(x):
    return x*x
assert squareIt(2)==4,"The square of 2 should be 4"
assert squareIt(3)==9,"The square of 3 should be 9"
assert squareIt(4)==16,"The square of 4 should be 16"
print(squareIt(2))
print(squareIt(3))
print(squareIt(4))
```

Lambda expressions

- ✓ *lambda function will simply the expressions, reduce the length of the code.*
- ✓ *Lambda functions are used along with built-in functions like filter(), map() etc.*

ex:

```
def f (x):
  print( x**2)

g = lambda x: x**2
data  = g(3)
print(data)

print(g(4))
```

ex:

```
s=lambda n:n*n
print("The Square of 4 is :",s(4))
print("The Square of 5 is :",s(5))

s=lambda a,b:a+b
print("Addition:",s(10,20))

s=lambda a,b:a if a>b else b
print("The Biggest of 10,20 is:",s(10,20))
```

```
print("The Biggest of 100,200 is:",s(100,200))


ex:
# Program to filter out only the even items from a list

my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(filter(lambda x: (x%2 == 0) , my_list))

# Output: [4, 6, 8, 12]
print(new_list)
```

ex:
We can use filter() function to filter values from the given sequence based on some condition.
filter(function,sequence)
where function argument is responsible to perform conditional check
sequence can be list or tuple or string.

```
Ex:
def isEven(x):
    if x%2==0:
        return True
    else:
        return False

l=[0,5,10,15,20,25,30]
l1=list(filter(isEven,l))
print(l1) #[0,10,20,30]

t=tuple(filter(isEven,l))
print(t) #[0,10,20,30]

ex:
l=[0,5,10,15,20,25,30]
l1=list(filter(lambda x:x%2==0,l))
print(l1)

t=list(filter(lambda x:x%2==0,l))
print(t)


ex:
l1 = ["ratan","anu","durga"]
```

```
def m1(x):
    if "ratan" in x:
        return True
    else:
        return False
print(list(filter(m1,l1)))

print(list(filter(lambda x:"ratan" in x,l1)))

ex:
def m1(x):
    if(x%2==0):
        return True
    else:
        return False

my_list = [1, 5, 4, 6, 8, 11, 3, 12]
l1 = list(filter(m1,my_list))
print(l1)

ex:
def disp(x):
    if "ratan" in x:
        return 1
    else:
        return 0

l=["ratan","anu","durga"]
l1=list(filter(disp,l))
print(l1)

t=tuple(filter(disp,l))
print(t)

ex:
l=["ratan","anu","durga"]
l1=list(filter(lambda x :"ratan" in x,l))
print(l1)

t=tuple(filter(lambda x :"ratan" in x,l))
print(t)
```

Mapper : we can do some modifications

```
ex:
l=[2,4,6,8]
def doubleIt(x):
 return 3*x

l1=list(map(doubleIt,l))
print(l1)

t=tuple(map(doubleIt,l))
print(t)

ex:
l1 = [1,2,3,4]

def m1(x):
      return x**2

x1 = list(map(m1,l1))
print(x1)

print(list(map(lambda a:a**2,l1)))

ex:
l=[2,4,6,8]

l2 = list(map(lambda a:a*2,l))
print(l1)

l2 = tuple(map(lambda a:a*2,l))
print(l1)

Ex:
l=["ratan","anu"]
l2 = list(map(lambda a:a+"it",l))
print(l2)

t = tuple(map(lambda a:a+"it",l))
print(t)

ex:
sentence = 'It is raining cats and dogs'
words = sentence.split()
print (words)  # ['It', 'is', 'raining', 'cats', 'and', 'dogs']
```

```
lengths = map(lambda word: len(word), words)
print (lengths)  # [2, 2, 7, 4, 3, 4]


ex:
print (map(lambda w: len(w), 'It is raining cats and dogs'.split()))

ex:
l1=[1,2,3,4]
def disp(x):
    return x*2
l  = list(map(disp,l1))
print(l)
t  = tuple(map(disp,l1))
print(t)

#application with lambda
l1=[1,2,3,4]
l  = list(map(lambda x:x*2,l1))
print(l)
t  = tuple(map(lambda x:x*2,l1))
print(t)
```

Zip File creation

Ex:

```
import zipfile
zf = zipfile.ZipFile('ratan.zip', mode='w')
try:
        zf.write('ratan.txt')
        zf.write('log.txt')
finally:
   zf.close()
print("zip file is ready")
```

ex:

```
import zipfile

for filename in [ 'ratan.txt', 'ratan.zip','example.zip' ]:
   print ('%s  %s' % (filename, zipfile.is_zipfile(filename)))
```

```
E:\>python first.py
ratan.txt  False
ratan.zip  True
example.zip  False
```

ex: