Python Digital Content

Module-01(core python)

Introduction:

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 at the Centrum Wiskande & Informatica(CWI) and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

There are two major Python versions: Python 2 and Python 3. Both are quite different.

Identfiers:

- A name in the python program is called as identifier.
- It can be a class name or function or module name or variable name.
- Ex: a=10
- def(): class test:

Rules to define identifiers in python:

- If identifier starts with underscore (_) then it indicates it is private.
- We can't use reserved words as identifiers Ex:
  def=10
- There is no length limit for python identifiers. But not recommended to use too lengthy identifiers.
- Dollar ($) symbol is not allowed in python.
- If identifier starts with (_ _)it indicates that strongly private identifiers.
- If the identifier starts and ends with two underscores symbols then the identifier is language defined special name, which is also o known as magic methods.
- Ex: __add__

Variables in python:

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

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Rules to define a variable:

1. Variable must start with either alphabet or _
2. Variable can't start with number
3. We can write alphanumeric value to the variable
4. We can't use special characters in name
5. We can only _ special character in variable name
6. No space is allowed in variable name

1).The only allowed characters in python are

- Alphabets(Either lower case or upper case)
- Digits(0-9)
- Underscore symbol(_)

Reserved Words:

->In python some words are reserved to represent some meaning or functionality. Such type of words are called as reserved words.
-->There are 35+ reserved words in python

```
>>> import keyword
    Keyword.kwlist
```

Note:

-->All the reserved words in python contain only alphabets symbols.
-->Except the following 3 reserved words, all contain only lower case alphabets.
True, False, None

```
Ex:     a=true(invalid)
        a=True(valid)
```

Data Types:

--> Data types are used to specify what type of data has to be store in the variable.
--> Without specifying data types variable memory allocation with not take place for variables.
--> Programming languages supports 2-types of data types.
1).Static 2).Dynamic

1).Static Data Types:

-->In static type programming languages programmer should define the data type to the variable explicitly.
-->Once if we define data type to the variable explicitly, we can't modify the data type of that variable throughout the program execution
→C, C++, .Net, Java.......languages supports static data types.

2).Dynamic Data Types:

-->In dynamic data type we not need to specify data type , it will be define automatically based on user input.
-->Python data types are categorized into 2-parts.
-->Fundamental types
-->Collection types

Python supports three types of numeric data.

Int **-** Integer value can be any length such as integers 10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer. Its value belongs to int

Float **-** Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.

complex **-** A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

Bool - it will store True and False values

Note:

Python contains several inbuilt functions

1) type():
To check the type of variable.
2) id():
To get the address of object.
3) print():
To print the value

Str Data Type:

--> Str represents String data type.
--> A string is a sequence of characters with in single quotes or double quotes. Ex: s='PRSOFTWARES'
s="PRSOFTWARES"
--> By using single quotes or double quotes we can't represent multi line string literals. Ex: s="PRSOFTWARES
Dilsukhnagar"(Invalid)
--> For this requirement we should go for triple quotes(''') or triple double
--> We can embeded one string in another string.
Ex: '''This "python classes very helpful" for java students'''.

Slicing of Strings:

--> Slice means a piece.
--> [:] is called as slice operator, which can be used to retrieve parts of string.
--> In python string follows zero based index.
--> The index can be either +ve or -ve.
--> +ve index means forward direction from Left to Right.
--> -ve index means backward direction from Right to Left

--> long data type is available in python2 but not in python3. In python3 long values also we can represent by using int type only.

Type Casting:

-->We can convert one type to other type. This conversion is called as Typecasting or Type coercion.
-->The following are various in-built functions for type casting.
--> int()
--> float()
--> bool()
--> str()
--> bytes()

1) int():

We can use this function to convert values from other types to int.

```
>>> int(123.456)
123
>>> int(10+4j)
TypeError: can't convert complex to int
>>> int(True)
1
>>> int(False)
0
>>> int("10")
10
>>> int("10.5")
ValueError: invalid literal for int() with base 10: '10.5'
>>> int("ten")
ValueError: invalid literal for int() with base 10: 'ten'
>>> int("0B1010")
ValueError: invalid literal for int() with base 10: '0B1010'
```

Note:

-->We can convert from any type to int except complex type.
-->If we want to convert str type to int type, compalsury str should contains only integral value and should be specified in base-10.

2) float():

We can use float() to convert other type values to float type.

```
>>> float(10)
10.0
>>> float(10+4j)
TypeError: can't convert complex to float
>>> float(True)
1.0
>>> float(False)
0.0
>>> float("10")
10.0
>>> float("10.5")
10.5
>>>float("ten")
ValueError: could not convert string to float: 'ten'
```

Note:

--> We can convert any type value to float type except complex type.
--> Whenever we are trying to convert str type to float type compulsory str should be either integral or floating point literal and should be specified only in base-10.

3) bool():

We can use this function to convert other type to bool type

```
>>> bool(0)
False
>>> bool(1)
True
>>> bool(10)
True
>>> bool(0.123)
True
```

4) str():

We can use this method to convert other type values to str type.

```
>>> str(10)
'10'
>>> str(10.5)
'10.5'
>>> str(True)
'True'
>>> str(10+4j)
```

```
'(10+4j)'
```

5) Bytes Data Type:

Bytes data type represents a group of byte numbers

just like an array. Ex:

```
>>> x=[10,20,30,40]
>>> b=bytes(x)
>>> type(b)
<class 'bytes'>
>>> b[0]
10
>>> b[-1]
```

Conclusion 1:

The only allowed values for byte data type are 0 to 256. BY mistakes if we are trying to provide any other values then we will get value error.

```
Ex:   >>> x=[10,20,300,40]
      >>> b=bytes(x)
ValueError: bytes must be in range (0, 256)
```

Conclusion 2:

Once we create bytes data type value, we can't change it values, if we are trying to change we will get an error.

```
Ex:   >>> x=[10,20,30,40]
>>> b=bytes(x)
>>> b[2]=50

TypeError: 'bytes' object does not support item assignment
```

-->Any person who is doing certain activity is called as an operator.
-->Operator is a symbol that performs certain operations.
-->Python provides the following set of operators.

Operators:

Python Operators
Operators are used to perform operations on variables and values.
Python divides the operators in the following groups:

--> Arithmetic operators
--> Assignment operators
--> Comparison operators
--> Logical operators
--> Identity operators
--> Membership operators
--> Bitwise operators

Python Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20, then

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power)	a**b =10

	calculation on operators	to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) −	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Python Assignment Operators:

Assume variable a holds 10 and variable b holds 20, then

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

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

Python Comparison Operators:

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

Assume variable a holds 10 and variable b holds 20, then −

Operator	**Description**	**Example**
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not	(a <> b)

	equal, then condition becomes true.	is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Logical Operators:

There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

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

Python Identity Operators:

Identity operators compare the memory locations of two objects. There are two Identity operators explained below −

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

Python Membership Operators:

Python’s membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below −

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and	x not in y, here not

	false otherwise.	in results in a 1 if x is not a member of sequence y.

Python Bitwise Operators:

Bitwise operator works on bits and performs bit by bit operation.
Assume if a = 60; and b = 13; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively.
Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands −
a = 0011 1100
b = 0000 1101
----
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a = 1100 0011

There are following Bitwise operators supported by Python language

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)

^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

input():

Input () function can be used to read data directly in our required format. We are not required to perform type casting.
Ex:

1) x=input(Enter value")
2) type(x)
3) 10==>int
4) "prsoftwares"==>str
5) 10.5==>float
6) True==>bool

Note:

--> But in python-3 we have only input () method and raw_input() method is not available.
--> Python-3 input () function behavior exactly same as raw_input() method in python-2. I.e. every input value is treated as str type only.
--> Raw_input() function of python-2 is renamed as input() function in python-3.

Ex:
1) >>> x=input ("Enter some value :")
2) Enter some value: 10
3) <Class 'str'>
4) Enter some value: True

Q: w.a.p to read employee data from the keyboard and print that data.

```
eno=int(input("Enter Employee No: "))
ename=input("Enter Employee Name: ")
esal=float(input("Enter Employee Salary: "))
eadd=input("Enter Employee Address: ")
emarried=bool(input("Enter Employee Married?[True | False]: "))
print("Confirm Information")
print("Employee No :",eno)
print("Employee Name :",ename)
print("Employee Salary :",esal)
print("Employee Address :",eadd)
print("Employee Married? :",emarried)
```

Q: w.a.p to read multiple values from the keyboard in a single line:

```
a,b=[int (x) for x in input("Enter two numbers:").split()]
print(a,b)
print("Product is :", a*b)
```

Q: w.a.p to read 3 float numbers from the keyboard with , separator and print their sum

```
a,b,c=[float(x) for x in input("Enter 3 float numbers :").split(",")]
print("The sum is :",a+b+c)
```

Note:

split() function can take space as separator by default, But we can pass anything as separator.

Q:How to read different types of data from the keyboard.

```
a,b,c=[eval(x) for x in input("Enter 3 values :").split(",")]
print(type(a))
print(type(b))
print(type(c))
o/p:Enter 3 values :10,True,10.3
<class 'int'>
<class 'bool'>
<class 'float'>
```

eval():

-->eval function take a string and evaluate the result.

```
>>> x=eval("10+20+30")
>>>x
>>>60

x=eval(input("Enter expression :"))
print(x)
Enter expression :10+2*3/4
11.5
```

-->eval() can evaluate the input to list, tuple, set etc...Based on provided input.

Ex: w.a.p to accept list from the keyboard and display.

```
l=eval(input("Enter list :"))
print(type(l))
print(l)
```

o/p: Enter list :[10,20,30]

```
<class 'list'> [10, 20, 30]
```

Q:How to read different types of data from the keyboard

```
a,b,c=[eval(x) for x in input("Enter 3 values :").split(",")]
print(type(a))
print(type(b))
print(type(c))
o/p:Enter 3 values :10,True,10.3
<class 'int'>
<class 'bool'>
<class 'float'>
```

Note:

-->split() always accept sting only as a separator.
-->If we are not using eval() then it is treated as str.

Ex:

```
a,b,c=[eval(x) for x in input("Enter 3 values :").split("5")]
print(type(a))
print(type(b))
print(type(c))
```

o/p:
```
Enter 3 values :105True510.3
<class 'int'>
<class 'bool'>
<class 'float'>
```

Output Streams:

We can use print() function to display output.

Form-1: print() without argument
just it prints new line character.

Form-2: print(String)
print("Hello world")-->We can use escape characters also

Ex: print(“prsoftwares”)

print("Hello \n world") print("Hello \t world")
-->We can use repetetion operator(*) in the string

print(10*"hello") print("hello"*10)

-->We can use + operator also print("Hello" + "World")

Note:

--> If both arguments are string type then + operator acts as concatenation operator.
--> If one argument is string type and second is any other type like int then we will get an error.
--> If both arguments are number type then + operator avts as arithmetic addition operator.

Q: what is diff between
print("Hello"+"world")==>Helloworld
print("Hello","world")==>Hello word

Form-3: print() with variable number of arguments
Ex: 1) a,b,c=10,20,30

O/p: The values are: 10 20 30
-->By default output values are separated by space. If we want we can specify separator by using "sep" attribute.

Ex:
1) a,b,c=10,20,30
2) print(a,b,c,sep=',')
3) print(a,b,c,sep=':')

o/p: D:\pythonclasses>py test.py 10,20,30
10:20:30

Form-4:print() with end attribute

1) print("Hello")
2) print("PR")
3) print("SoftWARES")

o/p: Hello
PR SoftWARES

-->If we want output in the same line with space

1) print("Hello",end=' ')
2) print("PR",end=' ')
3) print("Soft")

o/p: D:\pythonclasses>py test.py Hello PR Soft

Form-5:print(object) statement
-->We can pass any object(like list, tuple, set etc) as argument to the print statement.

Ex:
1) l=[10,20,30,40]
2) t=(10,20,30,40)
3) print(l)
4) print(t)

Form-6:print(String, variable list):
-->We can use print() statement with string and any number of arguments.
Ex:
1) s="PRSOFTWARES"
2) a=50
3) s1="PROGRAMMING"
4) s2="Python"
5) print("Hello",s,"Your age is :",a)
6) print("You are teaching", s1,"and", s2)

o/p: Hello PRSOFTWARES Your age is : 50
You are teaching PROGRAMMING and Python

Form-7:print(formatted string):
%d===>int
%f===>float
%s===>string type

syn: print("formatted string" %(variable list)) Ex:

1) a=10
2) b=20
3) c=30
4) print("a value is %d" %a)
5) print("b value is %d and c value is %d" %(b,c))

Form-8:print() with replacement operator { }

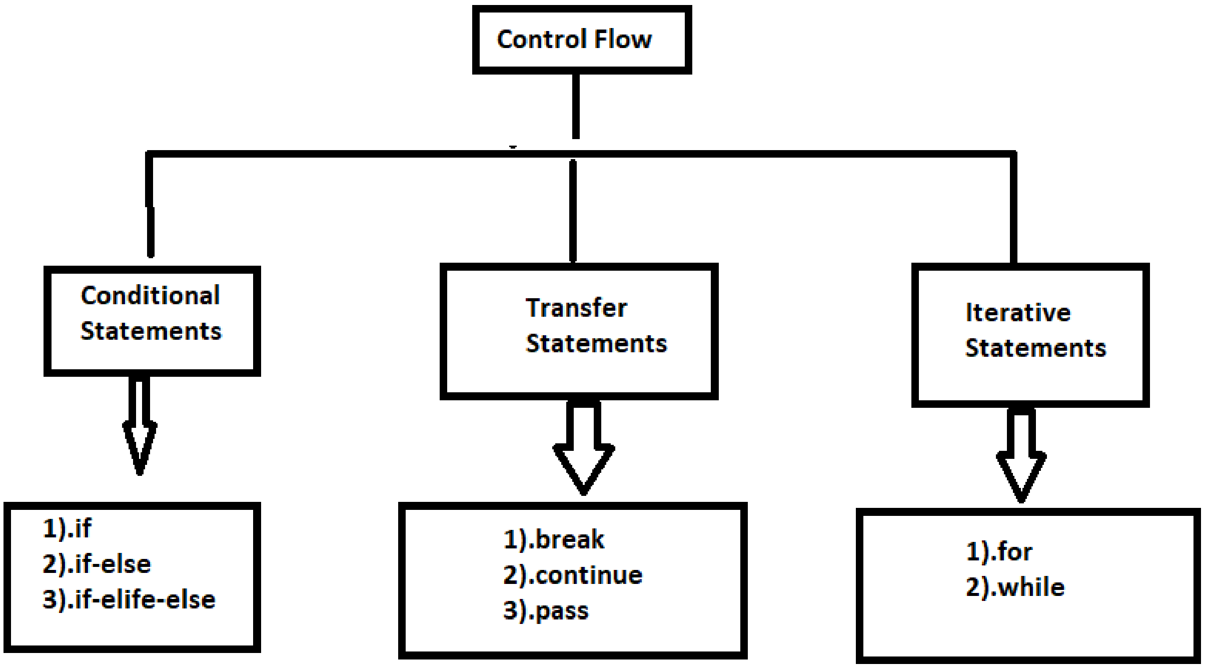
Ex:

1) name="Prsoftwares"
2) salary=10000
3) company="premsofts"
4) print("Hello {0} your salary {1} and your company is {2}
".format(name,salar
y,company))
5) print("Hello {} your salary {} and your girlfriend {} is
waiting".format(name,salary,compnay))
6) print("Hello {x} your salary {y} and your girlfriend {z} is
waiting".format(z=gf,x=nam
e,y=salary))

o/p: Hello Prsoftwares your salary 10000 and yor girlfriend premsofts is waiting Hello Prsoftwares your salary 10000 and your girlfriend premsofts is waiting Hello Prsoftwares your salary 10000 and your girlfriend premsofts is waiting

FLOW CONTROL

-->Flow control describes the order in which statements will be



executed at runtime.

Conditional Statements:

1).if:

if condition: statement
or
if condition: statement-1 statement-1 statement-1
--> If condition is true then statements will be executed.

2).if-else:

if condition:
Action-1 else:
Action-2

-->If condition is true Action-1 will be executed otherwise Action-2 will be executed

3).**if-elif-else: syn:**

if condition1:
Action-1
elif condition-2:
Action-2
elif condition-3:
Action-3 else
   Action-3

String data type:

Removing spaces from the string:

-->We can use the following 3-methods.
 1).rstrip()==>To remove spaces at right hand side.
2).lstrip()==>To remove spaces at left hand side.
3).strip()==>To remove spaces both sides

Counting substring the given string:

-->We can find the number of occurances of substring present in the given string by using count() method.
1).s.count(substring)==>It will search through out the string
2).s.count(substring,begin,end)==>It will search from begin index to end-1 index.

Replacing a string with another string:

Syntax:

```
   S=”prsoftwares”
    s.replace(oldstring,newstring)
    ex-s.replace(“s”,”ss”)
```

--> Inside s, every occurance of old string will be replaced with new string.

Splitting of string:

-->We can split the given string according to specified separator by using split() method. Syn: l=s.split(separator)
-->The default separator is space. The return type of split() method is list.

Ex:

a="python training institute"
a.split(" ")

Joining of Strings:

-->We can join a group of strings (list or tuple) wrt the given separator. Syn: s=separator.join(group of strings)

Ex:

```
1)    t=('prsoftwares','premsofts','belongs to Mr.Prem')
2)    s='-'.join(t)
3)    print(s)
```

Changing Case of a string:

-->We can change case of a string by using the following 5-methods.
1).**upper ():** To convert all characters to upper case
2).**lower ():** To convert all characters to lower case
3)**.swap case():** Converts all lower case characters to upper case and all upper case characters to lower case.
4)**.title ():** To convert all the characters to title case. i.e first character in every word should be upper case and all remaining characters should be lower case.
5)**.capitalize():** Only first character will be converted to upper case and all remaining characters can be converted to lower case.

Checking Starting and ending part of the string:

-->Python contains the following methods for this purpose.
1).s.startswith(substring)
2).s.endswith(substring)

To check type of characters present in a string:

1). isalnum():Returns True if all the characters are alphanumerics(a-z, A-Z, 0-9)
2). isalpha():Returns True if all the characters are only alphabate symbols(a-z, A-Z)
3) .isdigit():Returns True if all the characters are digits only(0-9)
4) .islower():Returns True if all the characters are lower case alphabet symbols.
5) .isupper():Returns True if all the characters are upper case alphabet symbols.
6). istitle():Returns True if the string is in title case
7) .isspace():Returns True if string contains only spaces.

Formatting The string:

-->We can format the string with variable values by using replacement operator{ } and format() method.
Ex:
1) name="Prem"
2) salary=500000
3) age=25
4) print("{}'s salary is {} and his age is {}".format(name,salary,age))
5) print("{0}'s salary is {1} and his age is {2}".format(name,salary,age))
6) print("{x}'s salary is {y} and his age is {z}".format(z=age,y=salary,x=name))

List Data Structure:

--> If we want to represent a group of individual objects as a single entity where insertion order preserved and duplicates are allowed, then we should go for list.
--> Insertion order is preserved.
--> Duplicates objects are allowed.
--> Heterogeneous objects are allowed.
--> List is dynamic because based on our requirement we can increase the size and decrease the size.
--> In the list elements will be placed with in square brackets and with comma separator.
--> We can differentiate duplicate elements by using index and we can preserve insertion order by using index. Hence index will play very important role.
--> Python supports both +ve and -ve indexes. +ve index means from left to right where as
-ve index means right to left.

--> List objects are mutable. i.e we can change the content.

Creation of list objects:

1).We can create empty list objects as follows......

Ex:
```
list=[ ]
print(list)
print(type(list))
```

2).If we know elements already then we can create list as follows.
Ex: list=[10,20,30,40]

3).With dynamic input:

```
1)    list=eval(input("Enter List:"))
2)    print(list)
3)    print(type(list))
```

4).With list() function:

```
1)    l=list(range(10))
2)    print(l)
3)    print(type(l))
```

5).with split() function:

```
1)    s="Learning Python Is Very Easy"
2)    l=s.split()
3)    print(type(l))
4)    print(l)
```

6).By using Index:

--> index starts from 0.
--> List supports both +ve & -ve indexes.
1) list=[10,20,30,40]
2) print(list[1])==>20
3) print(list[0])==>10
4) print(list[-1])==>40
5) print(list[10])==>List index out of range

7).By using slice operator:

Syn:
list2=list1[start:stop:step]
start-->It indicates the index where slice has to start. default value is 0.

List Vs mutability:

-->Once we create a list object, we can modify its content. Hence list objects are mutable. Ex:

```
1)    n=[10,20,30,40]
2)    print(n[ 3])
3)   n[1]=50
4)    print(n)
```

Traversing the elements of list:

-->The sequential access of each element in the list is called as traversal.

1).By using while loop:

```
1) n=[0,1,2,3,4,5,6,7,8,9,10]
2) i=0
3) while i<len(n):
4) print(n[i])
5) i=i+1
```

2).By using for loop:

```
1) n=[0,1,2,3,4,5,6,7,8,9,10]
2) for i in n:
3) print(i)
```

3).To display only even numbers:

```
1) n=[0,1,2,3,4,5,6,7,8,9,10]
2) for i in n:
3) if i%2==0:
4) print(i)
```

Important functions of List:

To get the information about list:

1. len():

Returns number of elements present in the list.

Ex:

1) l=[10,20,30,40]
2) len(l)==>4

2. count():

It returns the number of occurences of specified item in the list

Ex:

1) n=[1,2,2,2,2,3,3]
2) print(n.count(1))==>1
3) print(n.count(2))==>4
4) print(n.count(3))==>2

3. index():

It returns index of first occurance of the specified element.

Ex:

1) l=[1, ,2,3,3,4]
2) print(l.index(1))==>0
3) print(l.index(2))==>1
4) print(l.index(3))==>3
5) print(l.index(4))==>5

Note:

If the specified element not present in the list then we will get ValueError. Hence
before index() method we have to check whether item present in the list or not by using in operator.

2).Manipulating elements of list:

1). append() function:

We can use append() function to add item at the end of the list.
Ex:

```
1)     l=[ ]
2)     l [ ]
3)     l.append("A")
4)     l.append("B")
5)     l.append("C")
6)     l
['A', 'B', 'C']
```

Ex: To add all elements to list upto 100 which are divisible by10

```
1)    list=[ ]
2)    for i in range(101):
3)    if i%10==0:
4)    list.append(i)
5)    print(list)
```

2).insert() function:

To insert an item at specified index position.
Ex:

```
1)    >>> n=[1,2,3,4]
2)    >>> n.insert(1,333)
3)    >>> n [1, 333, 2, 3, 4]
```

Difference between append() and insert():
append():

In list when we add any element it will come in last. i.e it will be last element. insert():
In list we can insert any element in a particular index number

Note:

If the specified index is greater than max index then element will be inserted at
last position. If the specified index is smaller than min index then element will be inserted at first position.

3).extend() function:

--> To add all items of one list to another list. Syn: l1.extend(l2)
--> All items present in l2 will be added to l1.

Ex:

```
1)    order1=["Python","Java",".Net"] 2)
      order2=["RC","KF","FO"]
3)    order1.extend(order2)
4)    print(order1)
```

4) .remove() function:

We can use this function to remove specified item from the list. If the item present multiple times then only first occurence will be removed.

Ex:

```
1) n=[10,20,30,10,30]
3) print(n)
```

o/p: D:\pythonclasses>py test.py [20, 30, 10, 30]

-->If the specified item not present in the list we will get ValueError

Ex:

```
1)    n=[10,20,30]
2)    n.remove(50)
3)    print(n)
```

o/p: ValueError:list.remove(x):x not in list

Note:
Hence before using remove() method first we have to check specified element present in the list or not by using in operator.
Ex:
```
1)  l=[10,20,30,40,10,30]
2)  x=int(input("Enter element to be removed:"))
3)  if x in l:
4)  l.remove(x)
5)  print("Removed successfully...")
6)  else:
7)  print("Specified element is not available")
```

5).pop() function:

--> It removes and returne the last element of the list.
--> This is only the function which manipulates list and returns some element.
Ex:
```
1)  n=[10,20,30,40]
2)  print(n.pop())
3)  print(n.pop())
4)  print(n)
```

o/p: D:\pythonclasses>py test.py 40 30
[10, 20]
--> If the list is empty then pop() function raises indexerror. Ex:

```
1) n=[ ]
2) print(n.pop())
```

o/p: IndexError: pop from empty list

Note:

1. pop() is the only function which manipulates the list and returns some value.
2. In general we can use append () and pop () functions to implement stack datastructure by using list, which follows LIFO (Last In First Out) order.

-->In general we can use pop() function to remove last element of the list, but we can use to remove elements based on index.

n.pop(index)==>To remove and return element present at specified index. n.pop()==>To remove and return last element of the list

Differences between remove() and pop():

remove()	**pop()**
1.We can use to remove Specified element from the list	1.We can use to remove last element.
2.It can't return any value.	2.It returned removed element
3.If specified element not Available then we wil get ValueError.	3.If list is empty then we will get an error.IndexError.

Note:

List objects are dynamic i.e based on our requirement we can increase and
decrease the size.
append(),insert(),extend()==>For increasing size/growable

nature remove(),pop()==>For decreasing size/shrinking nature

3).Ordering elements of the List:
==============================

1. reverse():

We can use reverse() order of elements of list.
Ex:

```
1)   n=[10,20,30,40]
2)   n.revrse()
3)   print(n)
```

o/p: [40,30,20,10]

2. sort() function:

In list by default insertion order is preserved. If we want to sort the elements of list according to default natural sorting order then we should go for sort() method.

For Numbers==>default sording order is ascending order.
For Strings==>default sorting order is alphabetical order.

Ex:

```
1)   n=[20,10,40,30]
2)   n.sort()
3)   print(n)
```

o/p:[10,20,30,40]
To sort in reverse of default natural sorting order:

-->We can sort according to reverse of default natural sorting order by using reverse=True argument.
Ex:

```
1)  n=[10,30,20,40]
2)  n.sort()
3)  print(n)==>[10,20,30,40]
4)  n.sort(reverse=True)
5)  print(n)==>[40,30,2010]
6)  n.sort(reverse=False)
7)  print(n)==>[10,20,30,40]
```

Aliasing and cloning of List objects:
=================================
The process of giving another reference variable to the existing list is called as aliasing.
Ex:

```
1)  x=[10,20,30,40]
2)  y=x
3)  y[1]=333
4)  print(x)
5)  print(y)
```

```
o/p:D:\pythonclasses>py test.py
[10, 333, 30, 40]
[10, 333, 30, 40]
```

1).By using slice operator:

Ex:

```
1)   x=[10,20,30,40]
2)   y=x[:]
3)   y[1]=333
4)   print(x)
5)   print(y)
```

```
o/p:D:\pythonclasses>py test.py
[10, 20, 30, 40]
[10, 333, 30, 40]
```

2).By using copy() function:

Ex:

```
1)   x=[10,20,30,40]
2)   y=x.copy()
3)   y[1]=333
4)   print(x)
```

Using Mathematical operators for list objects:
===
==
-->We can use + and * operators for list objects.

1).Concatenation operator(+):
We can use + to concatinate 2 lists into a single list.
Ex:

```
1)   a=[10,20,30]
2)   b=[40,50,60]
3)   c=a+b
4)   print("c:",c)
```

Ex:

```
o/p:D:\pythonclasses>py test.py
c: [10, 20, 30, 40, 50, 60]
```

```
1)   a=[10,20,30]
2)   b=[40,50,60]
3)   c=a.extend(b)
4)   print("a:",a)
5)   print("b:",b)
6)   print("c:",c)
```

```
o/p:D:\pythonclasses>py test.py a: [10, 20, 30, 40,
50, 60]
b: [40, 50, 60]
c: None
```

Note:

To use + operator compulsory both arguments should be list objects, otherwise we will get an error.
Ex:

c=a+40==>TypeError: can only concatenate list (not "int") to list c=a+[40]==>valid

2).Repetition operator(*):

We can use repetition operator * to repeat the elements of list specified number of times.
Ex:

```
1)  x=[10,20,30]
2)  y=x*3
3)  print(y)==>[10,20,30,10,20,30]
```

Comparing List objects:
=====================
We can use comparison operators for list objects.
Ex:

```
1)  x=["Dog","Cat","Rat"]
2)  y=["Dog","Cat","Rat"]
3)  z=["DOG","CAT","RAT"]
4)  print(x==y)==>True
```

```
5)   print(x==z)==>False
6)   print(x!=z)==>True
```

clear() function:

We can use clear() function to remove all elements of list.
Ex:

```
1)   n=[10,20,30,40]
2)   print(n)
3)   n.clear()
4)   print(n)
```

```
o/p:D:\pythonclasses>py test.py [10, 20, 30, 40]
[ ]
```

Nested Lists:
============
Sometimes we can take one list inside another list. Such type of lists are called as nested lists.
Ex:

```
1)   n=[10,20,[30,40]]
2)   print(n)
3)   print(n[0])
4)   print(n[1])
5)   print(n[2][0])
6)   print(n[2][1])
```

```
o/p:D:\pythonclasses>py test.py [10, 20, [30, 40]]
10
20
30
40
```

Tuple Data Structure

-->Tuple is exactly same as List except that it is immutable. i.e once we create tuple object, we can't perform any changes in that object.
-->Hence tuple is read only version of list.
-->If our data is fixed and never changes then we should go for tuple.
-->Insertion order is preserved.
-->Duplicates are allowed.
-->Heterogeneous objects are allowed.
-->We can preserve insertion order and we can differentiate duplicate objects by using index. Hence index will play very important role in Tuple also.
-->We can represent tuple elements within parenthesis and with comma separator. Parenthesis are optional but recommended to use.

Ex:

```
1. t=10,20,30,40
2.   print(t)
```

3. print(type(t))

o/p:D:\pythonclasses>py test.py (10, 20, 30, 40)
<class 'tuple'>

t=()
print(type(t))==>tuple

Note: We have to take special care about single valued tuple. COmpulsory the value should ends with comma, otherwise it is not treated as tuple.
Ex:

1. t=10
2. print(t)
3. print (type(t))

Tuple Creation:

1.t=():
Creation of empty tuple

2.t=(10,) or t=10,
Creation of single valued tuple, paranthesis are optional, should ends with comma.

3.t=10,20,30 or t=(10,20,30)
Creatin of multiple values tupl & parenthesis are optional.

4. By using tuple() function:

```
1. l=[10,20,30,40]
2. t=tuple(l)
3. print(t)
4. print(type(t))
```

Accessing elements of tuple:

============================

We can access either by index or by using slice operator.

1. By using index:

```
1. t=(10,20,30,40)
2. print(t[0])==>10
3. print(t[-1])==>40
4. print(t[100])==>IndexError
```

2. By using slice operator:

```
1. t=(10,20,30,40,50,60)
2. print(t[2:5])
3. print(t[2:100])
4. print(t[::2])
```

o/p:

```
(30, 40, 50)
(30, 40, 50, 60)
(10, 30, 50)
```

Tuple Vs immutability:

====================

-->Once we create tuple, we can't change its content.
-->Hence tuple objects are immutable. Ex:

1. t=(10,20,30,40,50)
2. t[1]=60==>TypeError: 'tuple' objects does not support item assignment.

Mathematical operators for tuple:

==============================

-->We can apply + and * operators for tuple.
1.concatination operator(+):
1. t1=(10,20,30,40)
2. t2=(10,20,50,60)
3. t3=t1+t2
2. Multiplication operator or repetition operator(*):
1. t1=(10,20,30,40)
2. t2=t1*2
3. print(t2)

o/p:(10, 20, 30, 40, 10, 20, 30, 40)

Important function in tuple:

=========================

1. len():
To return number of elements present in tuple

Ex:

```
1.   t=(10,20,30,40)
2.   len(t)==>4
```

2. count():
To return number of occurences of given element in the tuple.
Ex:

```
1.   t=(10,20,30,10,10)
2.   t.count(10)==>3
```

3).index():
-->Returns index of first occurence of the given element.
-->If the specified element is not available then we will get ValueError.
Ex:

```
1.   t=(10,20,30,30,10,40)
2.   t.index(30)==>2
3.   t.index(50)==>ValueError
```

4. sorted():
To sort elements based on default natural sorting order.
Ex:

```
1.   t=(40,20,50,10,30)
```

```
2.   t1=sorted(t)
3.   print(t)
4.   print(t1)
```

 Ex:
 o/p:

```
(40, 20, 50, 10, 30)
[10, 20, 30, 40, 50]==>After sorting it will return as list
```

```
1.   t=(40,20,50,10,30)
2.   t1=tuple(sorted(t))
3.   print(t)
4.   print(t1)
```

o/p:

```
(40, 20, 50, 10, 30)
(10, 20, 30, 40, 50)
```

Ex:

```
1.   t1=sorted(t,reverse=True)
2.   print(t1)==>[50,40,30,20,10]
```

5. min() and max() functions:

These functions returns min and max values according to default natural sorting order. Ex:

```
1. t=(40,20,50,10,30)
2. print(min(t))==>10
3. print(max(t))==>50
```

Ex:

```
1. t="Prsoftwares"
2. print(min(t))==>a
3. print(max(t))==>w
```

6. cmp():

-->It compares the elements of both tuples.
-->If both tuples are equal then returns 0.
-->If the first tuple is less than second tuple then it returns -1
-->If the first tuple is greater than second tuple then it returns +1
Note: cmp() function is available only in python-2 but not in python-3

Tuple Packing and Unpacking:

-->We can create a tuple by using packing a group of variables. Ex:

```
1. a=10
2. b=20
3. c=30
4. d=40
5. t=a,b,c,d
```

```
6.   print(t)
```

o/p:(10, 20, 30, 40)
-->Here a,b,c,d are packed into a tuple t. This is nothing but tuple packing.

-->Tuple unpacking is the reverse process of tuple packing.
-->We can unpack a tuple and assign its values to different variables. Ex:

```
1.   t=(10,20,30,40)
2.   a,b,c,d=t
3.   print("a=",a,"b=",b,"c=",c,"d=",d)
```

Note:

o/p:
a= 10 b= 20 c= 30 d= 40

At the time of tuple unpacking the number if variables and number of values

should be same. otherwise we will get an error. Ex:

```
1.   t=(10,20,30,40)
```

```
2.   a,b,c=t
3.   print("a=",a,"b=",b,"c=",c)
```

Difference between List and Tuple

List	Tuple
1.List a group of comma separated values within square brackets and square brackets are mandatory. Ex: l=[10,20,30]	1.Tuple is a group of comma separated values within parenthesis parenthesis are optional. Ex: t=(10,20,30) or t=10,20,30
2.List objects are mutable i.e once we creates list object we can perform any changes in that object. ex:l[1]=50	2.Tuple objects are immutable i.e once we creates tuple objects we can't change it content. Ex:t[1]=50-->ValueError
3.If the content is not fixed and keep on changing then we should go for list.	3.If the content is fixed and never changes then we should go for tuple.
4.Comprehensions are available.	4.There is no comprehensions.

Set Data Structure

-->If we want to represent a group of unique values as a single entity then we should go for set.
-->Duplicates are not allowed.
-->Insertion order is not preserved. But we can sort the elements.
-->Indexing and slicing not allowed for the set.
-->Heterogeneous elements are allowed.
-->Set   perform any changes in that object based

on our requirement.
-->We can represent set elements with curly braces and with comma separation.
-->We can apply mathematical operations like union, intersection, difference etc on set objects.

Creation of Set Objects:

Ex:

```
1.   s={10,20,30,40}
2.   print(s)
3.   print(type(s))
```

o/p:

```
{40,10,20,30}
<class 'set'>
```

-->We can create set objects by using set() function. s=set(any sequence)

Ex:

Ex:

```
l=[10,20,30,40,10,20]
s=set(l)
print(s)

s=set(range(5))
print(s)
```

s={ }==>It treated as dictionary but not an empty set. Ex:

```
s={ }
print(type(s)) o/p: <class 'dict'>
```

Ex:

```
1.   s=set()
2.   print(s)
3.   print(type(s))
```

o/p:

```
set()
<class 'set'>
```

Important functions of set:
----
1. add(x):

Add item x to the set.

Ex:

```
1.   s={10,20,30}
2.   s.add(40)
3.   print(s)
```

2. update(x,y,z):

-->To add mutiple items to the set.
-->Arguments are not individual elements and these are iterable elements like list, range etc..
-->All elements present in the given iterable objects will be added to the set.
Ex:

```
1.   s={10,20,30}
2.   l=[40,50,60]
3.   s.update(l,range(5))
4.   print(s)
```

Q: What is difference between add() and update() functions in set?
-->We can use add() to add individual items to the set, where as we can use update() function to add multiple items to the set.
-->add() function can take only one argument where as update function can take any number of arguments but all arguments should be iterable objects.

Q: Which are valid?
 s.add(10)   Valid

s.add(10,20,30) Invalid
 s.update(30) Invalid
s.update(range(1,10,2),range(0,10,2))   Valid

3. copy():
-->Returns copy of the set
-->It is cloned object.
Ex:

```
1. s={10,20,30}
2. s1=s.copy()
3. print(s1)
```

4. pop():
It removes and returns some random element from the set.

Ex:

o/p:

```
{40, 10, 20, 30}
40
{10, 20, 30}
```

5. remove(x):

-->It removes specified element from the set.
-->If the specified element not present in the set then we will get KeyError. Ex:

```
1. s={40,20,30,10}
2. s.remove(30)
3. print(s)==>{40,20,10}
4. s.remove(50)==>KeyError:50
```

6. discard(x):

-->It removes specified element from the set.
-->If the specified element not present in the set then we won't get any error. Ex:

```
1. s={40,20,30,10}
2. s.discard(30)
3. print(s)==>{40,20,10}
4. s.discard(50)
5. print(s)==>{40,20,30,10}
```

7. clear():

-->To remove all elements from the set. Ex:

```
1. s={10,20,30}
2. print(s)
3. s.clear()
```

4. print(s)

o/p:
{10,20,30}
set()
Mathematical operations on the set:
==================================
1).union():
x.union(y)==>We can use this function to return all elements present in both sets.
Syn: x.union(y) or x|y Ex:

1. x={10,20,30,40}
2. y={30,40,50,60}
3. print(x.union(y))#{40, 10, 50, 20, 60, 30}
4. print(x|y) #{40, 10, 50, 20, 60, 30}

2).intersection():
x.intersection(y) or x & y
-->Returns common elements present in both x and y Ex:

1. x={10,20,30,40}
2. y={30,40,50,60}
3. print(x.intersection(y))#{40, 30}
4. print(x&y)#{40, 30}

3).difference():
x.difference(y) or x-y
-->Returns the elements present in x but not in y.
Ex:

```
1.   x={10,20,30,40}
2.   y={30,40,50,60}
3.   print(x.difference(y))#{10, 20}
4.   print(x-y)#{10, 20}



5.   print(y-x)#{50, 60}
```

4).symmetric_difference():
x.symmetric_difference(y)   or x^y
-->Returns element present in either x or y but not in both. Ex:

```
1.   x={10,20,30,40}
2.   y={30,40,50,60}
3.   print(x.symmetric_difference(y))#{10, 50, 20,
60}
4.   print(x^y)#{10, 50, 20, 60}
```

Membership operators:(in, not in)

Ex:

```
1.   s=set("premsofts")
2.   print(s)
3.   print('p' in s)
4.   print('z' in s)
```

```
o/p:{'p', 'r', 'e', 'm', 's', 'o', 'f', 't', 's'}
 True
False
```

Set comprehension:

-->set comprehension is possible.

s={i *i for i in range(5)} print(s)#{0, 1, 4, 9, 16}

s={2 **i for i in range(2,10,2)} print(s)#{16, 256, 64, 4}

set objects won't support indexing and slicing:

Ex:

```
1.   s={10,20,30,40}
2.   print(s[0])     TypeError: 'set' object does not support indexing
3.   print(s[1:3]) TypeError: 'set' object does not support indexing
```

Q: w.a.p to eliminate duplicates present in the list.
Approach-1:

```
1.   l=eval(input("Enter List Of Values:"))
3.   print(s)
```

```
o/p:Enter List Of Values:[10,20,10,20,30,40,30]
{40, 10, 20, 30}
```

Approach-2:

```
1.   l=eval(input("Enter List Of Values:"))
2.   l1=[ ]
3.   for x in l:
4.   if x not in l1:
5.   l1.append(x)

6.   print(l1)
```

```
o/p:
D:\pythonclasses>py test.py
Enter List Of Values:[10,20,20,10,30,40] [10, 20,
30, 40]
```

Q:w.a.p to print different vowels in the given word?

```
1.   w=input("Enter word to search for vowels:")
2.   s=set(w)
3.   v={'a','e','i','o','u'}
4.   d=s.intersection(v)
5.   print("The different vowels present
in",w,"are:",d)
```

Dictionary Data Structure

-->We can use list, tuple and set to represent a group of individual objects as a single entity.
-->If we want to represent a group of objects key-value pairs then we should go for dictionary.

Ex:

rollno-----name
phone number---address
ip address----domain name
product name----price

-->Duplicate keys are not allowed but values can be duplicated.
-->Heterogeneous objects are allowed for both key and values.
-->Insertion order is not preserved.
-->Dictionaries are mutable.

-->Dictionaries are dynamic.
-->Indexing and slicing concepts are not applicable.

Note:
In C++ and java Dictionaries are known as "Map" where as in perl and Ruby it is
known as "Hash"

How to create dictionary?
======================
d={ } or d=dict()
-->We are creating empty dictionary. We can add entries as follows.

```
1.  d={}
2.  d1=dict()
3.  print(type(d))
4.  d[100]="Prsoftwares"
5.  d[200]="Premsofts"
6.  d[300]="Prem"
7.  print(d)#{100: 'Prsoftwares', 200: 'Premsofts',
300: 'Prem'}
```

-->If we know the data in advance then we can create dictionary as follows
d={100:'Prsoftwares',200:'Premsofts',300:'Prem'}
d={key:value,key:value}

How to access data from the dictionary?

-->We can access data by using index.

d={100:"Prsoftwares",200:"Premsofts",300:"Prem"}
print(d[100])==>Prsoftwares print(d[300])==>Prem

-->If the specified is not available then we will get a KeyError print(d[400])==>KeyError:400

-->We can prevent this by checking whether key is already available or not by using has_key() function or by using in operator.

Ex:d.has_key(400)==>Returns 1 if key is available otherwise returns 0.

-->But has_key() function is available in python-2 but not in python-3. Hence compulsory we have to use in operator.

Ex:
```
if 400 in d:
print(d[400])
```

Q: w.a.p to enter name and percentage marks in a dictionary and display information on the screen.

```
rec={ }
n=int(input("Enter Number Of Students: "))#3
i=1
while i<=n:
    name=input("Enter Student Name:")
    marks=int(input("Enter % of Marks:"))
    rec[name]=marks
    i=i+1
print("Name of the student","\t","% of marks")
for x in rec:
    print("\t",x,"\t\t",rec[x])
```

How to update dictionaries?

Ex:

```
d={100:"Prsoftwares",200:"Premsofts",300:"Prem"}
print(d)
d[400]="python"
print(d)
d[100]="JAVA"
print(d)
```

How to delete elements from dictionary:
----

del d[key]:

It deletes entry associated with the specified key. If the key is not available then we will get KeyError.
Ex:

```
1.
    d={100:"Prsoftwares",200:"Premsofts",300:"Pr
em"}
2.  print(d)
3.  del d[300]
4.  print(d)
5.  del d[400]

6.  print(d)
```

d.clear():

To remove all entries from the dictionary.
Ex:

```
1.
    d={100:"Prsoftwares",200:"Premsofts",300:"Pr
em"}
2.  print(d)
3.  d.clear()
```

```
4.   print(d)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'Prsoftwares', 200: 'Premsofts', 300: 'Prem'}
{}
```

del d:

Ex:

To delete total dictionary. Now we can't access d.

```
1.   d={100:"Prsoftwares",200:"Premsofts",300:"Prem"}
2.   print(d)
3.   del d
4.   print(d)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'Prsoftwares', 200: 'Premsofts', 300: 'Prem'}
NameError: name 'd' is not defined
```

Important functions of dictionary:
----

1).dict():
To create a dictionary. d=dict()==>It creates empty dictionary

```
1.
      d=dict({100:"Prsoftwares",200:"Premsofts"})==
>It creates dictionary with specified elemen
ts
2.   print(d)
3.
      d=dict([(100,"Prsoftwares"),(200,"Premsofts")])
==>It creates a dictionary with the given lis
t of tuple lements.
4.   print(d)
5.
      d=dict(((100,"PRSOFTWARES"),(200,"PREM
SOFTS")))==>It creates a dictionary with the given
tuple of tuple lements.
6.   print(d)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'Prsoftwares', 200: 'Premsofts'}
{100: 'Prsoftwares', 200: 'Premsofts'}
{100: 'PRSOFTWARES', 200: 'PREMSOFTS'}
```

2).len():
Returns the number of items in the dictionary.

3).clear():

To remove all the elements from the dictionary.

4. get():
To get the value associated with the key.
d.get(key):
If the key is available then returns the corresponding value otherwise returns None. It wont raise any error.

d.get(key,defaultvalue):
If the key is available then returns the corresponding value otherwise returns default value.

Ex:

```
d={100:"Prsoftwares",200:"Premsofts",300:"Prem"}
print(d[100])#Prsoftwares
print(d[400])#KeyError:400
print(d.get(300))#Prem
print(d.get(400))#None
print(d.get(100,"Guest"))#Prsoftwares
print(d.get(400,"Guest"))#Guest
```

5).pop():
d.pop(key):
It removes the entry associated with the specified key and returns the corresponding value.

If the specified key is not available then we will get KeyError.
Ex:

```
1.
     d={100:"Prsoftwares",200:"Premsofts",300:"Prem"}
2.   print(d)
3.   print(d.pop(300))
4.   print(d)
5.   d.pop(400)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'Prsoftwares', 200: 'Premsofts', 300: 'Prem'}
Prem
{100: 'Prsoftwares', 200: 'Premsofts'}
KeyError: 400
```

6).popitem():
It removes an arbitrary item(key-value) from the dictionary and return it.
Ex:

```
1.
     d={100:"Prsoftwares",200:"Premsofts",300:"Prem"}
2.   print(d)
3.   print(d.popitem())
```

```
4.   print(d)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'Prsoftwares', 200: 'Premsofts', 300: 'Prem'}
{100: 'Prsoftwares', 200: 'Premsofts'}
```

-->If the dictionary is empty then we will get KeyError. d={ }
print(d.popitem())==>KeyError:'popitem():dictionary is empty'.

7).keys():
It returns all keys associated with dictionary.
Ex:

```
1.
     d={100:"Prsoftwares",200:"Premsofts",300:"Prem"}
2.   print(d.keys())
3.   for k in d.keys():
4.   print(k)
```

```
o/p:D:\pythonclasses>py test.py dict_keys([100, 200, 300])
100
200
300
```

8).values():
It returns all values associated with dictionary.

Ex:

```
1.
     d={100:"Prsoftwares",200:"Premsofts",300:"Pr
em"}
```

```
o/p:D:\pythonclasses>py test.py
dict_values(['Prsoftwares', 'Premsofts', 'Prem'])
Prsoftwares
Premsofts Prem
```

9).items():
It returns list of tupes represented key-value pairs.
[(k,v),(k,v),(k,v)]
Ex:

```
1.
     d={100:"Prsoftwares",200:"Premsofts",300:"Pr
em"}
3.   print(k,"--",v)
```

```
o/p:D:\pythonclasses>py test.py 100 --
Prsoftwares
200 -- Premsofts
300 -- Prem
```

10).copy():
To create exactly duplicate dictionary (cloned copy)
Ex:

```
1.
     d={100:"Prsoftwares",200:"Premsofts",300:"Pr
em"}
2.   d1=d.copy()
3.   print(id(d))
4.   print(id(d1))
```

```
o/p:D:\pythonclasses>py test.py 2791188556752
2791188556824
```

11).setdefault(): d.setdefault(k,v)
-->If the key is already available then this function returns the corresponding value.
-->If the key is not available then the specified key-value will be added as new item to the dictionary.
Ex:

```
1.
     d={100:"Prsoftwares",200:"Premsofts",300:"Pr
em"}
2.   print(d.setdefault(400,"secunderabad"))
3.   print(d)
```

```
5.   print(d)
```

```
o/p:D:\pythonclasses>py test.py secunderabad
{100: 'Prsoftwares', 200: 'Premsofts', 300: 'Prem',
400: 'secunderabad'} Prsoftwares
{100: 'Prsoftwares', 200: 'Premsofts', 300: 'Prem',
400: 'secunderabad'}
```

12).update(): d.update(x)
All items in the dictionary x will added to dictionary d.
Ex:

```
1.   d={100:"Prsoftwares",200:"Premsofts",300:"Prem"}
2.   d1={'a':"apple",'b':"banana",'c':"cat"}
3.   d.update((d1))
4.   print(d)
5.   d.update([(333,"A"),(999,"B")])
6.   print(d)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'Prsoftwares', 200: 'Premsofts', 300: 'Prem',
'a': 'apple', 'b': 'banana', 'c': 'cat'}
{100: 'Prsoftwares', 200: 'Premsofts', 300: 'Prem',
'a': 'apple', 'b': 'banana', 'c': 'cat', 333: 'A', 999: 'B'}
```

Q: w.a.p to take dictionary from the keyboard and print sum of values. sol:

```
d=eval(input("Enter Dictionary:"))
print("Sum:",s)
```

```
o/p:D:\pythonclasses>py test.py
Enter Dictionary:{'A':100,'B':200,'C':300} Sum: 600
```

Q:w.a.p to find number of occurances of each letter present in the given string? sol:

```
word=input("Enter any word:")
d={}
for x in word:
d[x]=d.get(x,0)+1
print(d)
print(sorted(d))
```

```
for k,v in d.items():
#print(k,"occurs",v,"times")
print("{} occured {}".format(k,v))
```

```
o/p:D:\pythonclasses>py test.py Enter any
word:mississippi
{'m': 1, 'i': 4, 's': 4, 'p': 2}
```

```
['i', 'm', 'p', 's']
m occured 1
i occured 4
s occured 4
p occured 2
```

Q: w.a.p to find number of occurances of each vowel present in the given string sol:

```
word=input("Enter some string:")
vowels={'a','e','i','o','u'}
d={}
for x in word:
if x in vowels:
d[x]=d.get(x,0)+1
for k,v in sorted(d.items()):
print(k,"occured",v,"times")
```

o/p:D:\pythonclasses>py test.py Enter some string:abaiobeioeiaieus a occured 3 times
e occured 3 times i occured 4 times o occured 2 times u occured 1 times

Q:w.a.p to accept student name and marks from the keyboard and creates a dictionary, also display student marks by taking student name as input?

```
n=int(input("Enter the number of students:"))
d={}
for i in range(n):
```

```
4)   name=input("Enter student name:")
5)   marks=input("Enter marks:")
6)   d[name]=marks
7)   print(d)
8)   while True:




9)   name=input("Enter student name to get
marks:")
10) marks=d.get(name,-1)
11) if marks==-1:
12) print("Student not found")
13) else:
14) print("The marks of",name,"are",marks)
15) option=input("Do you want to find another
student marks[Yes|No]")
16) if option=="No":
17) break
18) print("Thanks for using our application")
```

```
o/p:D:\pythonclasses>py test.py Enter the number
of students:3 Enter student name:Premsofts
Enter marks:90
Enter student name:Prsoftwares Enter marks:98
Enter student name:Prem Enter marks:100
{'Premsofts': '90', 'Prsoftwares': '98', 'Prem': '100'}
Enter student name to get marks:Prem The marks
of Prem are 100
```

```
Do you want to find another student
marks[Yes|No]Yes Enter student name to get
marks:secunderabad
Student not found
Do you want to find another student
marks[Yes|No]Yes Enter student name to get
marks:Prsoftwares
The marks of Prsoftwares are 98
Do you want to find another student
marks[Yes|No]No Thanks for using our application
```

Dictionary comprehension:

-->Comprehension concept applicable for dictionary also. Ex:

```
1.   squares={x:x*x for x in range(1,6)}
2.   print(squares)
3.   doubles={x:2*x for x in range(1,6)}
4.   print(doubles)
```

o/p:

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
{1: 2, 2: 4, 3: 6, 4: 8, 5: 10}
```

FUNCTIONS

If a group of statements is repeatedly required then it is not recommended to write these

statements every time separately. We have to define these statements as a single unit and we can call that unit any number of times based on our requirement without re-writting. THis unit is nothing but a function.
-->The main advantage of functions is code -Re-usability.
Note:In other languages functions are known as methods, procedures,subroutines etc...

-->Python supports
2-types of functions
1).Built in functions
2).User defined functions

1).Built in Functions:

The functions which are coming along with python s/w automatically, are called as built in functions or pre-defined functions.
Ex:
id(), type(), input(), eval().........

2).User Defined Functions:

The functions which are developed by programmers explicitly according to business requirements, are called as user defined functions.
Syn:
def function_name(parameter): body

-----
return value
Note: while creating functions we can use 2 keywords 1.def(mandatory)
2. return(optional)

Ex: write a function to print Hello gud mng.....

```
1. def wish():
2. print("Hello gud mng............")
3. wish()




5. wish()
```

```
o/p:D:\pythonclasses>py test.py Hello gud
mng............
Hello gud mng............
Hello gud mng............
```

Parameters:

Parameters are inputs to the function. If a function contains parameters, then at the time of calling, compulsory we should provide values, otherwise we will get error.

Ex: write a function to take name of the student as input and print wish message by name

```
def wish(name):
print("Hello", name," gud mng............")
wish("Premsofts")
wish("Guest")
```

```
o/p:D:\pythonclasses>py test.py Hello Premsofts
gud mng............
Hello Guest gud mng............
```

Ex: write a function to take number as input and print its square value.

```
def squareit(number):
print("the square of", number," is:",number**2)
squareit(4)
squareit(5)
```

```
o/p:D:\pythonclasses>py test.py the square of 4 is:
16
the square of 5 is: 25
```

Return Statement:

Function can take input values as parameters and executes business logic, and returns output to the caller with the return statement.

Q:write a function to accept 2-numbers as input and return sum

```
1.   def add(x,y):
3. result=add(10,20)





5. print("The sum is:",add(10,20))
```

```
o/p:D:\pythonclasses>py test.py The sum is: 30
The sum is: 30
```

-->If we are not writing return statement then default return value is None. Ex:

```
1. def f1():
2.   print("hello")
3. f1()
4. print(f1())
```

```
o/p:D:\pythonclasses>py test.py hello
hello None
```

Q:write a function to check whether the given number is even or odd?

```
def even_odd(num):
if num%2==0:
print(num,"is even number")
else:
print(num,"is odd number")
even_odd(10)
even_odd(15)
```

```
o/p:D:\pythonclasses>py test.py 10 is even
number
15 is odd number
```

Q:write a function to find factorial of given number.

```
def fact(num):
result=1
while num>=1:
result=result*num
num=num-1
return result
print(fact(5))
for i in range(1,5):
print("The factorial ",i,"is",fact(i))
```

```
o/p:D:\pythonclasses>py test.py 120
The factorial 1 is 1
The factorial 2 is 2
The factorial 3 is 6
```

The factorial 4 is 24

Returning multiple values from a function:

-->In other languages like C, C++ and java, function can return atmost one value. BUt in python, a function can return any number of values.

Ex:

```
def sum_sub(a,b):
sum=a+b
sub=a-b
return sum,sub
x,y=sum_sub(100,50)
print("sum is: ",x)
print("sub is: ",y)
```

o/p:D:\pythonclasses>py test.py sum is: 150
sub is: 50 Ex:

```
def calc(a,b):
sum=a+b
sub=a-b
mul=a*b
div=a/b
return sum,sub,mul,div
t=calc(100,50)
print(t)
print(type(t))
```

```
10. print("The Results are:")
11. for i in t:
12.  print(i)
```

```
o/p:D:\pythonclasses>py test.py (150, 50, 5000,
2.0)
<class 'tuple'>



The Results are: 150
50
5000
2.0
```

Types of arguments:

```
def f1(a,b):
--------
--------
--------- f1(10,20)
```

-->a,b are formal arguments where as 10,20 are actual arguments.

-->There are 4-types of actual arguments are allowed in python
1.positional arguments
2.keyword arguments
3.default arguments

4.variable length arguments

1.positional arguments:

These are the arguments passed to function in correct positional order Ex:

```
1.  def sub(a,b):
2.  print(a-b)
3.  sub(100,200)
4.  sub(200,100)
```

-->The number of arguments and position of arguments must be matched. If we change the order then the result may be changed.

```
o/p:D:\pythonclasses>py test.py
-100
100
```

-->If we change the number of arguments then we will get an error. Ex: sub(200,100,300)
o/p:ypeError: sub() takes 2 positional arguments but 3 were given

2).keyword arguments:

We can pass argument values by keyword i.e by parameter name. Ex:

```
1. def wish(name,msg):
```

```
2.   print("Hello",name,msg)
3. wish(name="Prsoftwares",msg="gud mng....")
4. wish(msg="gud mng....",name="Prsoftwares")
```

o/p:D:\pythonclasses>py test.py Hello Prsoftwares gud mng....
Hello Prsoftwares gud mng....

-->Here the order of arguments is not important but number of arguments must be matched.

Note:
We can use both positional and keyword arguments simultaneously. But first we
have to take positional arguments and then take keyword arguments, otherwise we will get an error.
Ex:

```
1.   def wish(name,msg):
2.   print("Hello",name,msg)
3.   wish("Prsoftwares","Gud Mng.....")==>Valid
4.   wish("Prsoftwares",msg="Gud
Mng.....")==>Valid
5.   wish(name="Prsoftwares","Gud
Mng........")==>Invalid
```

o/p:SyntaxError: positional argument follows keyword argument
3.Default argument:

Sometimes we can provide default values for our

positional arguments Ex:

```
1.   def wish(name="Prsoftwares"):
2.   print("Hello",name,"Gud Mng.....")
3.   wish()
4.   wish("Premsofts")
```

```
o/p:D:\pythonclasses>py test.py Hello Prsoftwares
Gud Mng.....
Hello Premsofts Gud Mng.....
```

-->If we are not passing any name then only default value will be considered.

4).Variable length arguments:

-->Sometimes we can variable number of arguments to our function, such type of arguments are called as variable length arguments.
-->We can declare a variable length argument with * symbol as follows.
Syn: def f1(*n):
-->We can call this function by passing any number of arguments including zero number, internally all these values represented in the form of tuple.
Ex:

```
def sum(*n):
total=0
for i in n:
total=total+i
print("Sum is:",total)
sum()
sum(10,20)
sum(10,20,30)
sum(10,20,30,40)
```

```
o/p:D:\pythonclasses>py test.py Sum is: 0
Sum is: 30
Sum is: 60
Sum is: 100 Ex:
```

Function vs Module vs Package vs Library:
====================================
-->A group of lines with some name is called as a function.
-->A group of functions saved to a file is called as module.
-->A group of modules is nothing but a package.
-->A group of packages is nothing but a library.

Types of Variables:

Python supports two types of variables.
-->Global Variable
-->Local Variable

Global variable:

-->The variables which are declared outside the function are called as global variables.
-->These variables can be accessed in all functions of that module.

Ex:

```
a=10#global variable
def f1():
print('f1:',a)
def f2():
print('f2:',a)
f1()

f2()
```

```
o/p: 10
10
```

Local Variable:

-->The variables which are declared inside a function are called as local variables.
-->Local variables are available only for the function in which we declared it. i.e from outside of function we can't access.
Ex:

```
def f1():
a=10
print('f1:',a)
def f2():
print('f2:',a)
f1()

f2()
```

o/p: f1: 10
NameError: name 'a' is not defined

global keyword:

We can use global keyword for the following 2 purposes
-->To delcare global variable inside a function.
-->To make global variable available to the function so that we can perform required modifications.
Ex:

```
a=10
def f1():
a=333
```

```
print('f1:',a)
def f2():
print('f2:',a)
f1()

f2()
```

```
o/p:D:\pythonclasses>py test.py f1: 333
f2: 10
```

Ex:

```
a=10#global
def f1():
global a
a=333#local
print('f1:',a)
def f2():
print('f2:',a)
f1() f2()
f2() f1()
```

```
o/p:D:\pythonclasses>py test.py
     o/p:D:\pythonclasses>py test.py f1: 333  f2:
f2: 333  f1: 333
```

Ex:

```
def f1():
global a
```

```
a=10
print('f1:',a)
def f2():
print('f2:',a)
f1()
f2()
```

```
o/p:D:\pythonclasses>py test.py f1: 10
f2: 10
```

Note: If the global variable and local variable having same name then we can access global variable inside a function as follows.

Ex:

```
a=10#global variable
def f1():
a=333#local variable
print('f1:',a)
print(globals()['a'])
f1()
```

```
o/p:D:\pythonclasses>py test.py f1: 333
10
```

Recursive Functions:

A function that calls itself is known as Recursive Function.

Ex: factorial(3):
3*factorial(2)
3*2*factorial(1)
3*2*1*factorial(0)

factorial(n):n*factorial(n-1)

The main advantages of recursive functions are:
-->We can reduce length of the code and improve readability.
-->We can solve complex problems very easily.

write a function to find factorial of given number with recusrion.

```
def factorial(n):
if n==0:
result=1
else:
result=n*factorial(n-1)
return result
print("Factorial of 4 is: ",factorial(4))
print("Factorial of 0 is: ",factorial(0))
```

```
o/p:D:\pythonclasses>py test.py Factorial of 4 is:
24
Factorial of 0 is: 1
```

Anonymous Functions:
====================
-->Sometimes we can declare a function without name, such type of nameless functions are called as anonymous functions or lamda functions.
-->The main advantage of anonymous function is just for instant use(i.e for one time usage)

Normal Function:

-->We can define by using def keyword.
```
def  squareit(n):
return n*n
```

lambda function:

-->We can define by using lambda keyword.
```
lambda n:n*n
```

Syntax of lambda function:
----
lambda argument_list:expression
Note:
By using lambda functions we can write concise code so that readability of the
program will be improved.

Q: w.a.p to create a lambda function to find square of given number
----

```
1. s=lambda n:n*n
2. print("The square of 3 is:",s(3))
3. print("The square of 5 is:",s(5))
```

o/p:D:\pythonclasses>py test.py The square of 3 is: 9
The square of 5 is: 25

Q: Lambda function to find sum of 2-given numbers

```
1. s=lambda a,b:a+b
2. print("The sum of 10 and 20 is:",s(10,20))
3. print("The sum of 100 and 200 is:",s(100,200))
```

o/p:D:\pythonclasses>py test.py

The sum of 10 and 20 is: 30
The sum of 100 and 200 is: 300

Q:Lambda function to find biggest of given values:

```
1.   s=lambda a,b:a if a>b else b
3. print("The biggest of 100 and 200
is:",s(100,200))
```

```
o/p:D:\pythonclasses>py test.py The biggest of 10
and 20 is: 20
The biggest of 100 and 200 is: 200
```

Note:
Lambda function internally returns expression value and we are not required to
write return statement explicitly.
Some times we can pass function as argument to another function. In such case lambda functions are best choice.

-->We can use lambda functions very commonly with **filter(), map() and reduce()** functions,bcoz these functions expect function as aegument.

1).filter function:

We can use filter() function to filter values from the given sequence based on some condition.
Syn:
filter(function,sequence)
-->Where function argument is responsible to perform conditional check sequence can be list or tuple or string.

Q: w.a.p to filter even numbers from the list by using filter()
----
without lambda function:

```
def isEven(n):
if n%2==0:
return True
else:
return False
l=[0,5,10,15,20,25,30]
print(type(filter(isEven,l)))
```

9. print(l1)

```
o/p:D:\pythonclasses>py test.py
<class 'filter'> [0, 10, 20, 30]
```

with lambda function:

```
l=[0,5,10,15,20,25,30]
l1=list(filter(lambda x:x%2==0,l))
print(l1)
```

```
4.   l2=list(filter(lambda x:x%2!=0,l))

5.   print(l2)
```

```
o/p:D:\pythonclasses>py test.py [0, 10, 20, 30]
[5, 15, 25]
```

2).map() function:

For every element present in the given sequenece, apply some functionality and generate new element with the required modifications. For this requirement we should go for map() function.

Ex: For every element present in the list perform double and generate new list of doubles

Syn:
map(function,sequenec)
-->The function can be applied on each element of sequence and generates new sequence.

without lambda:
---------------------- l=[1,2,3,4]

```
l1=[]
for i in l:
 l1.append(i*2)
print(l1)
```

with lambda:

```
1.   l=[1,2,3,4,5]
2.   l1=list(map(lambda x:2*x,l))
3.   print(l1)==> [2, 4, 6, 8, 10]
```

Ex: To find square of given numbers 1.
l=[1,2,3,4,5]
3. print(l1)==> [1, 4, 9, 16, 25]

3).reduce() function:
----

reduce() function reduces sequence of elements into a single element by applying the specified function.
Syn:
reduce(function, sequence)

-->reduce() function present in functools module and hence we should write import statement.
Ex:

```
1.   from functools import *
2.   l=[10,20,30,40,50]
3.   result=reduce(lambda x,y:x+y,l)
4.   print(result)
```

```
o/p:150
```

Ex:

```
from functools import *
l=[10,20,30,40,50]
result=reduce(lambda x,y:x*y,l)
print(result)
```

```
o/p:12000000
```

Ex:

```
from functools import *
result=reduce(lambda x,y:x+y,range(1,101))
print(result)
```

```
o/p:5050
```

Function Decorators

Decorator is a function which can take a function as a argument and extend its functionality and returns modified function with extended functionality.

Input function-------->
Decorator Function---------->o/p function with extended fun

The main objective decorator function is we can extend the functionality of existing functions without modifies that function.

Decorators help to make our code shorter and more pythonic. This concept is veryhelpful while developing web applications with Django.

Ex :
```
def wish(name):
print("Hello",name,"Good morning........")
```

-->This function can always print same output for any name. Hello Prsoftwares Good morning......... Hello Premsofts Good morning......... Hello Prem Good morning.........

-->But we want to modify this function to provide different message if name is "Prem". We can do this with out touching wish() function by using decorator.

Ex:

```
1) def decor(func):
```

```
2)   def inner(name):
3)   if name=="Prem":
4)   print("Hello Prem how is your health.....")
5)   else:
6)   func(name)
7)   return inner
8) @decor
9) def wish(name):
10) print("Hello",name,"Good morning.........")
11) wish("Prsoftwares")
12) wish("Premsofts")
13) wish("Prem")
```

```
o/p:D:\pythonclasses>py test.py Hello Prsoftwares
Good morning......... Hello Premsofts Good
morning.........
Hello Prem Bad Morning.....
```

-->In the above program whenever we call wish() function automatically decor function will be executed.

Generators

-->Generator is a function which is responsible to generate a sequence of values.

-->We can write generator function just like ordinary functions, but it uses "yield" keyword to return values.

Yield------>Generator--------->A sequence of values

Ex:

```
def mygen():
yield 'A'
yield 'B'
yield 'C'
g=mygen()
print(type(g))
print(next(g))
print(next(g))
print(next(g))
```

```
o/p:D:\pythonclasses>py test.py
<class 'generator'> A
B C
```

Ex-2:

```
def countdown(num):
print("start countdown")
while (num>0):
yield num
num=num-1
values=countdown(5)
for i in values:
print(i)
```

```
o/p:D:\pythonclasses>py test.py start countdown
5
4
3
2
1
```

Ex-3: To generate first n numbers.

```
1) def firstn(num): 2)      i=1
3)   while i<=num:
4)   yield i
5)   i=i+1
6)   values=firstn(5)
7)   for x in values:
8)   print(x)
9)   print(type(values))
```

```
o/p:D:\pythonclasses>py test.py 1
2
3
4
5
```

Generators

-->Generator is a function which is responsible to generate a sequence of values.

-->We can write generator function just like ordinary functions, but it uses "yield" keyword to return values.

Yield------>Generator--------->A sequence of values
Ex:

```
def mygen():
yield 'A'
yield 'B'
yield 'C'
g=mygen()
print(type(g))
print(next(g))
print(next(g))
print(next(g))
```

```
o/p:D:\pythonclasses>py test.py
<class 'generator'> A
B C
```

Ex-2:

```
def countdown(num):
print("start countdown")
while (num>0):
yield num
num=num-1
values=countdown(5)
```

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

```
o/p:D:\pythonclasses>py test.py start countdown
5
4
3
2
1
```

Ex-3: To generate first n numbers.

```
def firstn(num):      i=1
while i<=num:
yield i
i=i+1
values=firstn(5)
for x in values:
print(x)
print(type(values))
```

```
o/p:D:\pythonclasses>py test.py 1
2
3
4
5
```

-->We can convert generator into list as follows:
values=first(5)
l=list(values) print(l)==>[1,2,3,4,5]
<class 'generator'>

Ex-4: To generate fibonacci numbers..........

```
1) def fib():
2)   a,b=0,1
3)   while True:
4)   yield a
5)   a,b=b,a+b
6)   for x in fib():
7)   if x>100:



8)   break
9)   print(x)
```

o/p:D:\pythonclasses>py test.py 0 1 1 2 3 5 8 13
21 34 55 89

Advantages of generator functions:
----
1).When compared with class level iterations, generators are very easy to use.
2).Improves memory utilization and performance.
3).Generators are best suitable for reading data

from large number of large files.
4).Generators work great for web scraping and crawling.
5).Memory utilization, bcoz we are not required to store all values at a time.

MODULES
-->A group of functions, variables and classes saved into a file, which is nothing but module.
-->Every python file(.py) acts as module.
-->Advantages of modules are: Reusability, Readability & Maintainability.

Ex: prsoftwaresmath.py

```
1) x=333
2)   def add(a,b):
3)   return a+b
4)   def product(a,b):
5)   return(a*b)
```

-->py prsoftwaresmath.py
-->prsoftwaresmath module contains one variable and 2 functions.
-->If we want to use memebers of the module in our program then we should import that module.
import modulename

-->We can access members by using module name. modulename.variable
modulename.function()
test.py:

```
1)  import prsoftwaresmath
2)  print(prsoftwaresmath.x)
3)  print(prsoftwaresmath.add(10,20))
4)  print(prsoftwaresmath.product(10,20))
```

```
o/p: 333
30
200
```

Note:
Whenever we are using a module in our program, for that module compiled file will be generated and stored in the hard disk permanently
    pycache folder

Renaming a module at the time of import(module aliasing):

Ex:
import prsoftwaresmath as m
-->Here prsoftwaresmath is original module name and m is alias name.

-->We can access memebrs by using alias name m.

test.py

```
1)   import prsoftwaresmath as m
2)   print(m.x)
3) print(m.add(10,20))
4) print(m.product(10,20))
```

from......import:

-->We can import a particular member of module by using from....import
-->The main advantage of this is we can access memebrs directly without using module name.

Ex:

```
1)   from prsoftwaresmath import x,add
2)   print(x)
3) print(add(10,20))
4) print(product(10,20))==>Name Error:name 'product' is not defined
```

-->We can import all the members of a module as follows from prsoftwaresmath import *
Ex:

```
1)   from prsoftwaresmath import *
2)   print(x)
3) print(add(10,20))
4) print(product(10,20))
```

member aliasing:

```
1)   from prsoftwaresmath import x as y,add as
sum
2)   print(y)
3) sum(10,20)
```

-->Once we defined as alias name, we should use alias name only and we should not use original name.

Ex:

```
1)   from prsoftwaresmath import x as y
2)   print(x)==>Name Error:name 'x' is not defined
```

Various possibilities of import:

```
1)   import modulename
```

2) import module1,module2,module3
3) import module1 as m
4) import module1 as m1,module2 as m2,module3 as m3
5) from module import memeber
6) from module import memeber1,member2,member3
7) from module import memeber1 as x
8) from module import *

Reloading a Module:

-->Bydefault module will be loaded only once eventhough we are importing multiple times.

Ex: module1.py
----
print("This is from module1")

test.py
----

1) import module1
2) import module1
3) import module1
4) import module1
5) print("This is test module")

o/p:
This is from module1 This is from test module

-->In the above program test module will be loaded only once eventhough we are importing multiple times.

-->The problem in this approach is after loading a module if it is updated outside then updated version of module1 is not available to our program.
-->We can solve this problem by reloading module explicitly based on our requirement.
-->We can reload by using reload() function of imp module. import imp
imp.reload(module1)

Ex:

```
1)   import time
2)   import module1
3)   print("program entering into sleeping state")
4)   time.sleep(30)
5)   import module1
6)   print("This is last line of the program")
```

o/p:D:\pythonclasses>py test.py This is from module1
program entering into sleeping state This is last line of the program

Ex:

```
1)  import time
2)  from imp import reload
3)  import module1
4)  print("program entering into sleeping state")
5)  time.sleep(30)
6)  reload(module1)
7)  print("program entering into sleeping state")
8)  time.sleep(30)
9)  reload(module1)
10) print("This is last line of the program")
```

o/p:

```
This is from module1
program entering into sleeping state This is first
module1
program entering into sleeping state This is
second module1
This is last line of the program
```

-->The main advantage of explicit module reloading is we can ensure that updated version is always available to our program.

Finding members of module by using dir() function:

-->Python provides inbuilt function dir() to list out all the members of current module or a specified module.

dir()==>To list out all members of current module.
dir(module name)==>To list out all the members of specified module.

Ex:test.py

```
1) x=333
2)   def add(a,b):
3)   return a+b
4)   print(dir())
```

o/p:D:\pythonclasses>py test.py
```
['  annotations  ', '  builtins  ', '  cached  ', '  doc  ', '
file  ', '  loader  ',        ' name ', ' package ', ' spec ',
'add', 'x']
```
Ex-2: To display members of a particular module.
prsoftwaresmath.py

```
1) x=333
2)   def add(a,b):
3)   return a+b
4)   def product(a,b):
```

```
5)   return(a*b)
```

test.py

```
1)   import prsoftwaresmath
2)   print(dir(prsoftwaresmath))
```

```
o/p:D:\pythonclasses>py test.py
[' builtins ', ' cached ', ' doc ', ' file ', ' loader ', '
name ',
' package ', ' spec ', 'add', 'product', 'x']
```

Note: For every module at the time of execution python interpreter will add some special properties automatically for internal use.

Ex: ' builtins ', ' cached ', ' doc ', ' file ', ' loader ', ' name ',   ' package ', ' spec ',
-->Based on our requirement we can access these properties also in our program. Ex: test.py
----

```
1)   print( builtins )
2)   print( cached )
3)   print( doc )
4)   print( file )
5)   print( loader )
```

```
6)   print( name )
7)   print( package )
8)   print( spec )
```

```
o/p:D:\pythonclasses>py test.py
<module 'builtins' (built-in)> None
None test.py
<_frozen_importlib_external.SourceFileLoader
object at 0x000001F562583CF8>
    main  None None
```

The special variable   name :

-->For every python program, a special variable name will be added internally.This variable stores information regarding whether the program is executed as an individual program or as a module.
-->If the program executed as an individual program then the value of this variable is
   main .
-->If the program executed as a module from some other program then the value of this variable is the name of the module where it is defined.
-->Hence by using this name   variable we can identify whether the program executed directly or as a module.

Ex: module1.py

```
1)   def f1():
2)   if name ==" main ":
3)   print("The code executed directly as a program")
4)   print("The value of name :", name )
5)   else:
6)   print("The code executed indirectly as a module from some other module")
7)   print("The value of name :",  name  ) 8) f1()
```

```
o/p:D:\pythonclasses>py module1.py The code executed directly as a program The value of name : main
```

test.py

```
1)   import module1
2)   module1.f1()
```

```
o/p:D:\pythonclasses>py test.py
The code executed indirectly as a module from some other module The value of name : module1
The code executed indirectly as a module from some other module The value of name : module1
```

working with math module:

-->Python provides inbuilt module math
-->This module defines several functions which can be used for mathematical operations.
-->The main important functions are:
sqrt(x), ceil(x), floor(x), log(x), sin(x), tan(x), fabs(x)

Ex:

```
1)   from math import *
2)   print(sqrt(4))      #2.0
3) print(ceil(10.1))    #11
4) print(floor(10.1))  #10
5) print(fabs(-10.6)) #10.6
6) print(fabs(10.6))   #10.6
```

Note: We can find help of any module by using help() function Ex:

```
1)   import math
2)   help(math)
```

Working with random module:

-->This module defines several functions to generate random numbers.
-->We can use these functions while developing games, in cryptography and to generate numbers

on fly for authentication.

1).random():

This function always generates some float value between 0 and 1(not inclusive).
0<x<1
Ex:

```
1)   from random import *
2)   for i in range(10):
3)   print(random())
```

```
o/p:D:\pythonclasses>py test.py
0.7690917847505055
0.12958902402812178
0.4865336117685294
0.21174369485166067
0.22533721686631736
0.24270745076560807
0.6156350105877338
0.3588237251403473
0.5609192891678808
0.46565274922504374
```

2).randint():

To generate random integer between two given numbers(inclusive) Ex:

```
1)   from random import *
2)   for i in range(10):




3)   print(randint(1,100))#Generates random int
values between 1 and 100(inclusive)
```

```
o/p:D:\pythonclasses>py test.py 70
88
20
73
16
8
27
72
80
71
```

-->Generate 5 6-digit random numbers Ex:

```
1)   from random import *
2)   for i in range(5):
3)
     print(randint(0,9),randint(0,9),randint(0,9),randi
nt(0,9),randint(0,9),randint (0,9),sep='"”)
```

```
o/p:D:\pythonclasses>py test.py 420863
352322
626559
```

```
689042
735553
```

3).uniform():

-->It returns random float values between 2-given numbers(not inclusive)

Ex:

```
1)   from random import *
2)   for i in range(10):
3)   print(uniform(1,10))
```

```
o/p:D:\pythonclasses>py test.py
1.456481671599132
8.262379808015648



8.294591177579873
2.4766196390802415
3.9929683121049644
6.908124318470733
5.113015507787097
1.9677692845675518
8.48400436311528
7.760067312991328
```

random()==>in between 0 and 1(not inclusive)
randint()==>in between z and y(incluseve)
uniform()==>in between x and y(not inclusive)

4).randrange([start],stop,[step]):

-->Returns a random number from the range.
start<=x<stop
-->start argument is optional and default value is 0.
-->step argument is optional and default value is 1.

randrange(10)==>generates a number from 0 to 9
randrange(1,11(==>generates a number from 1 to 10 randrange(1,11,2)==>generates a number from 1,3,5,7,9

Ex 1:

```
1) from random import *
2) for i in range(10):
3) print(randrange(10))
```

Ex 2:

```
1) from random import *
2) for i in range(10):
3) print(randrange(1,11))
```

Ex 3:

```
from random import *
for i in range(10):
print(randrange(1,11,2))
```

5).choice():

-->It wont return random number.
-->It will return a random object from the given list or tuple,str.

Ex 1:

```
from random import *
list=["sunny","bunny","chinny","vinny","pinny"]
for i in range(10):
print(choice(list))
```

Ex 2:

```
from random import *
list=("sunny","bunny","chinny","vinny","pinny")
for i in range(10):
print(choice(list))
```

Ex 3:

```
1)   from random import *
2)   list={"sunny","bunny","chinny","vinny","pinny"}
3)   for i in range(10):
4)   print(choice(list))
```

```
o/p:D:\pythonclasses>py test.py
TypeError: 'set' object does not support indexing
```

Ex 4:

```
1)   from random import *
2)   for i in range(10):
3)   print(choice("prsoftwares"))
```

Ex: To generate random pwd of 6-length where as 1,3,5 are aplhabate symbols and 2,4,6 are digits.

```
1)   from random import *
2)   for i in range(5):




3)
     print(chr(randint(65,65+25)),randint(0,9),chr(ra
ndint(65,65+25)),randint(0,9
```

```
),chr(randint(65,65+25)),randint(0,9),sep='')
```

```
o/p:D:\pythonclasses>py test.py L0Z4J6
X1M8N6 A8L2U1 V7O2Z5 S9U2S1
```

PACKAGES

-->It is an encapsulation mechanism to group related modules into a single unit.
-->Package is nothing but folder or directory which represents collection of python modules.
-->Any folder or directory contains init .py file, is considered as a python package. This can be empty.
-->A package can contains sub packages also.
-->Python 3.3 has implicitly namespace packages that allows to create a package without
    init .py file.

-->The main advantages of package statements are: 1).We can resolve naming conflicts. 2).We can identify our components uniquely 3).It improves modularity of the application.

Ex1:

D:\pythonclasses

```
|--test.py
|--pack1
|-module1.py
|- init .py
```

init .py:
----
empty file.

module1.py:

```
def f1():
print("Hello this is from module1 present in pack1")
```

test.py(version-1):

```
import pack1.module1 pack1.modul1.f1()
```

o/p: Hello this is from module1 present in pack1

test.py(version-2):

```
from pack1.module1 import f1 f1()
```

Ex 2:

```
D:\pythonclasses
|--test.py
```

|--com

init .py:

|--module1.py
|-- init .py
|--prsoft
|--module2.py
|-- init .py

empty file

module1.py:
----
def f1():
print("Hello this is from module1 present in com")

module2.py:
----
def f2():
print("Hello this is from module 2 present in com.prsoft")

test.py:

----

o/p:

```
Hello this is from module1 present in com
Hello this is from module 2 present in com.prsoft
```