**LANGUAGE FUNDAMENTALS**

* Introduction
* Features of python- simple

open source

High Level Programming language

platform independent

portability

dynamically typed

both procedure oriented and OOP

interpreted

Extensible(we can use other languages programs in python)

Embedded(we can use python programs in other language programs)

Extensible library

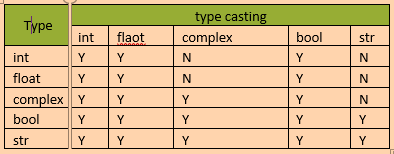
* Limitations of python
* Identifiers
* Reserved words
* Data types- int

Float

Complex

Bool

Str

Bytes

Bytearray

Range

List

Tuple

Set

Frozen set

Dict

* Slicing of strings
* Type casting
* Fundamental data types vs immutability

**OPERATORS**

* Arthimetic (+,\*,-,/)
* Relational(<,>,<=,>=)
* Equality(=)
* Special(identity (is, is not), membership(in, not in))
* Ternary operators

**FLOW CONTROLS**

* Conditional(if, if-elif-else, if-else)
* Iterative(for, while)- diff
* Transfer(break, continue, pass)

**Diff b/w Iterators and iterables**

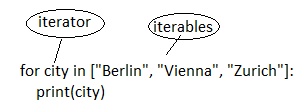


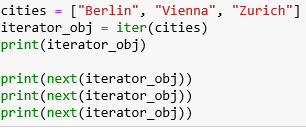
In one persepective they are the same

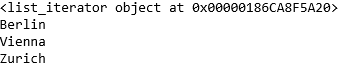
Every iterator is also an iterable, but not every iterable is an iterator.

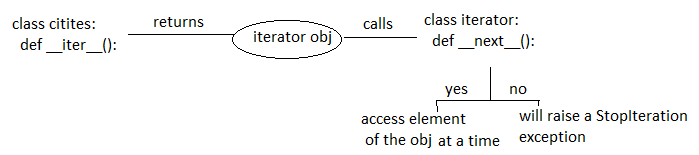
E.g. a list is iterable but a list is not an iterator!

The for statement calls iter() on the object ( which should be a so-called container object), which it is supposed to loop over. If this call is successful, the iter call will return return an iterator object that defines the method \_\_next\_\_() which accesses elements of the object one at a time. The \_\_next\_\_() method will raise a StopIteration exception, if there are no further elements available. The for loop whill terminate as soon as it catches a StopIteration exception. You can call the \_\_next\_\_() method using the next() built-in function.









**STRING DATATYPE AND FUNCTIONS**

* **String**
* How to define a multi line String literals
* Accessing of String characters
* Using index- s[0],s[-1]
* Using slice operator- s[start : stop : step]
* Mathematical operators( +, \*)
* Len()
* Checking membership(substring **in, not in** String)
* Comparison of String**(<, <=, >=, >, ==, !=)**
* Removing spaces from the String**( rstrip(), lstrip(), strip() )**
* Finding subStrings
* Forward direction(**find()** returns index value if not find substring returns -1)

(**index()** returns index value if not find substring returns value error)

* Backward direction**( rfind()-**It returns highest index of the substring if not returns -1

**rindex()-**It returns highest index of the substring if not returns valuerror **)**

* Counting substring**( count(subsring), count(substring, start. stop)** )
* Replacing string**( replace(old, new))**
* Splitting of String( s.**split(‘-’)**)
* Joining of Strings**( ‘-’.join(list or tuple))**
* Changing case**(upper(), lower(), swapcase(), title(), capitalize())**
* Starts with and ends with(**startswith()** and **endswith()** returns bool)
* To check the type of characters present in a string
* **(isalnum(), isalpha(), isdigit(), islower(), isupper(), istitle(), isspace())**

**List Data Structure and its functions:**

Insertion order is preserved, duplicates and heterogeneous objects are allowed

It is dynamic

Index based (positive and negative)

* Creation of list objects
* Empty list ( **list=[]** )
* If we know the elements( **lst=[10,20,30,40]** )
* Dynamic input ( **eval(input(“Enter list”))** )
* list() function ( **list(range(0,10,2))** )
* split() function
* Accessing of elements in list
* Using index- list[0], list[-1]
* Using slice operator- list[start : stop : step]
* List Vs Mutability
* Transversing the elements of list
* By using while loop

n = [10,20,30,40,50,60,70,80,90]

i=0

while i<len(n):

print(n[i])

i=i+1

* By using for loop
* To display only Even numbers
* To display elements by index wise

Ex:- lst=[‘A’,’B’,’C’,’D’]

x=len(lst)

For i in range(x):

Print(lst[i],”positive index:”,i,”negative index:”,i-x)

* Important functions of list
* To get the Info of list( **len(), count(), index())**

len()—length of list

count() - no. of occurences

index() - returns the index of 1st occurrence(if not valueError)

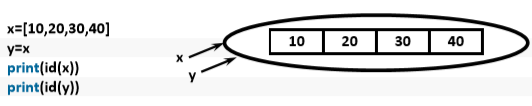
* Manipulating elements of list( **append(), insert(1,888), extend(), remove(), pop()** )

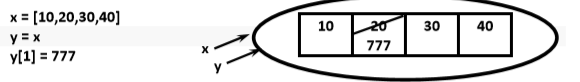
Remove()—we can remove specified item from the list. If they are repeatedly it removes

1st occurrence(If specified element not present ValueError )

Pop() -- it removes and returns the last element of list

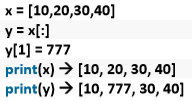
(If list is empty it raises index error)

* Diff between append and insert
* Ordering elements of list( **reverse(), sort(reverse=true)** )
* Aliasing and cloning of objetcs
* Aliasing 

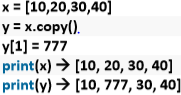
problem 

* Cloning

By using slice operator



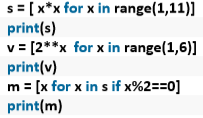
By using copy() function



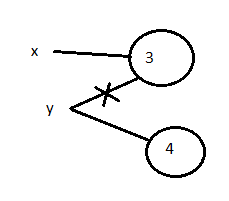
* Diff between ‘=’ and ‘copy()’
* Mathematical operators for list objects
* Concatenation(+)
* Repetition(\*)
* Comparing the list objects**( ==, !=, <, <=, >, >= )**
* Membership operators( **in, not in** )
* Clear() to clear all the elements in the list
* Nested lists n=[10,20,[30,40]]
* List comprehensions

COMPACT way of Creating list objects from iterable objects

**List = [expression for items in list if condition]**



**Shallow and deep copying: it is only relevant for compound objects i.e. objects containing other objects, like lists or class instances**

x = 3

y = x

print(id(x), id(y)) 🡪 9251744 9251744

y = 4

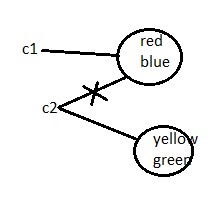
print(id(x), id(y)) 🡪 9251744 9251776

print(x,y)

3 4

**Copying list items:**

c1 = ["red", "blue"]

****c2 = c1

print(c1) 🡪 ['red', 'blue']

print(c2) 🡪 ['red', 'blue']

print(id(c1),id(c2)) 🡪 43444416 43444416

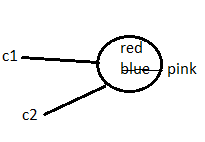
c2 = ["yellow", "green"]

print(c1) 🡪 ['red', 'blue']

print(c2) 🡪 [‘yellow’, ‘green’]

print(id(c1),id(c2)) 🡪 43444416 43444200

Now we have to examine, what will happen, if we change just one element of the list of c2 or c1:



c1 = ["red", "blue"]

c2 = c1

print(id(c1),id(c2)) 🡪 14603760 14603760

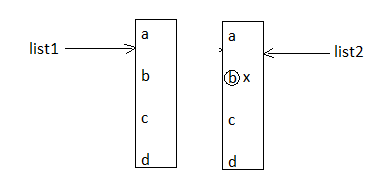
c2[1] = "pink"

print(id(c1),id(c2)) 🡪 14603760 14603760

print(c1) 🡪 ['red', 'pink']

print(c2) 🡪 ['red', 'pink']

**copy with slice operator:**



list1 = ['a','b','c','d']

list2 = list1[:]

print(id(list1)) 🡪 2425352

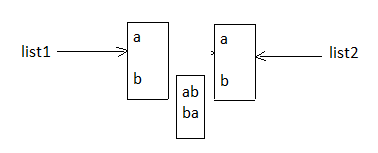
print(id(list2)) 🡪 2425416

list2[1] = 'x'

print(list2) 🡪 ['a', 'x', 'c', 'd']

print(list1) 🡪 ['a', 'b', 'c', 'd']

a list contains sublists : The sublists are not copied but only the references to the sublists

lst1 = ['a','b',['ab','ba']]

lst2 = lst1[:]

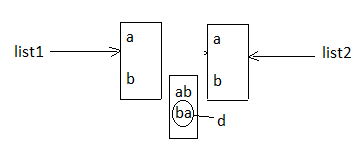
print(id(lst1[2])) 🡪 12345

print(id(lst2[2])) 🡪 12345

lst2[0] = 'c'

print(lst1) 🡪 ['a', 'b', ['ab', 'ba']]

print(lst2) 🡪 ['c', 'b', ['ab', 'ba']]



lst2[2][1] = 'd'

print(lst1) 🡪 ['a', 'b', ['ab', 'd']]

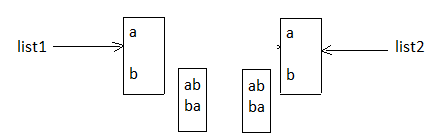
print(lst2) 🡪 ['c', 'b', ['ab', 'd']]

**Using Module Copy :**

**from** copy **import** deepcopy

lst1 = ['a','b',['ab','ba']]

lst2 = deepcopy(lst1)

print(lst1) 🡪 ['a', 'b', ['ab', 'ba']]

print(lst2) 🡪 ['a', 'b', ['ab', 'ba']]

print(id(lst1)) 🡪 1234

print(id(lst2)) 🡪 4567

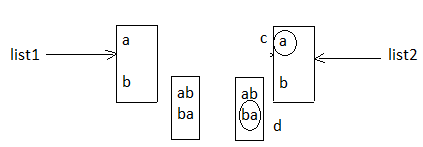
print(id(lst1[0])) 🡪 8910

print(id(lst2[0])) 🡪 8910

print(id(lst2[2])) 🡪 9177

print(id(lst1[2])) 🡪 2565

by using the id function that the sublist has been copied, because id(lst2[2]) is different from id(lst1[2]). An interesting fact is that the strings are not copied: lst1[0] and lst2[0] reference the same string. This is true for lst1[1] and lst2[1]



lst2[2][1] = "d"

lst2[0] = "c"

print(lst1) 🡪 ['a', 'b', ['ab', 'ba']]

print(lst2) 🡪 ['c', 'b', ['ab', 'd']]

**Tuple Data Structure and its functions:**

Same as list except tuple is immutable

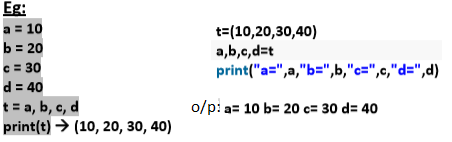
t=10,20,30,40

print(t) 🡪 (10, 20, 30, 40)

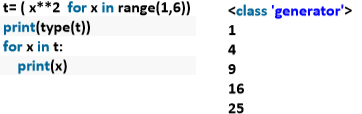
print(type(t)) 🡪 <class 'tuple'>

**NOTE:- t=(10)**

* Tuple creation
* Empty tuple( **t=()** )
* t=10, or t=(10,)
* t=10,20,30
* tuple() by using tuple function
* Accessing elements of tuple
* By using index(**t[0] it returns element, t[100] index error out of range** )
* Slice operator( **t[2:5]** )
* Tuple Vs immutability( **t[0] = 100** )
* Mathematical operators for tuple ( **+,\***)
* Imp functions of tuple
* len()
* count() ex: t=(10,20,30,10) print(t.count(10))🡪 2
* index() returns the index of 1st occurrence else return value error
* sorted() DNSO
* min() & max()
* cmp() it compares the both tuples elements are equal returns 0, < means -1
* tuple packing and unpacking



* Tuple comprehension : not supported by python



**Diff between list and tuple**

**Set Data Structure and its functions:**

* Duplicates are not allowed, insertion order not preserved(we can sort the elements)
* Indexing and slicing not allowed for set, heterogenius elements are allowed
* Creation
* S={10,20,30,40}

Print(s) 🡪 {40,10,20,30}

Type(s) 🡪 <class ‘set’>

* Set()

S={} treated as dictionary

* Imp functions of set
* add(x) 🡪 to add one item to set
* update(x,y,z) 🡪 to add multiple item to set

o/p:- {0,1,2,3,4,40,10,50,20,60,30}

**Diff beween add() and update()**

* copy() 🡪 clone that object and returns that copy object

o/p:- {10, 20, 30}

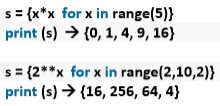
* pop() 🡪 it removes and return some random element from set

 o/p:- 40

* remove(x) 🡪 It removes the specific element from set else if not KEY ERROR

o/p:- {70,40,50,60}

* discard(x) 🡪 It removes the specific element from set else if not will not get any error
* clear() 🡪 to remove all elements from the set
* mathematical operations on set
* union() 🡪 x.union(y) or x/y
* intersection() 🡪 x.intersection(y) or x&y
* difference() 🡪 x.difference(y) or x-y
* symmetric\_difference() 🡪 x.symmetric\_difference(y) or x^y
* membership operators( **in, not in**) returns T/F
* set comprehensions: set comprehension is possible



**Dict Data Structure:** If we want to represent a group of objects as K-V pairs

Duplicate keys are not allowed but values can be duplicated

Heterogeneous objects are allowed for both K-V

Insertion order is not preserved

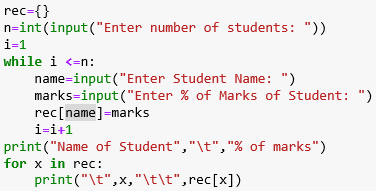
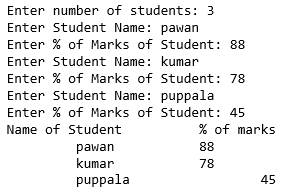
Mutable, dynamic, index & slicing are not applicable

* Creation of Dictionary
* d = {}
* d = dict() by using function
* we can add K-V by **d[100]=”pavan”, d[200]=”kumar”**
* if we know the data **d={100:”pavan”, 200:”kumar”}**
* Accessing data from dictionary
* We can access by using keys **print(d[100])**

If the specified key is not available then we will get **KeyError**

Before we accessing to check it is available or not by using **has\_key()** or **in**

**d.has\_key(400)-** returns 1 if key is avail otherwise returns 0



* Update

d[key] = value if the key is available then update the value

if not key is available a new entry will be added to the dict

* Delete
* del d[key] if key is not available then we will get KeyError
* d.clear() to remove all entries from the dictionary though we can access
* del d it delete total dictionary and we cannot access the d
* imp functions in dict
* dict() to create dictionary
* len() returns number of items in the dictionary
* clear() to remove all elements from dictionary
* get() to get the value associated with the key

d.get(key) or d.get(key, defaultvalue)

- pop()

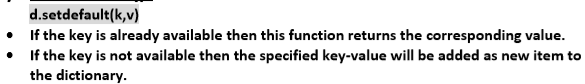
it removes the associated with specified key and return the corresponding value

if not will get key error

* popitem()

It removes an arbitrary item(key-value) from the dictionaty and returns it

* keys() It returns all keys associated eith dictionary
* values() It returns all values associated with the dictionary
* items() It returns list of tuples representing key-value pairs. [(k,v),(k,v),(k,v)]
* copy() To create exactly duplicate dictionary (cloned copy)
* setdefault()



* update()

d.update(x) All items present in the dictionary x will be added to dictionary d

* Dictionary Comprehension: Comprehension concept applicable for dictionaries also.

squares={x:x\*x for x in range(1,6)}

print(squares)

doubles={x:2\*x for x in range(1,6)}

print(doubles)

Output {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 no need to write these statements repeatedly. We can define the set of statements in a single unit, this unit is called function

Advantages:-

Code reusability

It enhances the comprehensibility & quality of the program

It also lowers the cost for development & maintenance of the software

Functions are two types

* Built in or pre-defined functions(along with python s/w)
* User- defined functions(which are developed by programmer acc to requirement)

Syntax:-

def function-name(Parameter list):

statements, i.e the function body

return value

note:- we use two keywords while defining a function def(mandatory), return(optional)

* Write a function to print hello

Test.py

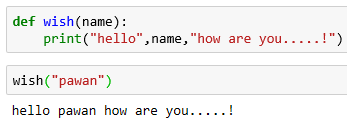
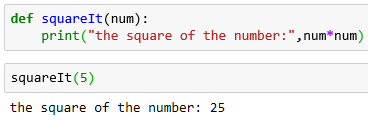
Def wish():

Print("hello good morning”)

wish()- function calling

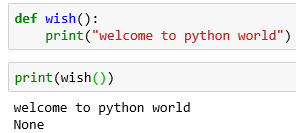
**parameters**:- if a function contain parameters at the time of calling a function, it is

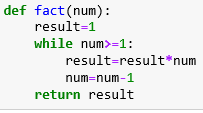
mandatory to provide the value O.W will get TypeError



**Return:-**

* A function can take a input value and sometimes it returns output to the caller
* If we are not writing return statement then default return value is None



* W.A.F to accept two number and return the sum
* W.A.F to check the given number is even or odd
* W.A.F to find the factorial of the given number
* W.A.F to return multiple values(tuple, unpack it)

**Type of Arguments:**

Def fun(a, b): here a, b are formal arguments

10, 20 are actual arguments

fun(10, 20)

**Four types of arguments**

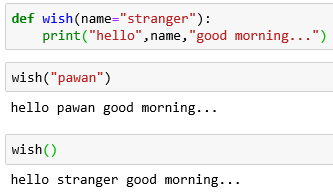
**Positional**(pass in order if not o/p will vary, no. of args must be matched if not error)

**Keyword**(we can pass the value by parameter name, no order, no. of args must match)

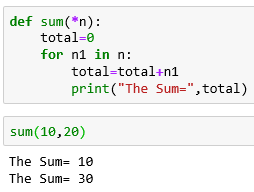
Note:- if we take both positional & keyword args at a time, write positional first later

Keyword if not it will raise syntax error

**Default or optional**:- sometimes we will provide default agrs to positional args



Note: After default arguments we should not take non-default arguments(syntax error)

**Variable or arbitary length arguments:-**

* we can pass variable no. of arguments to our function
* Symbol \*
* We can call by passing any number of arguments including 0
* Internally these values represented in the form of tuple
* Before variable len args
* We can mix variable length arguments with positional

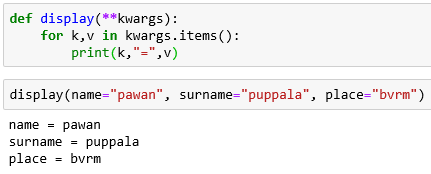
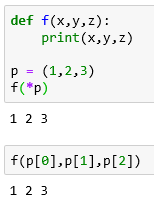
Arguments

* After variable length argument, if we are taking any other

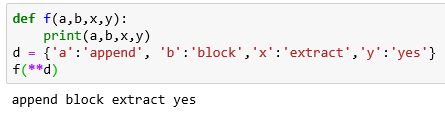
arguments then we should provide values as keyword arguments.

f1("A","B",n1=10)

Note : We can declare key word variable length arguments also.



Double Asterisk in function calls



Function Vs Module Vs Library

A group of lines 🡪 function

A group of functions 🡪 module

A group of modules 🡪 library

**Types of Variables:**

* Global(a variable which are declare outside of function, it can access in all

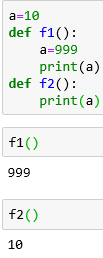
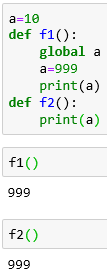
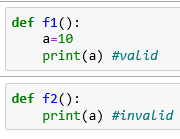
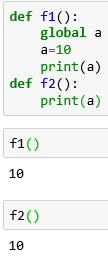
functions of that module)

* local(a variable which are declare inside of function, we cannot access outside

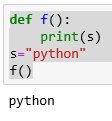
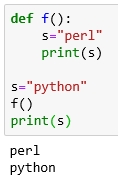
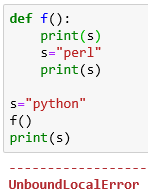
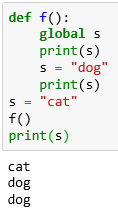
of the function of that module)

Global keyword:

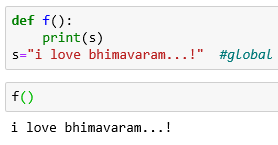
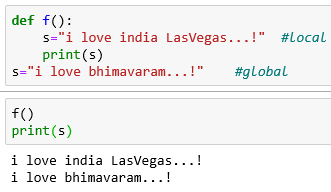
* to declare global variable inside the function
* to make global variable avail to all functions in that module



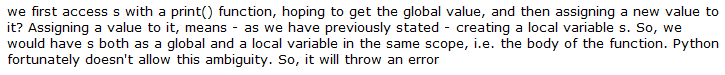
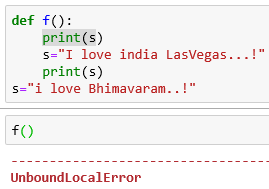
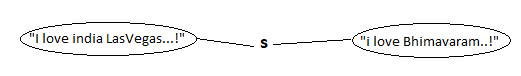
---------------------------------------------------------------------------------------------



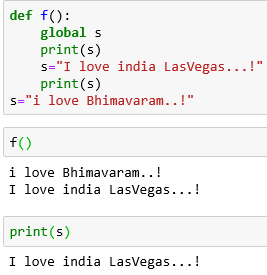
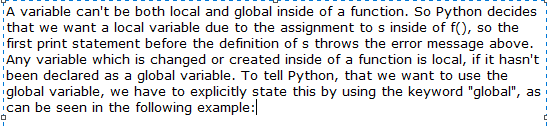
**-------------------------------------------------------------------------------------------------**

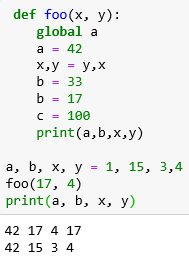


**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

****

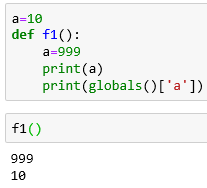
**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***





Note:- If global variable and local variable having the same name then

we can access global variable inside a function as follows



* **recursive function**
* a function that calls itself

reduces len of code and readability, solve complex problems

* **Normal function**
* We can define by using ‘def’ keyword

def squareIt(n):

return n\*n

* **anonymous or lambda function**
* sometime we will declare a function without any name

main use is just for instant use, we do not write return statement

syn: lambda arg\_list : expression

s = lambda n:n\*n, s=lambda a,b:a+b, s=lambda a,b: a if a>b else b

Note: sometimes we can pass function as a argument to another function in such cases

lambda functions are best choice

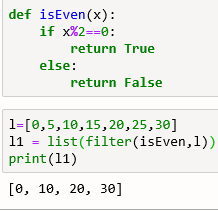
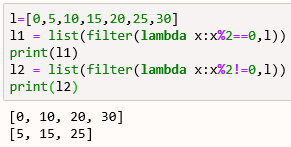
Ex:- def fun(lambda a, b: a if a>b else b)

Generally we use lambda functions very commonly with filter(), map(), reduce() because these functions expect function as a argument

**Filer():-** to filter values from the given sequence(list or tuple or string)

Filter(function, sequence)

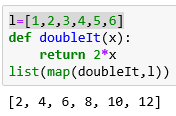
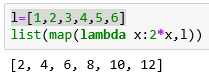
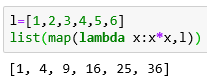
Without lambda with lambda



**Map():-** for every element in the given sequence apply some functionality

map(function, sequence)

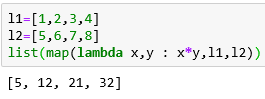
Without lambda with lambda to find the square of given num



Note:- We can apply map() function on multiple lists also. But make sure all list should have same length

Syntax: map(lambda x, y : x\*y,l1,l2))

x is from l1 and y is from l2

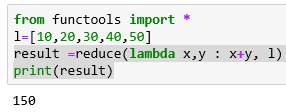
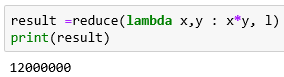


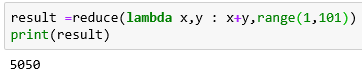
**Reduce():-** it reduces sequence of elements into a single element by applying the specified

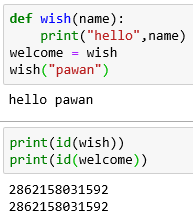
function.

* reduce() present in functools module

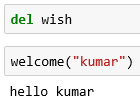
Syn:- reduce(function, sequence)



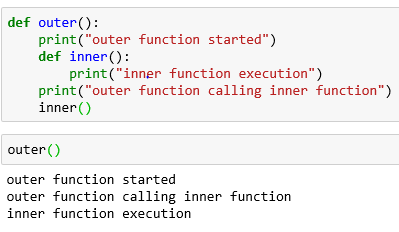


* Everything is an object in python
* Function aliasing
* If we delete one name still we can access that function

By using the alias name



**Nested functions:-** we can declare a function inside a function

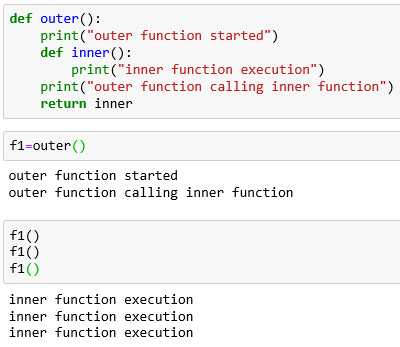


Here we cannot call inner()

directly from outer() because

it is local to Outer()

A function can return another function



**Diff b/w f1=outer🡪aliasing & f1=outer()🡪 assigning**

**Modules:-**

A group of classes, functions, variables saved to file(.py is called module)

If you want to access the members of a module we should import the module then with modulename.function()

Modulename.variable

Calculator.py

def add(x,y):

print("addition of two numbers:", x+y)

def sub(x,y):

print("subtraction of two numbers:", x-y)

def mul(x,y):

print("multiplication of two numbers:", x\*y)

def div(x,y):

print("divison of two numbers:", x/y)

test.py

**import** calculator as c

c.add(5,4)

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.

* **Module aliasing:-**
* **import calculator as c**
* **from calculator import add, sub**
* **various possibilities:-**
* **import** module
* **import** mod1, mod2, mod3
* **import** mod as m
* **import** mod1 as m1, mod2 as m2, mod3 as m3
* **from** module **import** member(function, variable)
* **from** module **import** mem1, mem2, mem3
* **from** module **import** \*
* **Member aliasing:-**
* **From calculator import add as a, sub as s, mul as m**

**Note:-** once we given the alias name we should call with alias name only

* **Reloading a module:-**

If we import the module multiple times it will be loaded only once

module1.py:

print("This is from module1")

\*To find all members in a module python provides dir() & dir(modulename)

dir()🡪 To list out all members of current module

dir(moduleName) 🡪 To list out all members of specified module

test.py:

**import** module1

**import** module1

**import** module1

print("This is test module")

Output:-This is from module1

This is test module

🡪A small drawback is after loading a module if it is updated outside then the updated

version of module1 is not available to our program

🡪To overcome we will reload the module explicitly

import imp

imp.reload(module1)

test.py:

**import** module1

**import** module1

**from** imp **import** reload

reload(module1)

reload(module1)

reload(module1)

print("This is test module")

In the above program module1 will be loaded 4 times in that 1 time by default and 3 times explicitly. In this case output is

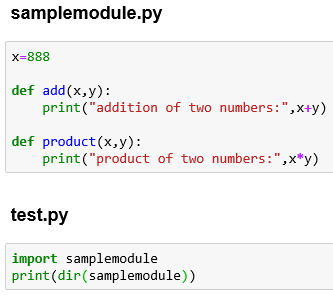
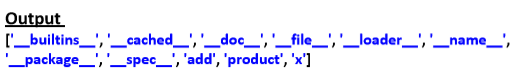
This is from module1

This is from module1

This is from module1

This is from module1

This is test module



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

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

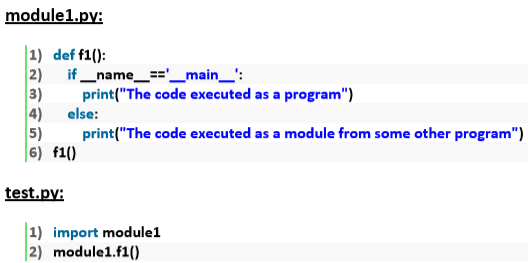
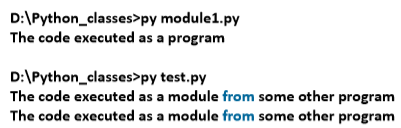
\_\_main\_\_

🡪If the program executed as a module from some other program then the value of this

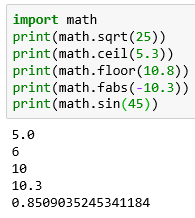
variable is the name of module where it is defined.

🡪Hence by using this \_\_name\_\_ variable we can identify whether the program executed

directly or as a module.



**Math module:-**

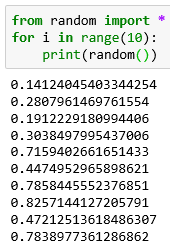
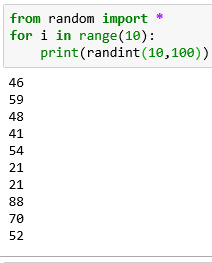
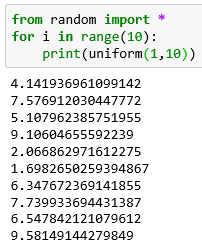


**Help(math) or dir(math)- it give total functions in math module**

**Random module:-** it generates random numbers these functions are helpful while developing the games, cryptpgraph and authentication

**random():- 0<x<1 randint():-** to generate **uniform():-** random float

numbers b/w to given numbers values b/w two numbers



**randrange([start], stop, [step])**

🡪Returns a random number from range

🡪start <= x < stop

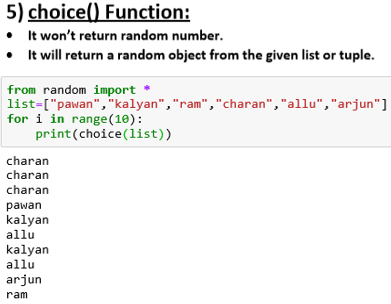
🡪start argument is optional and default value is 0

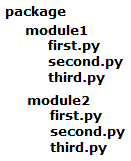
🡪step argument is optional and default value is 1

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



**Packages**:- A package contains no.of modules

A module contains no. of files

A file contains members

**File handling:-** file concept is used to store our data permanently for future purpose

Types of files:

* Text files:- To store character data ex: abc.txt
* Binary files:- To store binary data like images, videos, audio files etc

**Opening a File:-** before performing any operation(like read or write) on file, first we have to open that file

At the time of open we have specify the mode, which represents purpose of opening file

Syntax:- f = open(filename, mode)

Modes:-

r(read operation)🡪open an existing file. the file pointer is positioned at beginning of the file. If

file is not present FileNotFoundError. This is default mode

w(write operation) 🡪 open an existing file. If file already contains data then it will override.

If the file is not available then it will create that file

a(append operation)🡪 open an existing file. It will not override the existing data.

If the file is not available then it will create that file

r+(read & write data)🡪 the previous data in the file will not be deleted. The file pointer is placed at

the beginning of the file

w+( write & read data)🡪it will override the existing data

a+(append & read) 🡪 it will not override the existing data

x🡪To open a file in exclusive creation mode for write operation. If the file already exists then we will get FileExistsError**.**

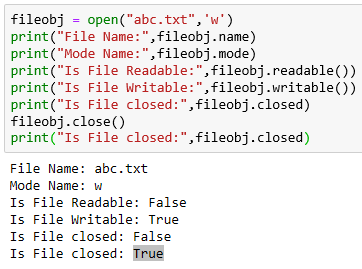
**Note: All the above modes are applicable for text files. If the above modes suffixed with 'b' then these represents for binary files.**

**Eg: rb,wb,ab,r+b,w+b,a+b,xb**

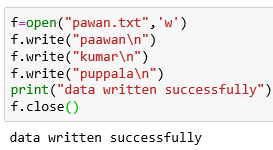
**Closing a file:-** f.close()

**Properties of file object:** file object **=** open(filename, mode)

* name🡪 name of opened file
* mode🡪 mode in which the file is opened
* closed🡪 returns bool value indicates that the file is closed or not
* readable()🡪 returns bool value indicates that the file is readable or not
* writeable()🡪 returns bool value indicates that the file is writeable or not



**Writing data to text files:-**

We can write char data to text by using two methods

* write(str)
* writelines(list of lines)

a new file is created and the data inside

of the file is paawan

kumar

puppala

**Note:** while writing data by using write() methods,

compulsory we have to provide line seperator(\n),

otherwise total data should be written to a single line.

**Note**: In the above program, data present in the file

will be overridden everytime if we run the program.

Instead of overriding if we want append operation then

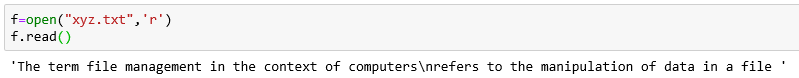
we should open the file as follows.

f = open("abcd.txt","a")

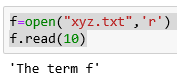
**Reading character data from text files:-**

We can read char data from text file by using four methods

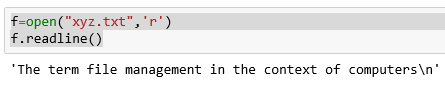
read() 🡪 To read total data from the file



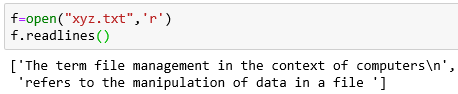
read(n) 🡪 To read 'n' characters from the file



readline() 🡪 To read only one line



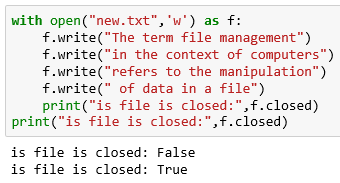
readlines()🡪 To read all lines into a list



**With statement:-**

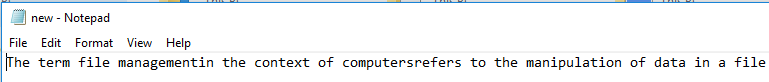
The with statement can be used while opening a file. We can use this to group file operation statements within a block.

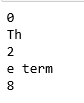
The advantage of with statement is it will take care closing of file, after completing all operations automatically even in the case of exceptions also, and we are not required to close explicitly.



**Seek() & tell():-**

tell()🡪 it returns the current position of cursor from beginning of the file

file:- 



Seek()🡪 We can use seek() method to move cursor(file pointer) to specified location

Syn**:- f.seek(offset, fromwhere)**

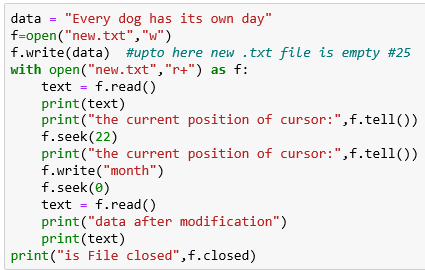
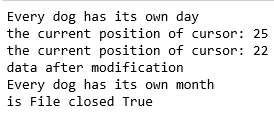
offset represents the number of positions

The allowed values for second attribute(from where) are

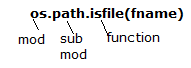
0---->From beginning of file(default value)-- python3 supports only 0

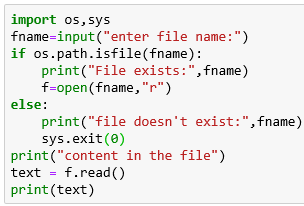
1---->From current position

2--->From end of the file



**Note:- How to check a particular file exists or not?**



**Q. Write a program to check whether the given file**

**exists or not. If it is available then print its content?**

**Q. Program to print the number of lines, words and**

**characters present in the given file?**

**Note: sys.exit(0) ===>To exit system without executing**

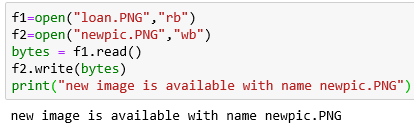
**rest of the program. argument represents**

**status code . 0 means normal termination**

**and it is the default value.**

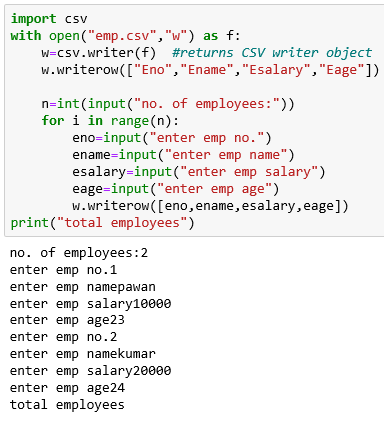
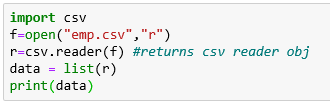
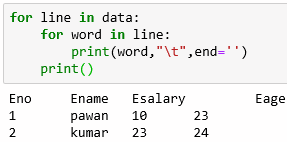
**Handling Binary Data:-**

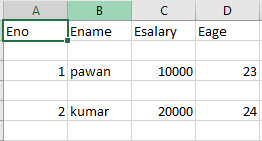
**Program to read image file and write to a new image file?**



**Handling the CSV files:-**

Writing data to csv file: newline=’’ Reading Data from csv file:





**Zipping and Unzipping Files:**

Advantage: 1. To improve memory utilization

2. We can reduce transport time 3. We can improve performance.

To perform zip and unzip operations, Python contains one in-bulit module zip file. This module contains a class : ZipFile

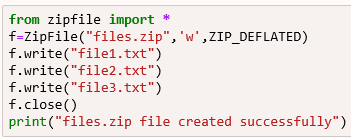
**To create Zip file:**

We have to create ZipFile class object with name of the zip file, mode and constant ZIP\_DEFLATED. This constant represents we are creating zip file.

**f = ZipFile("files.zip", "w", "ZIP\_DEFLATED")**

Once we create ZipFile object,we can add files by using write() method.

**f.write(filename)**



**To perform unzip operation:**

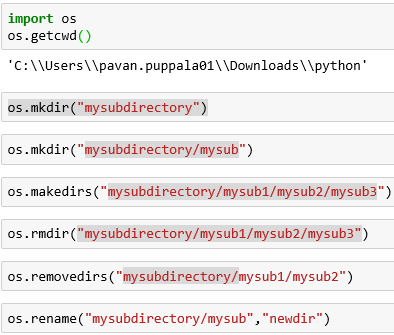
**We have to create ZipFile object as follows**

**f = ZipFile("files.zip", "r", ZIP\_STORED)**

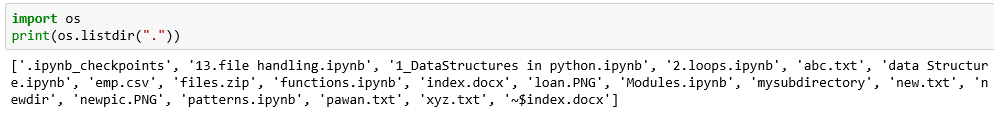
**ZIP\_STORED represents unzip operation. This is default value and hence we are not required to specify. Once we created ZipFile object for unzip operation,we can get all file names present in that zip file by using namelist() method.**

**names = f.namelist()**





**Q8. To know contents of directory:**



**Q9. To know contents of directory including sub directories:**

**os.walk(path,topdown=True,onerror=None, followlinks=False)**

**It returns an Iterator object whose contents can be displayed by using for loop**



**os.walk("directoryname")**

**Q. What is the difference between listdir() and walk() functions?**

In the case of listdir(), we will get contents of specified directory but not sub directory contents. But in the case of walk() function we will get contents of specified directory and its sub directories also.

**Running Other programs from Python program:**

os module contains system() function to run programs and commands.

os.system("commad string")

The argument is any command which is executing from DOS.

Eg: import os

os.system("dir \*.py")

os.system("py abc.py")

**How to get information about a File:**

We can get statistics of a file like size, last accessed time,last modified time etc by using stat() function of os module.

**stats = os.stat("abc.txt")**

The statistics of a file includes the following parameters:

st\_mode==>Protection Bits

st\_ino==>Inode number

st\_dev===>device

st\_nlink===>no of hard links

st\_uid===>userid of owner

st\_gid==>group id of owner

st\_size===>size of file in bytes

st\_atime==>Time of most recent access

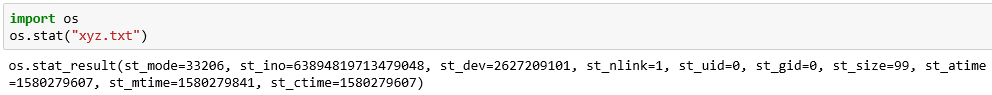
st\_mtime==>Time of Most recent modification

st\_ctime==> Time of Most recent meta data change

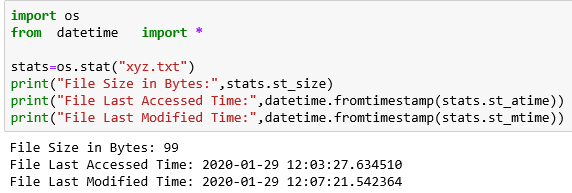
Note:

st\_atime, st\_mtime and st\_ctime returns the time as number of milli seconds since Jan 1st 1970 ,12:00AM. By using datetime module fromtimestamp() function,we can get exact date and time.

**Q. To print all statistics of file abc.txt:**



**Q. To print specified properties:**



**Pickling and Unpickling of Objects:**

Sometimes we have to write total state of object to the file and we have to read total object from the file.

The process of writing state of object to the file is called pickling and the process of reading state of an object from the file is called unpickling.

We can implement pickling and unpickling by using pickle module of Python.

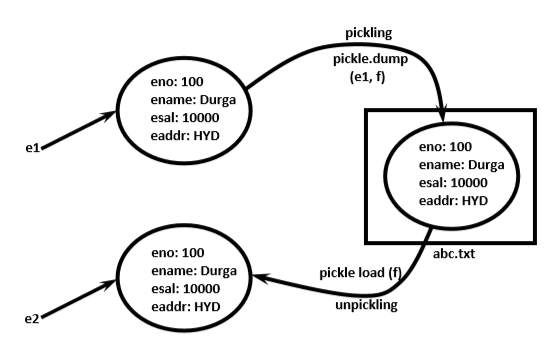
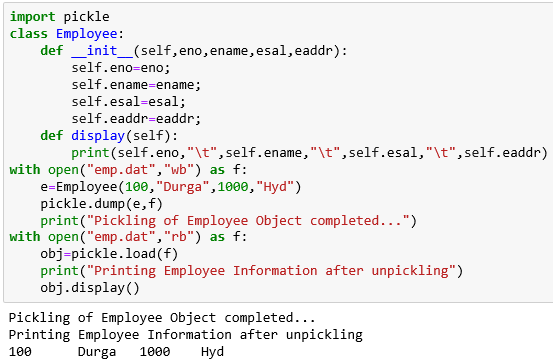
pickle module contains dump() function to perform pickling.

**pickle.dump(object,file)**

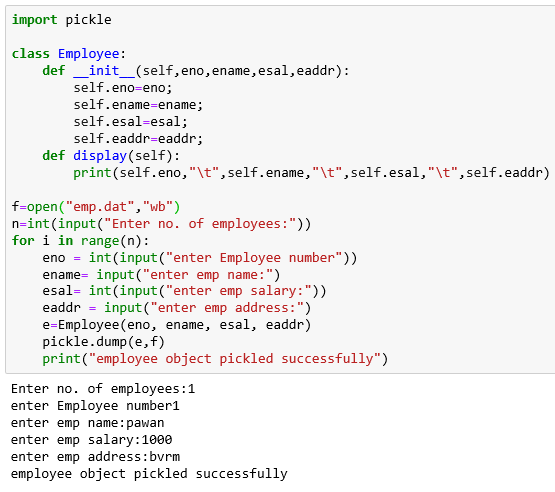
pickle module contains load() function to perform unpickling

**obj=pickle.load(file)**

**Writing and Reading State of object by**

 **using pickle Module:**

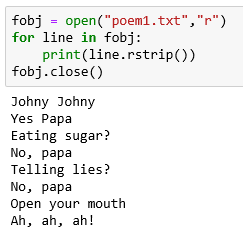
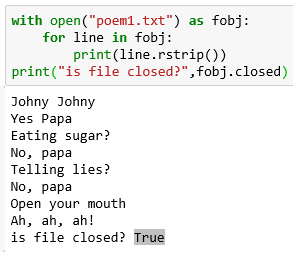
**Pickling**



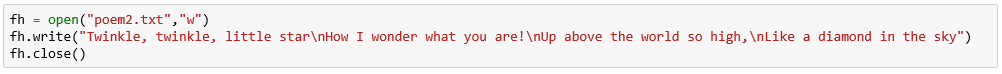
**Unpickling**



**Reading of a file**



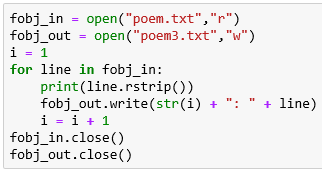
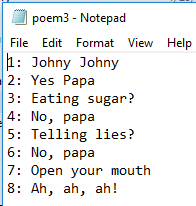
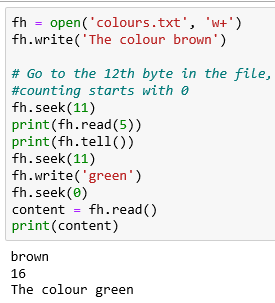
**Writing of a file**



**With statement: at end we need not to close file**



**Read & write Simultaneously Read & Write to the same file**



"Pickling" denotes the process which converts a Python object hierarchy into a byte stream, and "unpickling" on the other hand is the inverse operation, i.e. the byte stream is converted back into an object hierarchy. What we call pickling (and unpickling) is also known as "serialization" or "flattening" a data structure.

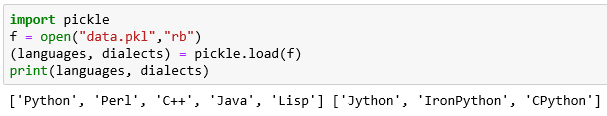


The file data.pkl can be read in again by Python in the same or another session or by a different program:

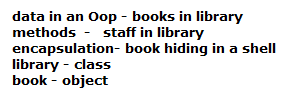




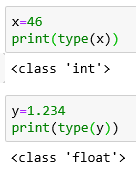
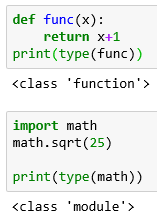
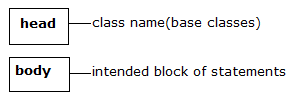
The pickled data from the previous example, - i.e. the data which we have written to the file data.pkl, - can be separated into two lists again, when we read back in again the data:



Oops: for code reusability and security purpose we use Oops concepts

* Class
* Object
* 4 major principles of object - orientation
* Encapsulation
* Data Abstraction
* Polymorphism
* Inheritance

🡪Everything is a class in python



🡪Class contains two parts

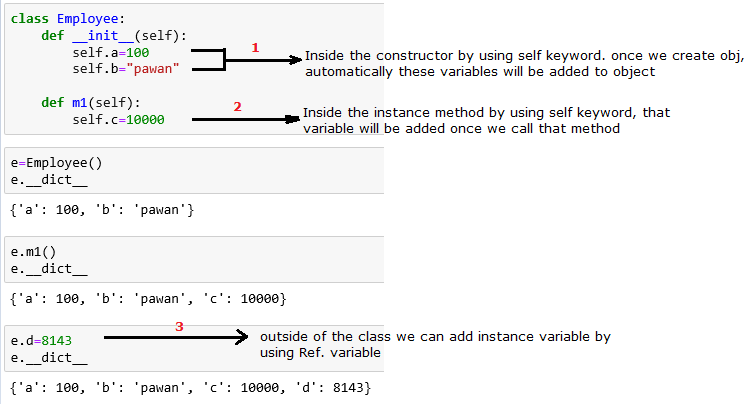
Class:- To create objects we required some Model or Plan or Blue print, which is nothing but class.

Types of variables:

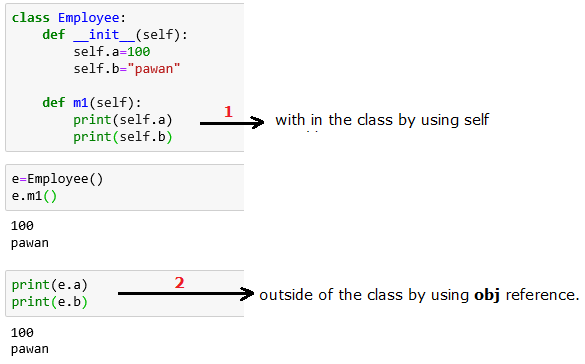
* Instance (Object level) variable
* Static(Class level) variable
* Local(Method level) variable

Instance:- if the value of variable is varied from object to object

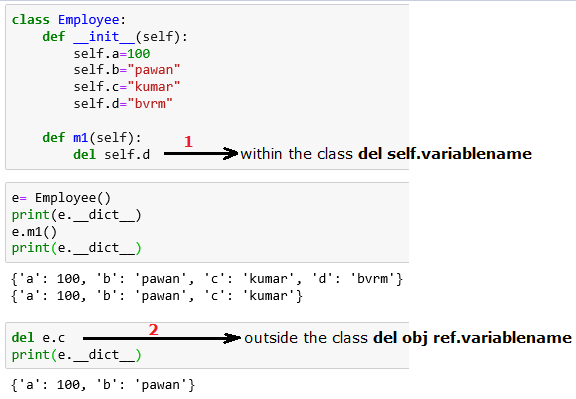
For every object a separate copy of instance variable will be created

**Declaration of Instance variables**:- 

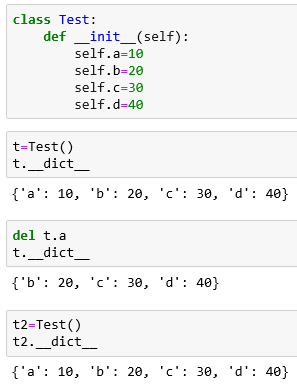
**Accessing of Instance varibles:-**



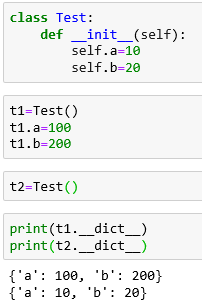
**Delete instance variable:-**



**Note:- The instance variables which are deleted from one object, will not be deleted from other objects.**



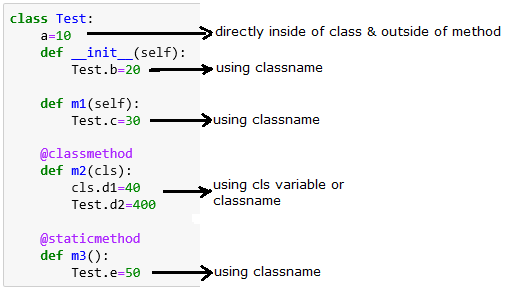
**If we change the values of instance variables of one object then those changes won't be reflected to the remaining objects**



**2.Static variables:-** If the value of a variable is not varied from object to object

For total class only one copy of static variable will be created and shared by all objects of that class

declaring:



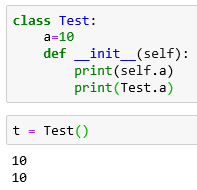
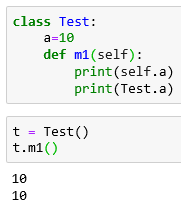






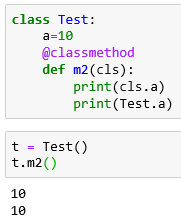
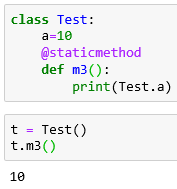
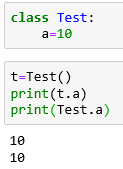
Accessing:

Inside constructor: self or classname Inside instance method: Self or classsname

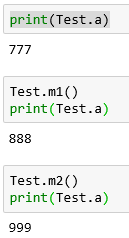


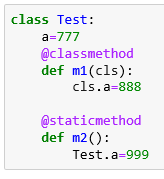
Inside class method: inside static method: classname From outside of class:

cls variable or classname object reference or classname



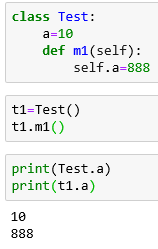
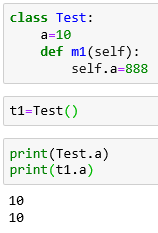
**Where we can modify the value of static variable:**

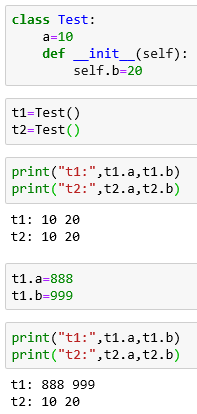
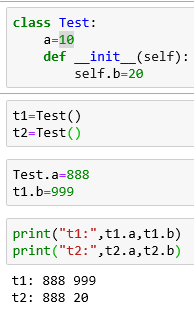
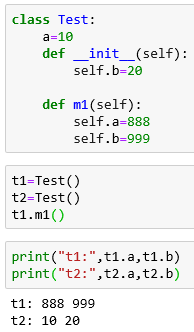
 Anywhere either with in the class or outside of class we can modify by using classname. But inside class method, by using cls variable.

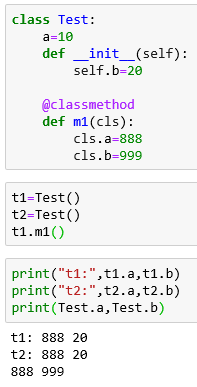


**If we change the value of static variable by using either self or object reference variable:**

Then the value of static variable won't be changed, just a new instance variable with that name will be added to that particular object

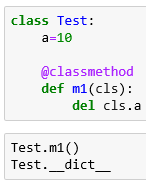


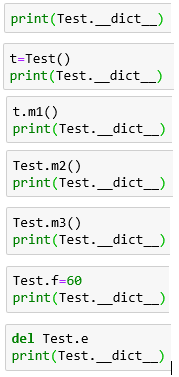


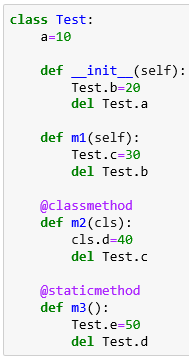


Delete: we can delete static variable from anywhere **del classname.variablename**

But inside the class method **del cls.variablename**







**Note**: By using object reference variable/self we can read static variables, but we cannot modify or delete.

If we are trying to modify, then a new instance variable will be added to that particular object. t1.a = 70

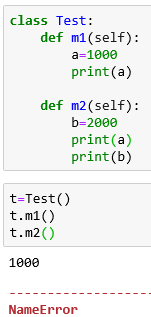
If we are trying to delete then we will get error.

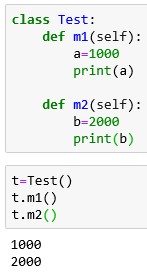
**Local variables:**

Sometimes to meet temporary requirements of programmer,we can declare variables inside a method directly,such type of variables are called local variable or temporary variables.

Local variables will be created at the time of method execution and destroyed once method completes.

Local variables of a method cannot be accessed from outside of method.





**Inside python class 3 types of methods:**

* **Instance methods**
* **Class methods**
* **Static methods**

**Instance Methods:** Inside method implementation if we are using instance variables then such type of

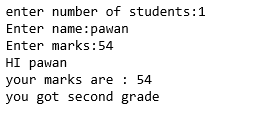
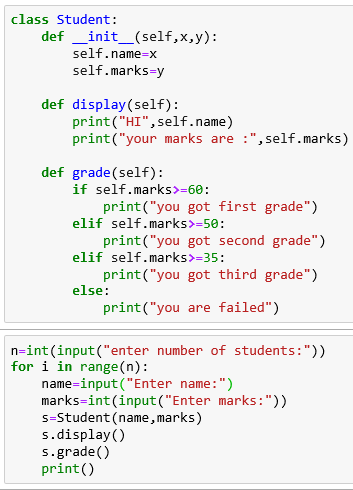
Methods are called instance methods

Inside instance method declaration we have to pass self variable

**Access**:

we can call instance method by using **self** variable –inside the class

**object** **reference** -- outside the class



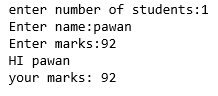
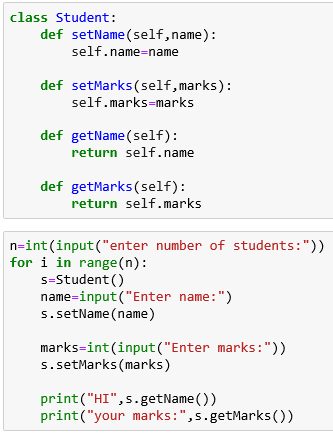
**Setter and Getter Methods:**

We can set and get the values of instance variables by using getter and setter methods

**syntax: syntax:**

def setVariable(self,variable): def getVariable(self):

self.variable=variable return self.variable

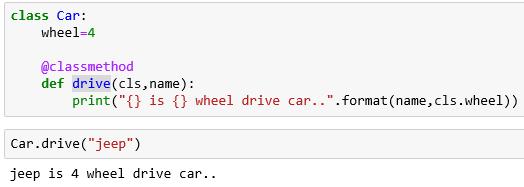


**Class Methods**: inside the method implementation if we are using static variables

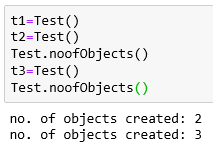
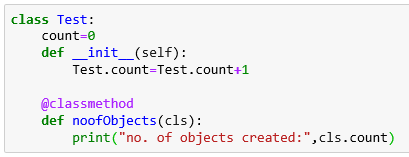
We can declare @classmethod decorator

And also cls variable inside the method

**Access**: class methods can access by using **classname** or **obj ref** variable



**Tracking no. of objects created**

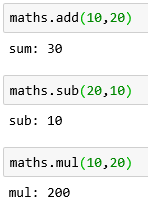
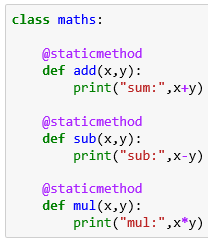


**Static method(utility methods):** we won’t use any **instance** or **class** variables

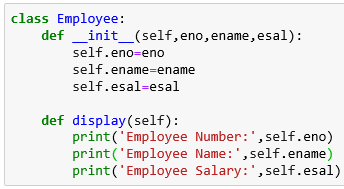
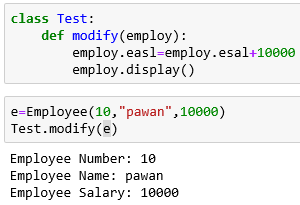
We won’t provide **self** or **cls** variable at the time of declaration

Just provide @staticmethod decorator

**Access: classname or obj ref variable**



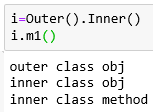
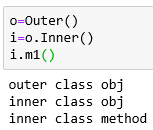
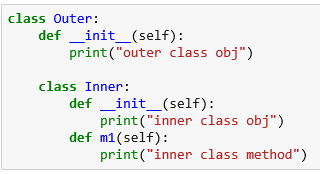
**We can access members of one class inside another class.**



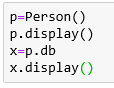
**Inner class:**

Sometimes we can declare a class inside another class, such type of classes are called inner classes.

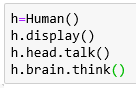
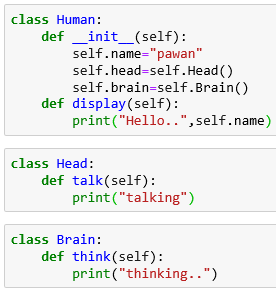
Without existing one type of object if there is no chance of existing another type of object, then we should go for inner classes.



**Demo:2**



**Demo:3**



GC:

**Import** gc

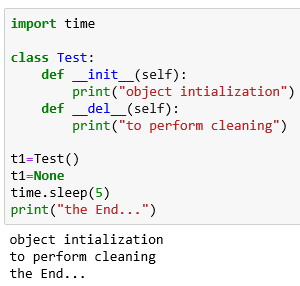
gc.isenabled()

gc.disable()

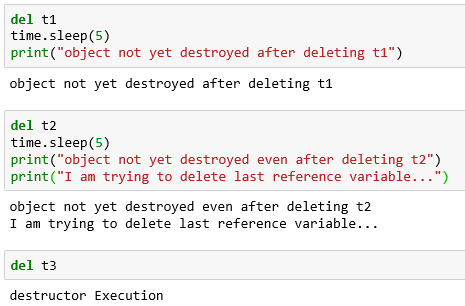
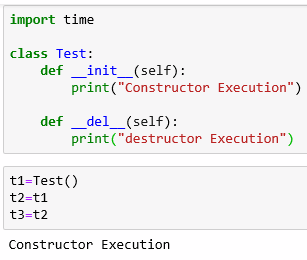
gc.enable()

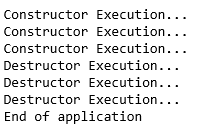
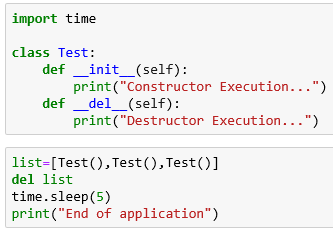
**destructors(to perform clean up activities)**

Destructor is a special method and the name should be \_\_del\_\_ Just before destroying an object Garbage Collector always calls destructor to perform clean up activities (Resource deallocation activities like close database connection etc). Once destructor execution completed then Garbage Collector automatically destroys that object.



**Note**: If the object does not contain any reference variable then only it is eligible fo GC. ie if the reference count is zero then only object eligible for GC

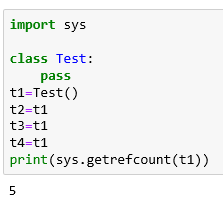




**How to find the number of references of an object:**

**sys** module contains **getrefcount()** function for this purpose**.**

**sys.getrefcount(objectreference)**



**Note: For every object, Python internally maintains one default reference variable self**

S