	Page No.	Teacher Sign / Remarks

Types of Data

1) Structured data → Tables → SQL
2) Semistructured data → Excel, Json → office tool
3) unstructured data → Emails, chats, msgs, video, Audio, movies, logs, photos, etc. → NO SQL (Mongo DB)

PYTHON (purely object oriented programming Language)

Programming

* Compiled Base language
* Interpreted language.

wap to print "helloworld"

C

```
# include<studio.h>
# include<conio.h>
  void main()
{
  clrscr();
  Printf("hello world");
  getch();
}
```

Java

```
class demo
{
  psvm(String args[])
  {
  s.o.p("helloworld");
  }
}
```

Python

```
Print("helloworld");
```

Syntax :- Set of Rules & Regulation to write a code in particular Language.

Compiler :-

* It is used for checking syntax.
* It will check line by line whether the code is syntatically correct or not if complete code is syntatically correct then only it will generate o/p otherwise it will give error.

Syllabus

1) core python
2) Adv
   - oops
   - Intorator, generator, decorator
   - file operations
   - Exceptional Handling
3) Basics of Data science
   - Numphy
   - pandas
   - matplotlib
4) Django

Introduction to python

→ python is a high level programming language with application in numerous array including web programming, scripting, scientific computing & artificial intelligence

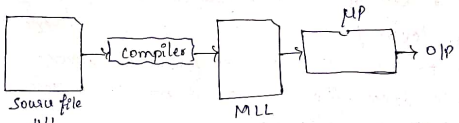
→ Python is a general purpose interpreted lang. object oriented & high level programming lang.

→ It was created by "Guido van Rossum" in the year of 1985 to 1990 (1989) exact release in the year of 1991

features of python

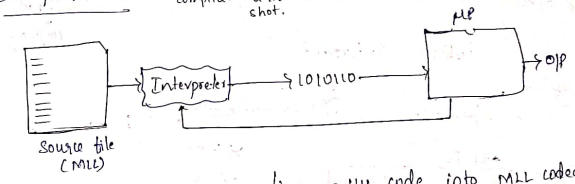
* python is interpreted language
* python is object oriented programming lang.
* python is interative language.
* python is a buisiness language

Compilation



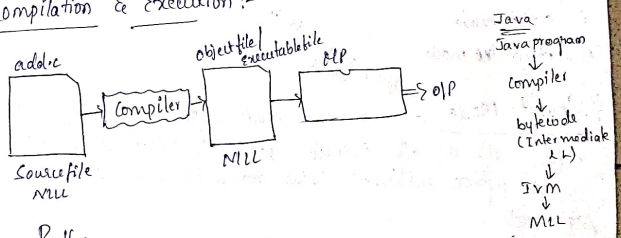
it is the process of converting the HLL code into the MLL code. In this process a compiler s/w is used. In the process of compilation a HLL code is converted into the MLL code in one shot.

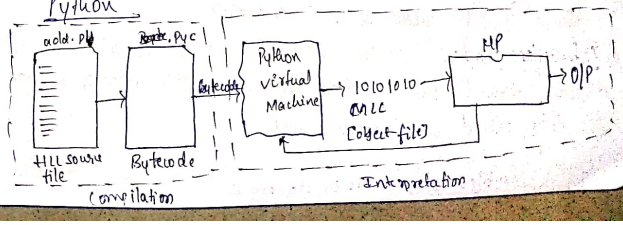
Interpretation:-



* it is the process of converting a HLL code into MLL coded
* in this process an interpreter s/w is used. in the process of interpretation a high level code is taken from source code line by line & is converted into the machine level code.
* this process continues until the last line is finished its execution

Compilation & Execution:-





* Python is a hybrid programming language. python is both compiled as well as interpreted programming language.

but if we look from Execution point of view it is considered to be as interpreted programming language. But initially a source file is converted into the byte code which is platform independent

* Python is portable programming language.

Notepad → command Prompt

Python Interpreter

Script mode — Interaction mode

add.Py

```
Print ("add two numbers")
a=2
b=3
c=a+b
Print(c)
```

```
>>> print ("Add 2 no")
>>> a=2
>>> b=3
>>> c=a+b
>>> print (c)
```

inorder to Execute the python on the interpreter there are 2 ways
1) script mode
2) interactive mode

1) Script Mode :- a script mode in python is nothing but a file with an .Py Extension which is given as an i/p to the python interpreter. This process is only called as script mode.

(In file, program is in static)

Script is also called as file & module

2) Interactive mode :- in the interactive mode we can give the i/p directly through the python interpreter this mode of Execution is only called as interactive mode.

In file, program is in dynamic

Command line & arguments

Command prompt

```
C:/users/user1> python addpy ABC FOR TECH
Add 2 no's
5
```

Sys.argv → [add.py | ABC | FOR | TECH]
sys.argv[0] sys.argv[1] sys.argv[2] sys.argv[3]

Command line arguments are the data inputs that is given to the Python interpreter before the execution via the command form. These data i/ps are collected by the python program in the form of list. called as sys.argv

Objects in python

Object
- has / state → handle by data type
- does / Behaviour → handle by functions

in java
does part by handle by methods

wap to print 500 times

Java

```
class Demo
{
psvm ( )
{
for (int i=0; i<=500 ;i++)
{
S.O.P ("hello world");
}
}
```

Python

```
print ("Hello world\n" * 500)
```

o/p
hello world
hello world
:

```
print ("Hello world" * 500)
```

o/p HelloworldHello worldHello world
- - - -

Public static
↓ ↓
accessible for operating sys to access the program

Swapping of two no

```
class demo
psvm (
```

```
int a = 10;
int b = 20;
int temp;
temp = a;
a = b;
b = temp;
s.o.p(a);
s.o.p(b);
}
```

O/P
10 20
20 10

```
a, b = 10, 20
print(a)
Print(b)
Swap b, a = a, b
Print(a)
Print(b)
```

O/P
10 20
20 10

Objects in python

Datatypes in python

Real world data	Data types in :	
1) integer type data	→ int	Basic data types
2) Real no. type data	→ float	
3) character type data	→ string	
4) yes/no type data	→ boolean	
5) still pictures		Built in classes
6) Audio		
7) video		

Advanced datatypes
* list
* tuple
* set
* Dictionary

Why Python is dynamic change language
Python decide the data type based on the o/p is given.

Eg: age = 20
print(age) // 20
Print(type(age))
// <class 'int'>

1) Number type data:

In order to store no. there are 3 datatype in python.
→ int
→ float
→ complex.

Ex:- age = 20
Print(age), // 20
Print(type(age)) // <class 'int'>

compile & execute $5

commenting a line use # to multiple line """ """

weight = 60.5
print(weight) // 60.5
Print(type(weight)) // <class 'weight'>

x = 2+4j
Print(x) // 2+4j
Print(type(x)) // <class 'complex'>

2) Yes/No type data.

In Order to store yes/no type data in python bool data type is there

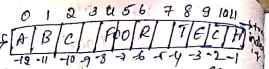
Ex:-
```
is_married = False
Print(is_married)          // False
Print(type(is_married))    // <class 'bool'>

is_placed = True
print(is_placed)           // True
Print(type(is_placed))     // <class 'bool'>
```

Note:- python is dynamically type programming language by, the type of the variable will be automatically decided based upon the value which will be assigned to a variable.

3) Character type data

The character type data will be stored using string datatype
Strings in python is an array of characters.
In Order to access the individual character present within a string we can make use of index operator [].
String can be accessed by both positive indexing as well as -ve indexing.

Ex:- S = 'ABC FOR TECH'




```
Print (S)
Print (id(S)) // 53936400
Print (type(S)) // <class 'str'>
Print (S[4]) // F
Print (S[0]) // A
Print (S[-3]) // E
Print (S[3]) // ' '
Print (S[12]) // string index out of range
Print (S[-13]) // string index out of range
```

Note:- The string has to be enclosed within a double ~~coded~~ quotes or single ~~coded~~ quotes

Escaping a character inside a string

```
S = 'This is a famous quote: "I think therefore I'am"'
Print (S) -> Error
S = 'this is a famous quote: "I think therefore I\'am"'
Print (S) // this is a famous quote: "I think therefore I'am"
S = 'this is a famous quote: "I think therefore I'am" '
Print (S) -> Error
```

Note:- In order to escape a character use backward slash(\) before single quote & double quote

~~Mist~~ Advanced Data type

① List

List is a ordered collection of data which can store both homogeneous as well as hetrogeneous data.

List in python are mutable which means that addition, modify, insertion of the value for the existing element & deletion of the element is possible.

```
Ex:- lst1 = [10, 20.30, 40, 50] -> Homogeneous data
     print (lst1) // [10, 20, 30, 40, 50]
     Print (type (lst1)) // <class 'list'>
     lst2 = [10, 20, 'abc', 22.3, "xyz", True] -> Heterogeneous data
     Print (lst2) // [10, 20, 'abc', 22.3, "xyz", True]
     Print (lst2[1]) // 20
     Print (lst2[-3]) // 22.3
```

Addition

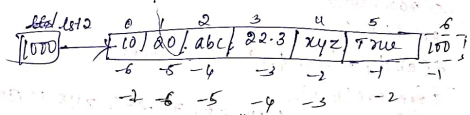
```
lst2.append (100) // [10, 20, 'abc', 22.3, "xyz", True, 100]
Print (lst2)
print (lst2.append (1000)) // None
```

modification

```
lst2 [2] = "ABC"
Print (lst2) // [10, 20, "ABC", 22.3, "xyz", True, 100]
```

Deletion

```
lst2.remove ('xyz')
print (lst2) // [10, 20, "ABC", 22.3, True, 100]
```



"List is mutable"

② Tuple -> immutable"

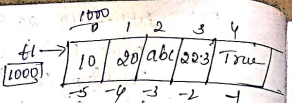
Tuples are similar to a list where it is a older collection of data which can store both homogeneous as well as hetrogeneous type of data.

```
Ex:- t1 = (10, 20, 30, 40, 50)
     Print (t1) // (10, 20, 30, 40, 50)
     Print (type (t1)) // <class 'tuple'>
     t1 = (10, 20, 'abc', 22.3, True)
     Print (t1) // (10, 20, 'abc', 22.3, True)
     Print (type (t1)) // <class 'tuple'>
     Print (t1[3]) // 22.3
```

addition

```
Print t1.append (100) -> Error
Print (t1)
```

Modification t1[3] = 'ABC' → Error
Print (t1)
Deletion t1.remove (True) → Error
Print (t1)



3) Set

Set is an ordered collection of data which can store both homogeneous as well as heterogeneous data. Set in python are mutable.

Ex:- S = {10, 20, 'abc', True, 70, 70, 60, 90, 90, xyz}
Print(S) // {True, 10, 20, 70, 'abc', xyz, 90, 60}
Print (type(S)) // <class 'set'>

addition
Error ← S.append (100) // (True, 10, 20, 70, abc, 100, xyz, 60, 90, }
S.add (100)
print (S)

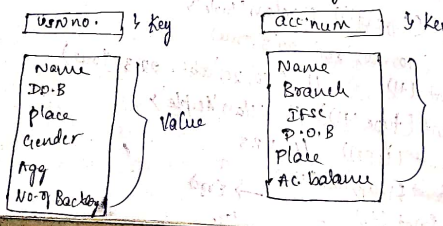
modification
S['abc'] = 'abc for tech' → error

deletion
S.remove (True)
print (S) → {10, 20, 70, abc, xyz, 60, 90, 100}
S.pop()
print (S) → {20, 70, xyz, abc, 100, 90, 60}

Note:- Elements which are stored inside a set should be unique. i.e., the duplicates are not permitted.

4) Dictionary

It is ordered collection of data where the data will be stored in the form of {Key: value} pairs.

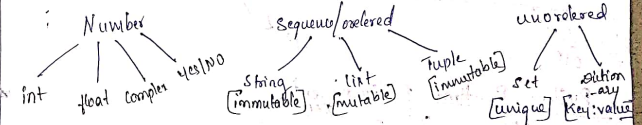


Ex:- d = {1: ('vikki', 18, 'abc10dc'), 2: ('sandesh', 20, xyz10)}
Print (d) // {1: ('vikki', 18, 'abc10dc'), 2: ('sandesh', 20, 'xyz10')}
d1 = {1: 'a', 2: 'b', 3: 'c'}
Print (d1) // {1: 'a', 2: 'b', 3: 'c'}
Print (type (d1)) // <class dict>

addition
d[4] = 'e'
Print (d1) // {1: 'a', 2: 'b', 3: 'c', 4: 'e'}

modification
d1[2] = 'g'
Print (d1) // {1: 'a', 2: 'g', 3: 'c', 4: 'e'}

deletion
del (d1[2])
Print (d1) // {1: 'a', 3: 'c', 4: 'e'}



Note:- * declaration is not possible in python. However we should declare a variable and in the same line we should assign the value to the variable.

a → Error
a = 10

Java/C
int a; declare
a = 10; assign or definite

* In python single line assignment is possible
a, b, c = 10, 20, 30

Java/C
int a = 10;
int b = 20;
int c = 30;

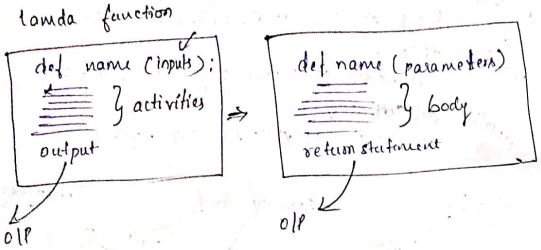
Pvm → it gives control to the python file [Python virtual machine] by using Py we can call pvm

Type casting :- converting one data type to another data type
→ implicit → automatically
→ Explicit → user have to mention

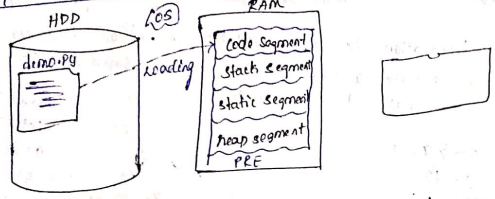
Functions

there are 4 types of functions.

1) user defined function
2) Built in function [BIF's]
3) Recursive function
4) Lamda function



Python Runtime Environment



PRE is a region which present inside the RAM & ~~allocated~~ by operating System inorder to execute python file

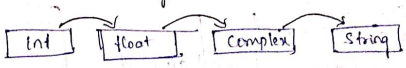
Type Casting in python

Converting one data type into another data type is referred as "Type casting".

Implicit Type casting

if smaller data type has been converted into a larger datatype by the python interpreter automatically. then it is referred as "implicit typecasting"

Eg:-
```
a=10
print(a)        → 10
print(type(a)) #class 'int'
b= 20.5
Print(b)        → 20.5
print(type(b)) # <class 'float'>
c = a+b
Print(c)        # 30.5
Print(type(c)) # <class 'float'>
```



Explicit type casting

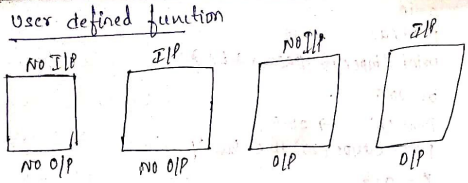
Ex1 :-
```
a =10
Print(a)
Print(type(a))
b= '20'
print(b)
print(type(b))
c= a+b
print(c)  ——→ Error
print(type(c))
```

Ex2:-
```
a=10
Print(a)          # 10
print(type(a)) <class 'int'>
b='20'
print(b)          # (20)
print(type(b)) <class 'str'>
c= a+int(b)       10+20
print(c)          → 30
print(type(c)) #<class 'int'>
```

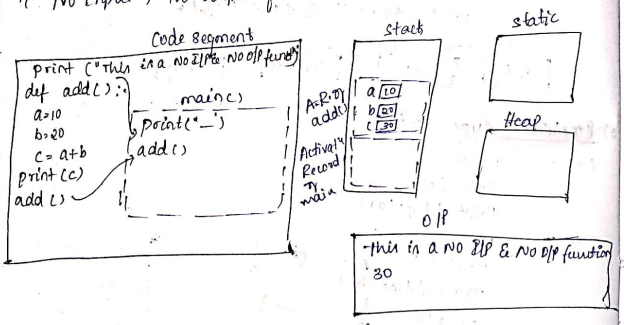
int float complex string

if larger datatype has to be converted into a smaller data type then python interpreter will not automatically convert rather it programmer has to convert using built in functions this is referred as "Explicit type casting"

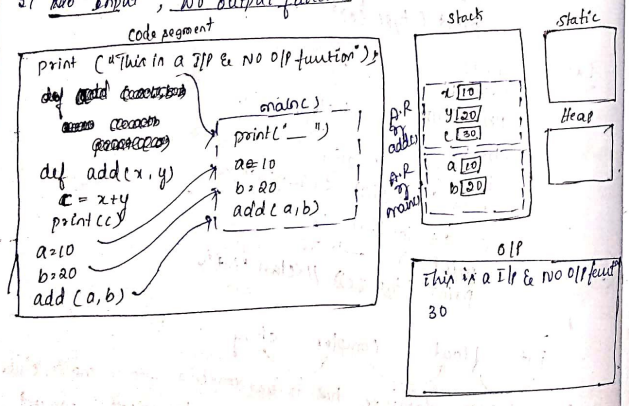
1) User defined function



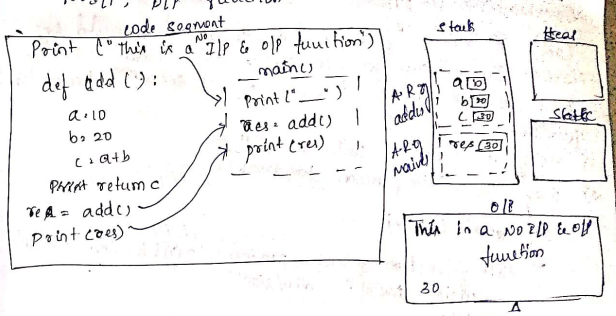
1) No Input, No output function.



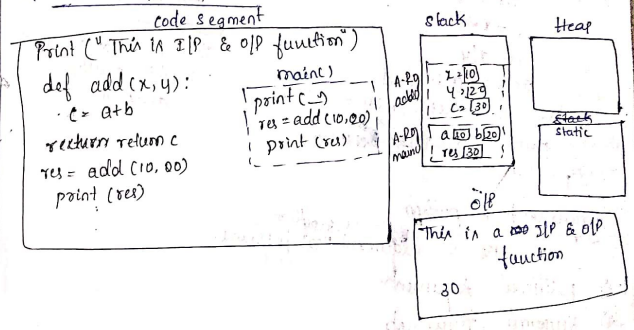
2) Input, No output function:



3) No I/P, O/P function



4) I/P, O/P function



Note: Returning multiple outputs in python.

It can provide multiple inputs as well as it can return multiple o/ps in python.

```
def calc(x,y):
    a = x+y
    b = x-y
    c = x*y
    d = x/y
    return a,b,c,d
res1, res2, res3, res4 = calc(20,10)
print(res1, res2, res3, res4)
```

O/P

30, 10, 200, 2.0

Documentation in python.

Each & every function has to be documented by placing the sentence with in Tripple single quots.
There are 2 ways to access the documented function.

1) using a variable called as __doc__
2) using the builtin function help()

```
def calc(x, y):
    "this function which performs addition, subtraction, multiplication, division"
    a = x+y
    b = x-y
    c = x*y
    d = x/y
    return a, b, c, d
res1, res2, res3, res4 = calc(20, 10)
print(res1, res2, res3, res4)
```

O/P
30, 10, 200, 2.0

this function is to add, subtract, multiply & division.

Arguments in python

there are 5 arguments in python as follows.

1) positional Arguments
2) Keyword Arguments
3) default Arguments
4) variable length Arguments & Arbitrary Argument
5) variable length keyword Arguments (&) Arbitrary Keyword Argument

1) Positional Arguments

Eg:-
```
def power_of(num, power):
    return num ** power
print(power_of(5,2)) → 25
print(power_of(2,5)) → 32
```

whenever the data has been passed using positional arguments then the order & position of the data has to be compulsorily remembered. if not the output is not obtained as per the expectation.

the disadvantage of positional arguments has been overcome by using keyword arguments.

2) Keyword Arguments

Eg:-
```
def power_of(num, power):
    return num ** power
print(power_of(num=2, power=5)) → 25
print(power_of(power=5, num=2)) → 25
```

In keyword arguments the data will be passed by making use of keyword as an index.

3) Default argument / optional arguments

→ using function we can reuse the code.

In python, the parameter can be given a default value i.e., "default argument & optional argument"

→ In order to collect the argument Parameters are need

Ex: 1
```
def power_of(num, power=1):
    return num ** power
print(power_of(5))
```
O/P 5

Ex: 2
```
def power_of(power=1, num)
    return num ** power
print(power_of(5)) → Error
print(power_of(num=5)) → Error
```
} non-default arguments follows default

Note: if the parameter is declared a default value then it must be placed after non default argument if not, it would result in the error as shown above.

4) Variable length arguments (or) Arbitary Arguments

Ex:- def pizza_toppings (toppings)
    return print (toppings)
pizza_toppings ('onion', 'cheese', 'corn', 'chicken', 'paneer').

O/P [Error] → pizza_topping() takes 1 positional arguments but 5 were given.

to overcome the above error variable length arguments are used as shown below.

Ex:- def pizza_toppings (*toppings):
    print (toppings)
pizza_toppings ('onion', 'cheese', 'corn', 'chicken', 'paneer')

O/P :- ('onion', 'cheese', 'corn', 'chicken', 'paneer')

whenever length of arguments been passed is not fixed i.e., it keeps on changing then we should go for variable length arguments and while creating parameter it should be prefixed with '*'

the output will be stored in the form of tuple which is immutable

5) Variable length keyword arguments or Arbitary Keyword Argument

Ex:- def details_of (**details):
    print (details)
details_of ('name': 'deepak', 'age': 18, 'gender': 'Male', 'country': 'India', 'state': 'Haryana', 'location': 'palwal', 'Aggr': 70.2, 'No of Backlog': 2, 'weight': 60.5)

O/P: {'name': 'deepak', 'age': 18, 'gender': 'Male', 'country': 'India', 'state': 'Haryana', 'location': 'palwal', 'Aggr': 70.2, 'noOfBacklog': 2, 'weight': 60.5}

If the arguments are passed is of unknown datas then we should go for variable length keyword arguments where the data will be stored in the form of dictionary (key:value pairs) the output variable length keyword arguments will be stored in the form of dictionary. (key-value pairs)

Recursive function : (can call function within a function)

A function calling itself is referred as Recursive function

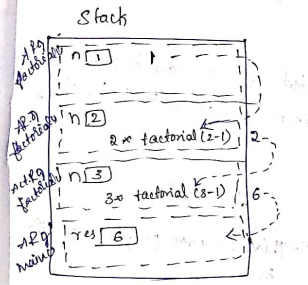
Ex:- Prog to print the factorial of a number

1! = 1
2! = 2*1
3! = 6 = 3*2!
4! = 24 = 4*3!
n! = n*(n-1)!

code segment

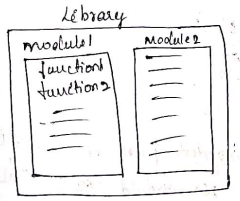
def factorial_of (n):
    if (n==1):
        return 1
    else:
        return n*factorial_of(n-1)
res = factorial_of (3)
print (res)

main
res = factorial_of(3)
print (res)



O/P 6

Modules in Python:



Module.calc.py

```
def add(x,y):
    "This function performs addition"
    return x+y
def sub(x,y):
    "This function performs subtraction"
    return x-y
def mul(x,y):
    "this function performs multiplication"
    return x*y
def div(x,y):
    "This function performs division"
    return x/y
```

Script

demo.py

```
import calc
print (dir (calc))
print (calc.add (__doc__))
print (calc.add (10,20))
```

o/p

```
add, sub, mul, div
this function
addition
30
```

* A module in python is a collection of functions & arguments.
* A module as to be imported into another python file inorder to use the functions present within that module.
* the function will be accessed by making use of module name.

Alias of a Module

A module will be imported within another program and for the case of programming it will be alias as shown below.

Ex:

demo.py

```
import calc as c
print (dir (c))
print (c.add.__doc__)
Print (c.add (10,20))
```

o/p

this function performs addition
30

demo1.py

```
from calc import *
print (add (10,20))
print (sub (10,20))
```

o/p

30
-10

whenever the functions are imported from module then while passing the arguments we should call only the function not the module.

Importing a specific function from a module

Ex:-

```
from calc import add,div.
print (add (10,20))  → 30
print (div (10,20))  → 0.5
print (sub (10,20))  → Error because it is not imported (sub is not defined)
```

Executing the module by verifying __name__ variable

module calc.py

```
def add ():
    print ("addition")
def sub():
    print ("subtraction")
def mul():
    print ("multiplication").
def div():
    print ("division")
if __name__ == __main__
    sub()
```

o/p

subtraction

script.

demo.py

```
import calc
```

o/p

Every python file will be having a builtin variable called as __name__ and it will be having the value __main__ whenever the functions are called within the module then it should be called within the main function as shown above

Taking input from the keyword

Ex:-

```
Print ("enter the first number");
a = input()
Print (type (a)) # <class 'str'>
```

```
Print ("Enter the second number")
b, input()
Print (type (b)) # <class 'str'>
c = a+b
print (c)
print (type(c))
```

O/P :-
10
20
1020

In, python input will be taken from the keyboard using input() by default the i/p will be stored in the form String to overcome this input function has to be typecast

Ex2:-
```
print ("enter the first no.")
a = int (input())
print ("enter the second no.")
b = int (input())
c = a+b      c = 10+20 # 30
print (c)  -> 30
print (type (a)) # int
print (type (b)) # int
```

Turtle program

A turtle is nothing a integer present with in a function called Turtle function will be there with in a module turtle within a module turtle

Ex:- Wap to create square

```
import turtle
sau = turtle.Turtle()
sau.fd(100)
sau.lt(90)
sau.fd(100)
sau.lt(90)
sau.fd(100)
sau.lt(90)
sau.fd(100)
sau.lt(90)
```

O/P

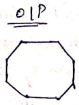
```
import turtle
sau = turtle.Turtle()
def square(length):
    for i in range(4):
        sau.fd(length)
        sau.lt(90)
square(100)
```

O/P

Wap to create a polygon where the user should provide the length & no. of sides

```
import turtle
sau = turtle.Turtle()
def polygon(length, n):
    angle = 360/n
    for i in range(n):
        sau.fd(length)
        sau.lt(angle)
polygon(70, 8)
```

O/P



Lambda functions (Anonymous function)

General syntax:- lambda inputs: expression

using def Keyword

```
def power_of (num, power):
    return num ** power
print (5, 2)
```

O/P:- 25

using lambda keyword

```
res = (lambda num, power: num ** power)(5,2)
print (res)
```

O/P 25

* lambda functions are anonymous functions that is name less functions.
* lambda functions are single line functions
* the output of lambda functions will be returned implicitly
* the inputs to the lambda functions has to be passed in the same line where the function got created.

Note:- 1. lambda functions can be called multiple times by making use of the variable name where the output of the lambda function is stored.

Ex:- using def keyword

```
def power_of (num, power):
    return num ** power
print (power_of (5,2)) -> 25
print (power_of (2,5)) -> 32
print (power_of (1,0)) -> 1
```

using lambda keyword

```
p = lambda num, power: num ** power
print (p(5,2)) -> 25
print (p(2,5)) -> 32
print (p(1,0)) -> 1
```

2 multiple lines are not permitted inside a lambda function.

using def keyword

```
def power_of (num,power):
    print ("This is def keyword"):
    return num** power
print (power_of (5,2))
```

O/P this is def keyword
25

using lambda keyword

```
p=(lambda num,power: print ("this is def keyword")
    num** power) (5,2)
print (p)
```

O/P error

Diff b/w def Keyword & lambda keyword

def Keyword	lambda keyword
* these are the functions created using names	* these are nameless functions.
* To obtain the output return statement has to be used explicitly	* the o/p of lambda function is return implicitly.
* these are multiline functions	* these are single line functions
* these are multiuse functions	* generally these are single use functions.

Returning the output within a function using lambda function

we can create a function within a function

Ex1 :

```
def func(n):
    return lambda x: n*x
doubler=func(7)
print (doubler(2)) → 14
tripler = func(7)
print (tripler(3)) → 21
```

O/P
14
21

Ex2:-

```
def table(n):
    return lambda x: n*x.
n= int (input ("enter the table of your choice"))
table_calc = table(n)
for i in range (1,11):
    print ("n", 'x', i, '=', table_calc(i))
```

O/P Enter the table of your choice

```
8*1 = 8
8*2 = 16
.
.
8*10 = 80
```

Note: lambda functions are used in some built in function like filter function filter()
map function map()
reduce function (reduce())

Filter()

general syntax : filter (lambda function, sequence)
sequence: list, tuple, set

Ex: lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_lst = list (filter (lambda i: i%2 ==0, lst))
print (even_lst)

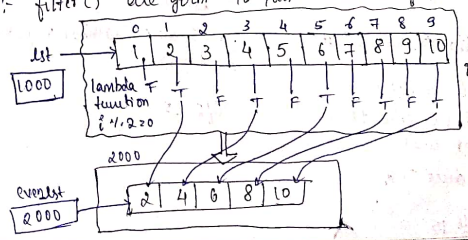
O/P [2, 4, 6, 8, 10]

without using filter()

```
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_lst = []
def even_no (lst):
    for i in lst:
        if i%2==0:
            even_lst.append(i)
even_no(lst)
```

O/P:
[2, 4, 6, 8, 10]

Note:- filter() are good to take lambda functions and sequence as input & o/p will be return in the form of filter object.



Map()

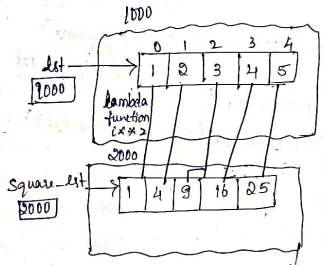
general Syntax:- map(lambda function, sequence)

Ex:- lst = [1, 2, 3, 4, 5]

```
Square_lst = []
def square(lst):
    for i in lst:
        Square_lst.append(i**2)
square(lst)
```

```
lst = [1,2,3,4,5]
Square_lst = list(map(lambda i: i**2, lst))
print(square_lst)
```

o/p [1, 4, 9, 16, 25]



Note: map function going to take lambda() & sequence as i/p & going to return the o/p in the form of map object

Reduce function

General Syntax:- reduce(lambda function, sequence)

Ex:- 1, 2, 3, 4, 5

1+2 = 3
3+3 = 6
6+4 = 10
10+5 = 15

i 12345
sum 01815

(a) lst = [1,2,3,4,5]

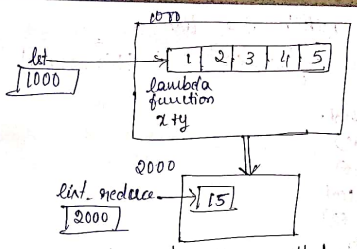
```
def list_reduce(lst):
    sum=0
    for i in lst:
        sum=sum+i
    return sum
res= list_reduce(lst)
print(res)
```

o/p 15

(b)

```
from functools import reduce
lst= [1,2,3,4,5]
res= reduce(lambda x,y: x+y, lst)
print(res)
```

o/p 15



Note:- reduce function will be present within a module called as functools. It is going to make use of lambda functions to retrieve the output as a single element.

Some Built in functions present within math module.

```
import math
print(math.pi)          # 3.141592...
print(math.factorial(5)) # 120
print(math.fabs(-5))    # 5.0
print(math.copysign(5,-10)) # -5.0
print(math.ceil(5.4))   # 6
print(math.floor(6.8))  # 6
```

LIST

Creation of list

Ex:- lst = []
print(lst) # []

Empty list

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

Homogeneous

```
lst = [10, 20, 30, 40]
print(lst) # [10, 20, 30, 40]
```

Heterogeneous

```
lst = [10, 'abc', 10, ABC, 20]
print(lst) # [10, 'abc', 10, ABC, 20]
```

Set, dictionary, tuple

```
lst = [10, 20, [30,40], (50,60), {2,3,5,7}, {1:'a', 2:'b'}]
print(lst) # [10, 20, [30,40], (50,60), {2,3,5,7}, {1:'a', 2:'b'}]
```

type

```
print(type(lst)) # <class 'list'>
```

address

```
print(id(lst)) # 16882704
```

Accessing the element within the list

lst = [10, 20, [30, 40], (50, 60), {2, 3, 5, 7}, {1:'a', 2:'b'}]

print (lst[0]) → 10

Accessing a list within a list:

print (lst[2]) → [30, 40]

print (lst[2][1]) [40]

Accessing a tuple within a list

print (lst[3]) (50, 60)

print (lst[3][0]) 50

Accessing a set within a list

print (lst[4]) {2, 3, 5, 7} print (lst[4].pop()) 2

print (lst[4][1]) //error

Accessing dictionary within a list

print (lst[5]) {1:'a', 2:'b'}

print (lst[5][2]) b

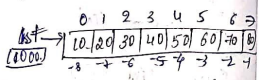
List Slicing :-

List slicing is used to obtain a portion of a list from the original list

Slicing is used to create a sublist using the original list

general syntax:- new_list [start : stop : step]

Ex:- lst = [10, 20, 30, 40, 50, 60, 70, 80]

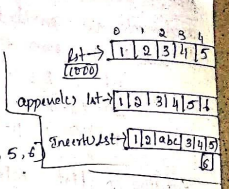



```
new_list = lst[0:6]
new_list = lst[:]
new_list = lst[4:]
new_list = lst[4::2]
new_list = lst[4::-1]
new_list = lst[4:7:-1]
new_list = lst[::-1]
new_list = lst[-3:-7]
new_list = lst[-3:-7:-1]
new_list = lst[-1:0:-1]
print (lst)
print (new_list)
```

Adding the elements to a list

Ex:-

```
lst = [1, 2, 3, 4, 5]
print (lst)          [1, 2, 3, 4, 5]
lst.append (6)
print (lst)          [1, 2, 3, 4, 5, 6]
lst.insert (2, 'abc')
print (lst)          [1, 2, 'abc', 3, 4, 5, 6]
```



Replication of a list :-

(duplicate)

Ex:-

```
lst = ['a'] * 1000
print (lst)   // 'a', 'a' ... 1000 times
lst = [1, 2, 3, 4, 5] * 30
print [lst]   // 1, 2, 3, 4, 5 ... 30 times
```

Note :- * append() is used to add the elements to a list at the rear end.

* insert() is used to add the elements at the specified position by making use of indexing.

Adding multiple elements to a list or concatination of a list.

Ex:-

```
lst = [1, 2, 3, 4, 5]
lst.append (6,7)
print (lst) // error [append() takes exactly one argument (2 given)]
```

using append() multiples elements cannot be added to a list.

In order to add multiple elements to a list we make use of '+' operator (or) extend().

Ex:- 1) lst = [1, 2, 3, 4, 5]

```
temp = [6,7]
lst = lst + temp (i.e. lst = temp + lst) // [6,7,1,2,3,4,5,6]
print (lst) // [1, 2, 3, 4, 5, 6, 7]
```

2) lst = [1, 2, 3, 4, 5]

```
lst.extend ([6,7])
print (lst) // [1, 2, 3, 4, 5, 6, 7]
```

Note:- extend() is going to take only one argument. if multiple arguments are provided as i/p to the extend() then it is going to result an error shown below.

```
lst= [1,2,3,4,5]
lst.extend(6,7)
print(lst)  #--> Error
```

Adding a list within a list

Ex:-
```
lst= [1, 2, 3, 4, 5]
lst.append([6,7])
print(lst)  # [1, 2, 3, 4, 5, [6, 7]]
```

Note:- using append() list, tuple, set, and dictionary can be added within a list.

```
lst = [1, 2, 3, 5, 5]
lst.extend((6,7))
print(lst)  # [1, 2, 3, 4, 5, 6, 7]
```

Modification of the element present within a list.

Ex:-
```
lst = [1, 2, 3, 4, 5]
lst[2] = 333
print(lst)  # [1, 2, 333, 4, 5]
```

* Using indexing operator the existing element with in a list can be modified

Removing the elements from a list.

Ex:-
```
lst= [1, 2, 3, 4, 5, 'abc', 6]
lst.remove(5)
print(lst)  # [1, 2, 3, 4, 'abc', 6]
lst.pop(1)
print(lst)  # [1, 3, 4, 'abc', 6]
lst.pop()
print(lst)  [1, 3, 4, 'abc']
```

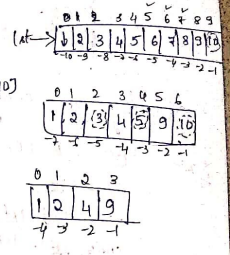
* remove() is used to remove the elements with in a list it takes the elements as inputs.

* pop() is used to remove the elements from the list it takes the index as the i/p

Removing a portion of list using slicing.

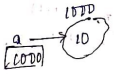
a portion of a list in access using slicing inorder to delete a portion of list 'del' keyword in used

Ex:-
```
lst= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
del lst[5:8]
print(lst)  # [1, 2, 3, 4, 5, 9, 10]
del lst[-1:-6:-2]
print(lst)  # [1, 2, 4, 9]
del lst[-1:-2:-1]
print[lst]  # [1, 2, 4]
del lst[-1:-2]
print(lst)  # [1, 2, 4, 9]
```



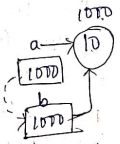
Note:- python is a purely object oriented programming language
∴ Everything in a python is treated as object

Ex:-
```
a=10
print(a)  --> value of object
print(type(a)) -> class of object
print(id(a)) -> address of object
```



Reference type assignment in python

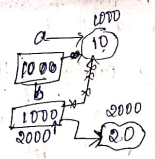
Ex:-
```
a=10
b= a
print(b)  -> 10
print(type(b)) -> class 'int'
print(id(a)) -> 1000
print(id(b)) -> 1000
```



Note:- In reference type assignment if a variable is assign to one more variable then the address will be copied so that both the references will start to point the same object

Ex:-2

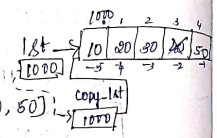
```
a=10
print(a)
b=a
print(b)
print(id(b))
b=20
print(b)
print(id(b))
```



Creating a copy of list

Ex:-

```
lst = [10, 20, 30, 40, 50]
print(lst)          # [10, 20, 30, 40, 50]
print(id(lst))      # 1000
copy_lst = lst
print(copy_lst)     # [10, 20, 30, 40, 50]
print(id(copy_lst)) # 1000
lst.remove(40)
print(lst)          # [10, 20, 30, 50]
print(copy_lst)     # [10, 20, 30, 50]
```

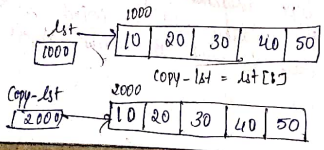


In order to create a copy of original list there are two ways

* Creating a copy of original list using slicing.

Ex:-

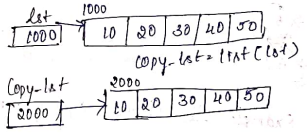
```
lst = [10, 20, 30, 40, 50]
copy_lst = lst[:]
print(lst)      → [10, 20, 30, 40, 50]
print(id(lst))  → 1000
print(copy_lst) → [10, 20, 30, 40, 50]
print(id(copy_lst)) → 2000
```



* Creating a copy of original list using built in function called as 'list()'

Ex:-

```
lst = [10, 20, 30, 40, 50]
copy_lst = list(lst)
print(lst)      # [10, 20, 30, 40, 50]
print(copy_lst) # [10, 20, 30, 40, 50]
print(id(lst))      → 1000
print(id(copy_lst)) → 2000
```

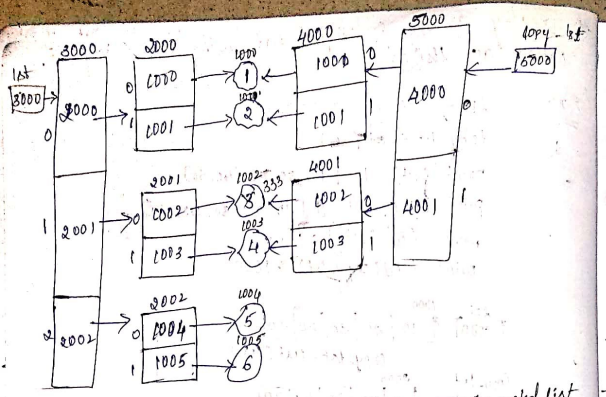


Creating Copy of Nested list

Two
Shallow Copy

Ex:-

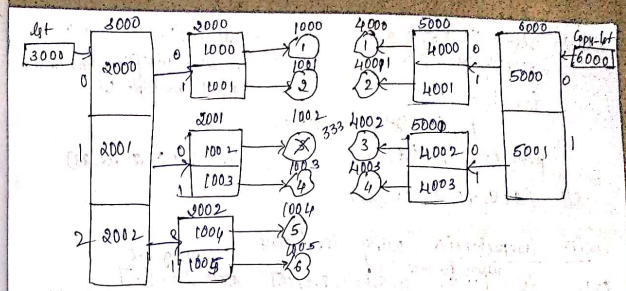
```
lst = [[1,2],[3,4]]
print(lst)      → [[1,2],[3,4]]
copy_lst = lst[:]
print(copy_lst) → [[1,2],[3,4]]
lst.append([5,6])
print(lst)      → [[1,2],[3,4],[5,6]]
print(copy_lst) → [[1,2],[3,4]]
lst[1][0] = 333
print(lst)      → [[1,2],[333,4],[5,6]]
print(copy_lst) → [[1,2],[333,4]]
print(id(lst))      → 3000
print(id(copy_lst)) → 5000
```



* In the above example the shallow copy of nested list is created
* In the shallow copy of a list objects are not copied instead references are copied
* so any modification done wrt. original list will get reflected in the copy
* inorder to avoid the disadvantages of shallow copy we should make use of deepcopy of a list

Deep copy

```
import copy
lst = [[1,2],[3,4]]
print(lst)  → [[1,2],[3,4]]
copy_lst = copy.deepcopy(lst)
print(copy_lst)
lst.append([5,6])
print(lst)  → [[1,2],[3,4],[5,6]]
print(copy_lst) → [[1,2],[3,4]]
lst[1][0] = 333
print(lst) → [[1,2],[333,4],[5,6]]
print(copy_lst) → [[1,2],[3,4]]
print(id(lst)) → 3000
print(id(copy_lst)) → 6000
```



* 0. deepcopy of a list can be created using deepcopy() which present with in a copy module.
* In deepcopy of a list references will not be copied instead objects are copied so any modification done w.r.to original list will not reflected on the copy

Difference b/w Shallow copy and deep copy

Shallow copy	Deep copy
* shallow copy of a list is created using list slicing or list()	* deep copy of a list is created using deepcopy().
* References are copied. not the objects	* objects are copied. not the references.
* Any modification done wrt. to original list. is getting reflected on the copy.	* Any modification done w.r.to. original list will not get reflected on the copy.

List Comprehensions (comprehension in need to reduce the code)

List Comprehension is a technique used to create a brand new list using a original list.

general syntax:- new_list = [what i want where i want it from based on what condition]

Ex:- lst = [1, 2, 3, 4, 5]

```
square_lst = []
for i in lst
    square_lst.append(i**2)
print(square_lst)
```

```
lst = [1,2,3,4,5]  (optional)
square_lst = [i*2 for i in lst]
print(square_lst)
```

O/P [1, 4, 9, 16, 25]

Ex2:- lst = [1, 3, 5, 7, 9, 11]
multiple_lst = []
for i in lst
multiple_lst.append(i*3)
print (multiple_lst)

o/p [3, 9, 15, 21, 27, 33]

lst = [1, 3, 5, 7, 9, 11]
multiple_lst = [i*3 for i in lst]
print (multiple_lst)

o/p [3, 9, 15, 21, 27, 33]

List Comprehension using conditions

using for loop

Ex1:- lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_lst = []
for i in lst
if i%2 == 0
even_lst.append(i)
print (even_lst)

o/p. [2, 4, 6, 8, 10]

using L.C

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

o/p. [2, 4, 6, 8, 10]

Ex-2 using for loop

lst = [1, 0, 20, 'abi', 'xyz', -1, 3, 5, 7]
integer_lst = []
for i in lst :
if (type(i) == int):
int square_lst.append(i**2)
print (intSquare lst)

o/p [1, 0, 400, 1, 9, 25, 49]

using L.C.

lst = [1, 0, 20, 'abi', 'xyz', -1, 3, 5, 7]
integer_lst = [i**2 for i in lst if (type(i) == int)]
print (integerlst)

general syntax of L.C. new_list = [expression for <elements> in sequence condition]

List Comprehension using multiple if condition

using for loops

Ex:- lst = [1, 2, 4, 6, 10, 12, 17, 21, 23]
new_lst = []
for i in lst:
if i%6
if i%2 == 0:
new_list.append(i)
print (new_list

o/p. [10, 12]

using L.C

lst = [1, 2, 4, 6, 10, 12, 17, 21, 23]
newlist = [i for i in lst if i%6 if i%2 == 0]
print (new_list)

o/p [10, 12]

Ex:- lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
new_lst = []
for i in lst :
if i%2 == 0 :
new_lst.append('*')
else;
new_lst.append(i)
print (new_lst)

o/p [1, '*', 3, '*', 5, '*', 7, '*', 9, '*']

using L.C

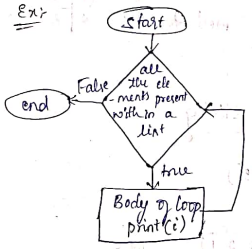
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
new_lst = [i for i in lst if i%2==0]
new_lst = ['*' if i%2 == 0 else i for i in lst]
print (new_lst)

* while writing L.C. if there is if else condition then it has to be placed after the expression.

Accessing the Elements present with in a list using for loop

Using for loop the data present within a list will be fetched out

Ex:-



lst = [1, 2, 3, 4, 5]
for i in lst
print(i)

o/p 1
2
3
4
5

Accessing the value and index from a list using.

1) Enumerate()
2) range()

1) Enumerate

```
lst = [1,2,3,4,5]
for i,j in enumerate(lst):
    print('Index:', i, 'value:', j)
```

O/P

Index	Value
0	1
1	2
2	3
3	4
4	5

2) Range()

```
lst = [1, 2, 3, 4, 5]
for i in range(len(lst)):
    print('index:', i, 'value:', lst[i])
```

O/P

Index	Value
0	1
1	2
2	3
3	4
4	5

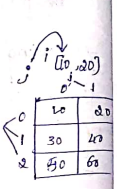
Accessing the data present within nested list.

Ex:-

```
lst = [[10,20], [30,40], [50,60]]
for i in lst:
    for j in i:
        print(j)
```

O/P

10
20
30
40
50
60

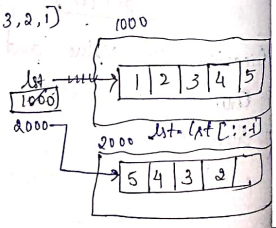


Note:- The above process will be used for the flattening of matrix.

Reversing of list using list slicing

Ex:-

```
lst = [1, 2, 3, 4, 5]
print(lst) → [1,2,3,4,5]
print(id(lst)) → 1000
lst = lst[::-1] → [5,4,3,2,1]
print(lst)
print(id(lst)) → 2000
```



Comparision of a list

Ex:-

```
lst1 = [10,20,30,40,50]
lst2 = [10,20,30,40,50]
print(lst1 == lst2) → true
lst1 = [10,20,30,40,50]
lst2 = [10,20,30,40,50,60]
print(lst1 < lst2) → True
lst1 = [10,20,30,40,50,60]
lst2 = [10,20,30,40,50,60]
Print(lst1 < lst2) → False
lst1 = [0,20,30]
lst2 = [30,40,50,60]
print(lst1 < lst2) → True
lst1 = [99,66]
lst2 = [99,66,78]
print(lst1 > lst2) → True
```

[it compares the elements with in a list]

Arranging the elements present withina list

1) Ascending order

```
lst = [10,34,56,78,90,36,89]
print(lst) → [10,34,56,78,90,36,89]
print(id(lst)) → 1000
lst = sorted(lst)
print(lst) → [10,34,36,56,78,89,90]
print(id(lst)) → 2000
```

2) Descending Order

```
lst = [10,34,56,78,90,36,89]
print(lst) [10,34,56,78,90,36,89]
print(id(lst)) → 2000
lst = sorted(lst, reverse=True)
print(lst) → [90,89,78,56,36,34,10]
print(id(lst)) → 2000
```

Membership check of a element with in a list

Ex:- lst = [10, 34, 56, 78, 89, 90, 36]
print (34 in lst) → true
print (87 not in lst) → true

Slicing Assignment of a list

Ex- lst = [10, 20, 30, 40, 50]
print (lst [1:4]) → [20, 30, 40]
lst [1:4] = [99]*4
print (lst) → [10, 99, 99, 99, 99, 50]
lst [1:4] = [99]*6
print [lst] [10, 99, 99, 99, 99, 99, 99, 99, 50]
lst [1:4] = [99]*3
print (lst) → [10, 99, 99, 99, 50]
lst [1:4] = [99]*2
print (lst) → [10, 99, 99, 99, 50]
lst [1:4] = [99]*1
print (lst) → [10, 99, 50]

TUPLES :-

Tuples are ordered collection of data which can store both homogeneous & heterogeneous data.
Duplicates are permitted within a tuple.

Ex:- tup = ()
print (tup) → ()
print (type (tup)) →
tup = (10, 20, 30, 40)
print (tup) → (10, 20, 30, 40)
tup = (10, 'abc', 4+3j, 'xyz', 30)
print (tup) → (10, 'abc', 4+3j, 'xyz', 30)
tup = (10, 20, [30, 40], (50, 60), {7, 6, 8}, {1: 'a', 2: 'b'})
print (tup) → (10, 20, [30, 40], (50, 60), {7, 6, 8}, {1: 'a', 2: 'b'})
tup = 10
print (tup) → 10
print (type (tup)) → int

Creation of Single item tuple

Ex:- tup = 10,
print (tup) → (10)
print (type (tup)) → class 'tup'
tup = (10)
print (tup) → 10
print (type (tup)) → 'int'
tup = (10, 20)
print (tup) → (10, 20)
print (type (tup)) → 'tup'
tup = (10, 20, 30)
print (tup) → (10, 20, 30)
print (type (tup)) → 'tup'

* A single item tuple will be having exactly one element
* in order to create single item tuple (,) should be used after the element, if not it will be treated as integer.
* if the elements are stored with in the common brackets it is called as packing of tuple.
* in order to create a tuple common brackets are optional.

unpacking of a tuple

Ex: tup = (10, 20, 30, 40, 50)
print (tup) → (10, 20, 30, 40, 50)
Unpacking a, b, c, d, e = tup
print (a) → 10
print (b) → 20
print (c) → 30
print (d) → 40
print (e) → 50

disposable variable '_'

_, _, c, d, _ = tup
print (c) → 30
print (d) → 40
print (a) → error

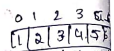
Note:- tuples are immutable in nature i.e., addition, modification, deletion in not possible.
using the keyword 'del' we can delete complete tuple.

Ex:- tup=(10,20,30)
print (tup) → (10,20,30)
tup.append(40)
print(tup) → Error
tup.insert(0,99)
print(tup) → Error
tup.pop(-2)
print(tup) → Error
tup.remove(30)
print(tup) → Error
del tup
print(tup) → Error

Tuple Slicing

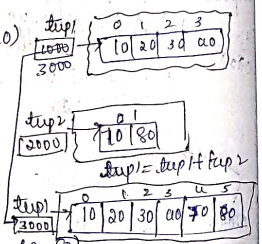
It is used to slice the portion of elements from the original tuple.

tup = (1,2,3,4,5,6)
print(tup) → (1,2,3,4,5,6)
print(tup[2:5]) → (3,4,5)
print(tup[-2:-5]) → ()
print(tup[0:-1:2]) → (1,3,5)
print(tup[::-1]) → (6,5,4,3,2,1)



Concatination of tuple

Ex:- tup1 = (10, 20, 30, 40)
print(tup1) → (10,20,30,40)
print(id(tup1)) → 1000
tup2 = (70,80) → (70,80)
print(tup2) → 2000
print(id(tup2)) →
tup1 = tup1 + tup2
print(tup1) → (10,20,30,40,70,80)
print(id(tup1)) → 3000

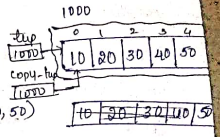


if two tuples are concatinated then the elements will be stored in a brand new object i.e different address will be there.

Shallow copy of tuple

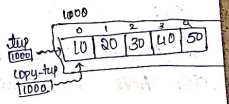
A copy of tuple cannot be created using slicing function (tuple()) if attempted then brand new objects are cannot be created rather addresses are copied. Hence tuple are more memory efficient than list.

tup = (10,20,30,40,50)
copy_tup = tuple(tup)
print(tup) → (10,20,30,40,50)
print(id(tup)) → 1000
print(copy_tup) → (10,20,30,40,50)
print(id(copy_tup)) → 1000



Deep copy of tuple

import copy
tup = (10,20,30,40,50)
copy_tup = copy.deepcopy(tup)
print(tup) → (10,20,30,40,50)
print(id(tup)) → 1000
print(copy_tup) → (10,20,30,40,50)
print(id(copy_tup)) → 1000



If tried to create a deep copy of a tuple then objects will not be created rather address will be copied.

Few Built-in Functions in tuple :-

Ex:- tup = (10, 20, 30, 40, 50)
print(max(tup)) → 50
print(min(tup)) → 10
for i,j in enumerate(tup)
print(i,j) → 0 10
1 20
2 30
3 40
4 50

differences b/w list and tuple

Feature	list	tuple
Mutability	YES	NO
ordered	YES	YES
creation	[ ]	( )
Addition	.. append(), extend()	NA
modification	[] = value, insert()	NA
deletion	remove(), pop()	NA
slicing	lst [start : stop : step]	tup [start : stop : step]
slicing assignment	[:] = [99] * value	NA
iteration	for i in lst:	for i in tup:

Set

* A Set is an ordered collection of data which can store both homogeneous as well as heterogeneous data.
* duplicates are not permitted with in a set. that means Element stored within a set should be unique.
* within a set we cannot store a list, set, dictionary However, we can store tuples.

Ex:-
```
S = { }
print(S) -> { }
print (type(S)) -> class 'dict'
S = set()
print (S) -> { } (or) set()
print (type(S)) -> class 'set'
S = {10, 20, 30, 40} -> Homogeneous
print(S) -> {20, 10, 40, 30}
S = {10, 20, 'abc', 'xyz', 10, 20, 30} -> Heterogeneous
print (S) -> {10, 'abc', 30, 'xyz'}
S = {10, [40, 20]}
print (S) -> error (unhashable type -> list)
S = {10, (40, 20)}
print(S) -> {10, (40, 20)}
S = {10, {10, 20}}
print(S) -> error (unhashable type -> set)
S = {10, {1: 'a', 2: 'b'}}
print (S) -> error (unhashable type -> dict)
```

Set Operation

There are four important set operations.

1) union

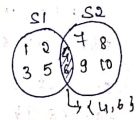
upon performing union b/w two sets then the op will be the combination of both the sets

Ex:-
```
S1 = {1, 2, 3, 4, 5, 6}
S2 = {4, 6, 7, 8, 9, 10}
S3 = S1 | S2 (or)
S3 = S1.union(S2)
print (S3) -> {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

2) Intersection

upon performing intersection b/w two sets then the common elements b/w two sets will be written as output

Ex:-
```
S1 = {1, 2, 3, 4, 5, 6}
S2 = {4, 6, 7, 8, 9, 10}
S3 = S1 & S2 (or)
S3 = S1.intersection S2
print (S3) -> {4, 6}
```



3) Difference

upon performing difference b/w two sets this going to eliminate common elements and the remaining elements present within the individual set will be printed as output.

Ex:-
```
S1 = {1, 2, 3, 4, 5, 6}
S2 = {4, 6, 7, 8, 9, 10}
S3 = S1 - S2 (or)
S3 = S1.difference S2
print (S3) -> {1, 2, 3, 5}
```

s3 = s2 - s1 (or)
s3 = s2.difference(s1)
print(s3) → {7, 8, 9, 10}

4) Symmetric Difference

Upon performing symmetric diff b/w two sets it is going to eliminate the common elements b/w the sets and it is going to return, the remaining elements as output.

Ex:- s1 = {1, 2, 3, 4, 5, 6}
s2 = {4, 6, 7, 8, 9, 10}
s3 = s1 ^ s2 (or)
s3 = s1.symmetric_difference(s2)
print(s3) → {1, 2, 3, 5, 7, 8, 9, 10}

5) Checking if a set is a subset

Ex:- s1 = {1, 2, 3, 4, 5, 6}
s2 = {4, 5, 6}
print(s2.issubset(s1))
print(s2 <= s1) (or) → True

s1 = {1, 2, 3, 4, 5, 6}
s2 = {2, 5, 6}
print(s1.issuperset(s2))
print(s1 >= s2) (or) → True

s1 = {1, 2, 3, 4, 5, 6}
s2 = {8, 9}
print(s1.isdisjoint(s2)) → True

Features	operators	functions
union	|	s1.union(s2)
Intersection	&	s1.intersection(s2)
difference	-	s1.difference(s2)
Symmetric_difference	^	s1.symmetric_difference(s2)
subset	<=	s1.issubset(s2)
superset	>=	s1.issuperset(s2)
disjoint		s1.disjoint(s2)

Adding and Removing the elements from a set

Ex:- s = {1, 2, 3, 4, 5, 6}
print(s) → {1, 2, 3, 4, 5, 6}
s.add(7)
print(s) → {1, 2, 4, 5, 6, 7, 3}
s.pop()
print(s) → {2, 1, 4, 5, 6, 7}
s.remove(4)
print(s) → {2, 5, 6, 7, 1}
s.discard(5)
print(s) → {2, 6, 7, 1}
s.remove(90) → Error
s.discard(90) → {2, 6, 7, 1} // No error.
print(s)

* pop() will pop the random elements. it will not take any arguments because no indexing in there for set.
* Remove() and discard() is used to remove a single element from a set. it takes the values as input.
* if we provide the unknown element to both the function then remove() gives an error (Key Error).

Frozen Set in Python

Sets are mutable in nature however by making use of Frozenset we can create a immutable set.

Ex:- s = frozenset({1, 2, 3, 4, 5})
print(s) → {1, 2, 3, 4, 5}
s.add(50)
print(s) → Error
s.remove(4)
print(s) → Error

Set comprehension

s1 = {10, 2, 4, 3, 1, 7, 9}
s2 = set()
for i in s1:

using comprehension
s1 = {10, 2, 4, 3, 1, 7, 9}
s2 = {i*i for i in s1 if i%2==0}
print(s2)

if i%2==0
s2.add(i*i)

O/P: {100, 4, 16}

print(s2)

O/p: {100, 4, 16}

All() and ANY() in Python

using for loop

Ex: s={1,2,3,4,5,6,7,-8,9,10}

new_list = []

for i in s

new_list.append(i>0)

print(all(new_list)) → False

print(any(new_list)) → True

print(new_list) → [True, True, True, True, True, False, True, True, True, True]

using List comprehension

S={1,2,3,4,5,6,7,-8,9,10}

print(all([i>0 for i in S])) → False

print(any([i>0 for i in s])) → True

* All() → if any of the condition is False then o/p is false
* Any() → if any of the condition is true then o/p is True

List Slicing

list = [1, 2, 3, 4, 5, 6, 7, 8, 9] [start : stop : step]

	0	1	2	3	4	5	6	7	8
list →	1	2	3	4	5	6	7	8	9
	-9	-8	-7	-6	-5	-4	-3	-2	-1

print(list) → [1, 2, 3, 4, 5, 6, 7, 8, 9]

print(list[:]) → [1, 2, 3, 4, 5, 6, 7, 8, 9]

print(list[0:]) → [1, 2, 3, 4, 5, 6, 7, 8, 9]

print(list[2:8]) → [3, 4, 5, 6, 7, 8]

print(list[-10:-6:]) → []

print(list[-9:6]) → [1, 2, 3, 4, 5, 6]

print(list[::-1]) → [9, 8, 7, 6, 5, 4, 3, 2, 1]

print(list[8:-6:-1]) → [9, 8, 7, 6, 5]

print(list[2:]) → [3, 4, 5, 6, 7, 8, 9]

print(list[-9:8:-1]) → []

print(list[::-3]) → [9, 6, 3]

print(list[-7::-2]) → [3, 1]

print(list[::-6]) → [9, 3]

print(list[-6:5:-2]) → []

print(list[-7::2]) → [3, 6, 8]

print(list[-8::-1]) → [2, 1]

print(list[-6::3]) → [4, 7]

Note:- *All() is used to return the output as true if and only if all the conditions are true. if not the o/p will be false

* Any() is used to written the o/p is true if the single condition is true. among all the elements. if not false.

Internal Implementation of a list

In order to store the data list internally make use of variable length array data structure.

Ex: list = []

list.append(10)

list.extend([20, 40, 50])

list.append(30)

list []

memory

No. of elements : 0

No. of list: 0

No memory allocated.

list.append(10)

memory

0	10
1	
2	
3	

No. of elements:-

No. of size :-

list.extend([20, 30, 40])

memory:-

No. of elements : 1

No. of size : 4

10	
20	
30	
40	

5

list.append(50)

memory :-

No. of elements: 1, 2, 3, 4, 5

No of size : 4 8

0	10
1	20
2	30
3	40
4	50
5	
6	
7	
8	
9	
10	
11	

} accessible

} not accessible over allocation of memory

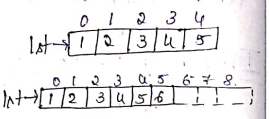
Note:- With in a list overallocation of memory will happen based upon the resizing pattern of this is in order to execute the in future the elements will be added because list is mutable in nature.

Performance analysis of a list

Insertion → raise end
→ random position
→ front end

Case i: insertion at raise end (appends)

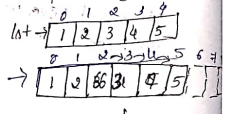
lst = [1, 2, 3, 4, 5]
lst.append (6)
print (lst)



No. of operation: 1 insert operation
= 1

Case ii:- insertion at the ~~raise end~~ (random position (insert())

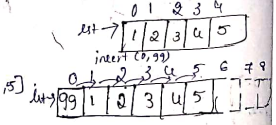
lst = [1, 2, 3, 4, 5]
lst.insert (2, 66)
print (lst) → [1, 2, 66, 3, 4, 5]



No. of operation:- 3 shift operation + 1 insert operation
= 4

Case iii:- insertion at the front end (insert())

lst = [1, 2, 3, 4, 5]
lst.insert (0, 99)
print (lst) → [99, 1, 2, 3, 4, 5]



(or)

lst = [1, 2, 3, 4, 5]
temp = [99]
lst = temp + lst
print (lst)

lst = [1, 2, 3, 4, 5]
temp = [99]
lst = temp + lst
print (lst)

No. of operation: 5 shift operation + 1 insert operation
= 6

Note:- Insertion at raise end < Insertion at random position < Insertion at front end

* list is memory inefficient because of over allocation of memory will happen.

Execution time

Ex:-
```
import timeit
print (timeit.timeit (stmt = "lst.append (6)", setup = "lst = [1,2,3,4,5]", number = 10,000)) → 0.0008
print (timeit.timeit (stmt = "lst.insert (2,66)", setup = "lst = [1,2,3,4,5]", number = 10,000)) → 0.026
print (timeit.timeit (stmt = "lst.insert (0,99)", setup = "lst = [1,2,3,4,5]", number = 10,000)) → 0.027
```

When to go for a list?

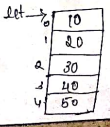
* Whenever the data has to be added at the raise end.
* whenever the length of the data is unknown.
* to store homogeneous and heterogeneous data.
* Whenever the data has to be stored in the ordered manner (~~0 based~~ indexing).
* to store the duplicate data.

Internal implementation of a tuple.

In order to store the data tuple internally makes use of "fixed length array".

No over allocation of memory will happen because tuples are immutable in nature.

Ex:-
```
tup = (10, 20, 30, 40, 50)
print (tup)
```



Creation of a tuple.

* over Allocation of memory is not performed with respect to tuple
* i.e., tuples are immutable in nature
* from the creation perspective tuple takes less time when compared to list.

ex:-
```
import timeit
print(timeit.timeit(stmt="lst=[1,2,3,4,5]", number=10000))
print(timeit.timeit(stmt="tup=[1,2,3,4,5]", number=10000))
```

o/p
0.0007
0.0003

tuple < list

Note:- * A copy of list can be created using list slicing or list function.
* A copy of tuple cannot be created so tuples are memory efficient when compare to list

ex:-
```
lst = [1,2,3,4,5]
copy_lst = lst[:]
print(id(lst)) → 1000
print(id(copy_lst)) → 2000

tup = (1,2,3,4,5)
copy_tup = tup[:]
print(id(tup)) → 1000
print(id(copy_tup)) → 1000
```

when to use tuple ?
* if the data stored as to be immutable.
* whenever the length of the data is known.
* to store duplicates, to store homogeneous and heterogeneous data.

difference b/w list and tuple

list	tuple
* internally it makes use of variable length array.	* internally it makes use of fixed length array.
* it can store homogeneous and heterogeneous data	* it can store homogeneous and heterogeneous data.
* duplicates are permitted.	* duplicates are permitted.
* list are mutable in nature.	* tuples are immutable in nature.
* copies of list can be created. (objects are created)	* copies of tuple cannot be created (reference are copied)
* it is memory inefficient	* it is memory efficient. C:. Over Allocation of memory is not performed)
* performance is slow compared to tuple	* performance is fast compared to list
* list are used if the length of the data is unknown	* it is used if the length of the tuple is known.
* creation of list is slow when compared to tuple.	* creation of tuple is fast when compared to list.
* over allocation is performed	* over allocation of memory is not performed.

Set (Internal implementation of a set)

inorder to store the data set internally makes use of Hashing

S = {16, 8, 21, 75, 165, 8, 32, 3}
print(S)

Hash function: sum(data)/size
Hashkey

16 → 1+6 = 7 % 12 → 7
8 → 8 = 8 % 12 → 8
21 → 2+1 = 3 % 12 → 3
75 → 7+5 = 12 % 12 → 0 (collision occurs, adjusting)
165 → 1+6+5 = 12 % 12 → 0 (1) max index
8 → 8 = 8 % 12 → 8
32 → 3+2 = 5 % 12 → 5
3 → 3 = 3 % 12 → 3

Hash table	
0	75
1	165
2	3
3	21
4	
5	32
6	
7	16
8	8
9	
10	
11	

if multiple index present, it check the data are different then only data are go & sit in Hash table
Empty space orderly on Hash table
otherwise if data are duplicates it won't print the data.

Note:- * Hashing will be having two component
1) Hash function
2) Hash table

* Hash function is going to take the data which has to be stored within hashtable. and it is going to make use of some formula (~~data~~ sum(data)/size) and it generates Hash key.
* using hash key the data will be stored within hash table.
* properties of perfect hash key
  1) it should generate unique Hash key
  2) it should avoid collision.
  3) if collision occurs then using concept called as open addressing it going to resolve the issue. i.e, it will compare the data.

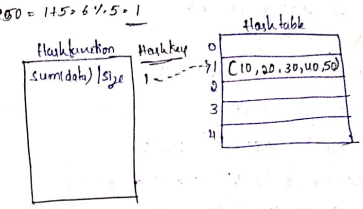
Keypoints to be remembered.

* The order in which data is stored in the Hash table is never decided by the programmer instead it is decided by the Hash function and hence data within a set is unordered.
* if duplicate value is provided to hash function it would generate a hash key & upon going to location on the hash table realize that similar data is already present and hence set does not allow duplicates. ∴ if collision occurs it is going to compare associated data and if the data is similar it is going to eliminate that hash key, if the data is diff then memory will be allocated based upon open addressing.
* one of the expectation of the hash function is that the data provided for hashing must be immutable in nature. This is because if the data changes then the corresponding the hash key which is generated also changes. Hence the set doesn't allow mutable objects such as list, set & dictionary to be stored. but tuples are allowed within a set

lst = [10, 20, 30, 40, 50]
S = {(10, 20, 30, 40, 50)}
10+20+30+40+50 = 150 = 14×10+6 ∴ 6 = 1
Immutable objects

10+20+30+40+60 = 210 = 21×10+3 ∴ 3 = 3 (2) if we append the element one more hash key get changed

tuple
S = {(10, 20, 30, 40, 50)}
150 = 14×10+6 ∴ 6 = 1



* (pointed one box)

Dictionary in Python

Dictionary in python will make use of hashing store the data in the form of key value pair

* the complex values will be stored with the simple keys.
* In order to make the process of hashing symbol simple instead of value keys will be provided for the hash function

Characteristic of a key

* Key should be unique.
* Keys should be simple.
* Duplicate keys are not permitted
* Keys should be immutable objects

Characteristic of a value:

* There is no such constraints for a value we can store complex mutable, immutable objects.

Creation of Empty dictionary

Ex:- d = { }
print(d) → { }
print(type(d)) → <class 'dict'>
d = dict( )
print(d) → { }
print(type(d)) → <class 'dict'>

Creation of dictionary

Ex:- d = {1:'a', 2:'b'}
print(d) → {1:'a', 2:'b'}
lst = [(1,'a'),(2,'b')]
print(dict(lst)) → {1:'a', 2:'b'}

tup = ([8,'c'],[4,'d'])
print(dict(tup)) → {8:'c', 4:'d'}

print(dict({1:'a', 2:'b'})) → {1:'a', 2:'b'}

Adding the elements to the dictionary

Ex:- d = {1:'a', 2:'b'}
print(d) → {1:'a', 2:'b'}
d.update([[3,'a'],[4,'b']])
print(d) → {1:'a', 2:'b', 3:'a', 4:'b'}
d.update({7:'f'})
print(d) → {1:'a', 2:'b', 3:'a', 4:'b', 7:'f'}
d.update(one=700, two=800)
print(d) → {1:'a', 2:'b', 3:'a', 4:'b', 7:'f', one=700, two=800}

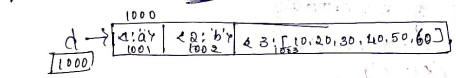
d[86] = 'cv'
print(d) → {1:'a', 2:'b', 3:'a', 4:'b', 7:'f', one=700, two=800, 86='cv'}

Note:- Values present with in a dictionary can be modified by making use of index operator and providing keys.

Ex:- d = {1:a, 2:'b'}
d[1] = 'abc'
print(d) → {1:'abc', 2:'b'}

Accessing the value present within a dictionary

d = {1:'a', 2:'b', 3:[10,20,30,40,50]}
print(d) → {1:'a', 2:'b', 3:[10,20,30,40,50]}
print(id(d)) → 1000
print(d[3]) → [10,20,30,40,50]
print(id(d[3])) → 1003
lst = d[3]
print(lst) → [10,20,30,40,50]
lst.append(60)
print(lst) → [10,20,30,40,50,60]
print(d) → {1:'a', 2:'b', 3:[10,20,30,40,50,60]}
print(id(d)) → 1000
print(id(d[3])) → 1003



Accessing the data using get()

Ex:- d = {1:'a', 2:'b'}
print(d) → {1:'a', 2:'b'}
print(d[1]) → a
print(d.get(1)) → a
print(d[5]) → error → KeyError
print(d.get(5)) → None

Error
↓
Runtime error (exception)
compile time error (error)

In C language exception handling is not there. Hence it cause 'abnormal termination', it leads to crash in os, hardware etc.

* get() is used to fetch the value using key
* If a unknown key is provided as input then o/p will be None

Removing the elements present within a dictionary

Ex:- d = {1:'a', 2:'b', 3:'c', 4:'d'}
print(d) → {1:'a', 2:'b', 3:'c', 4:'d'}
print(d.pop(2)) → b
print(d.pop()) → error
print(d) → {1:'a', 3:'c', 4:'d'}
d.popitem()
print(d) → {1:'a', 3:'c'}
d.clear()
print(d) → {}
del d
print(d) → error

* pop() in dictionary will take exactly one argument & it is used to remove the value using keys.
* popitem() is used to remove the last key value pairs present with in dictionary.
* clear() is used to remove all the key-value pairs present within dictionary.
* del key word is used to remove the complete dict.

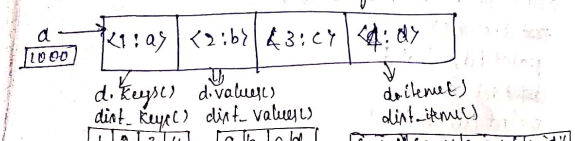
Merging of two dictionary

Eg:-
```
d1 = {1:'a', 2:'b'}
d2 = {3:'c'}
d1.update(d2)  → {1:'a', 2:'b', 3:'c'}
print(d1)
d3 = {**d1, **d2}
print(d3) → {1:'a', 2:'b', 3:'c'}
d3 = d1.update(d2)
print(d3) → None
```

Different way of accessing over the dictionary using dictionary use.

```
d = {1:'a', 2:'b', 3:'c', 4:'d'}
print(d)
print(d.keys())  dict_keys = ([1,2,3,4,5])
print(list(d.keys())) → [1,2,3,4,5]
print(d.values()) → dict_values([a', 'b', 'c', 'd'])
print(list(d.values())) → ['a','b','c','d']
print(d.items()) → dict_items [(1,'a'),(2,'b'),(3,'c'),(4,'d')]
print(list(d.items())) → [(1,'a'),(2,'b'),(3,'c'),(4,'d')]
```

keys() → access the keys
values() → access the values
items() → access in the form of (K:V)



Different ways of iterating over a dictionary

```
d = {1:'a', 2:'b', 3:'c', 4:'d'}
for i in d:
    print(i)
for i in d.keys():
    print(i) → 1 2 3 4
for i in d.values():
    print(i) → a b c d
for i in d.items():
    print(i) → (1,'a')
               (2,'b')
               (3,'c')
               (4,'d')
```

Membership check with in a dictionary

Eg:
```
d = {1:'a', 2:'b', 3:'c', 4:'d'}
print(1 in d) → True
print(4 in d.keys()) → True
print('b' in d.values()) → False
print((3,'c') in d.items()) → True
```

Dictionary Comprehension

Eg:- using for loops
```
lst = [1,2,3,4,5]
d = {}
for i in lst
    d[i] = i**2
print(d)
```

using for loop
```
lst = {1,2,3,4,5}
d = {}
for i in lst:
    if i in lst:
        d[i] = i**2
print(d)
```

using DC
```
lst = [1,2,3,4,5]
d = {i: i**2 for i in lst}
print(d)
```
O/P {1:1, 2:4, 3:9, 4:16, 5:25}

using DC
```
lst = {1,2,3,4,5}
d = {i: i**2 for i in lst if i%2==0}
print(d)
```
O/P {2:8, 4:16}

using for loop

Eg: lst = {1, 2, 3, 4, 5}
d = {}
for i in lst:
    if i%2==0
        d[i**2] = i**3
    else:
        d[i**3] = i**2
print(d)

using DC

lst = {1, 2, 3, 4, 5}
d = {i**2 if i%2==0 else i**3 : i**3 if i%2==0 else i**2 for i in lst}
print(d)

O/P {1:1, 4:8, 27:9, 16:64, 125:25}

Ex-4 using for loop

lst = {1, 2, 3, 4, 5}
d = {}
for i in lst:
    if i%2==0:
        d[i**2] = 'cube'
    else
        d[i**3] = 'square'

using DC

lst = {1, 2, 3, 4, 5}
d = {i**2 if i%2==0 else i**3 : ('cube' if i%2==0 else square) for i in lst}
print(d)

O/P {1:'square', 4:'cube', 27:'square', 16:'cube', 125:'square'}

Zip() in python

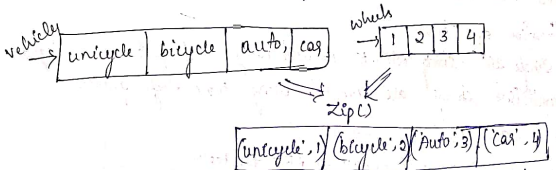
Ex:- Vehicle = ['unicycle', 'bicycle', 'auto', 'car']
wheels = [1, 2, 3, 4]

for i in zip(vehicle, wheels):
    print(i)

for i, j in zip(vehicle, wheels):
    print('key:', i, 'values:', j)

O/P ('unicycle', 1)
('bicycle', 2)
('Auto', 3)
('car', 4)

Key: unicycle values: 1
Key: bicycle value: 2
Key: Auto value: 3
Key: car value: 4



Note:- * zip() is used to merge two sequence and obtain the o/p in the form of key-value pair

Ex:- Zip() on varying lists.

Ex:- Vehicle = ['unicycle', 'bike', 'Auto', 'car']
wheels = [1, 2, 3, 4]
energy_source = ['pedal', 'petrol', 'gasoline', 'diesel']
for i in zip(vehicle, wheels, energy_source)
    print(i)

for i, j, k in zip(vehicle, wheels, energy_source)
    print(i, j, k)

O/P

i	j	k	print(i)
unicycle	1	pedal	('unicycle', 1, 'pedal')
bike	2	petrol	('bike', 2, 'petrol')
Auto	3	gasoline	('Auto', 3, 'gasoline')
car	4	diesel	('car', 4, 'diesel')

Data type	Memory Efficiency	Creation	Insertion			Search
			frontend	Randomend	Rearend	
list	Inefficient	slow	Extremely inefficient	Inefficient	efficient	inefficient
tuple	efficient	fast	NA			inefficient
set	Inefficient	slow	unordered			inefficient
Dictionary	Inefficient	slow	unordered			inefficient

Strings in python

Strings in python is an array of characters.

* string is an ordered collection of data with both positive index and -ve index

Creation of String

* Single line string will be created using single quotes or doublequotes
* multiline strings are created using triple single quotes.

Ex:- S = 'ABC' → ABC

S = "ABC FOR TECH" → ABC for tech

```
S = '''ABC for
JAVA
TESTING
PYTHON
'''
```

print(S) → ABC for
JAVA
TESTING
PYTHON

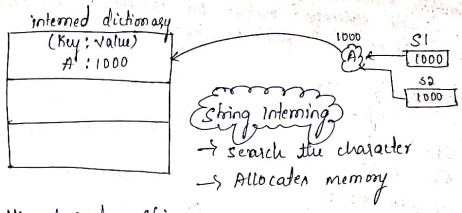
print(type(S)) → <class 'str'>

Internal implementation of String

Single character string

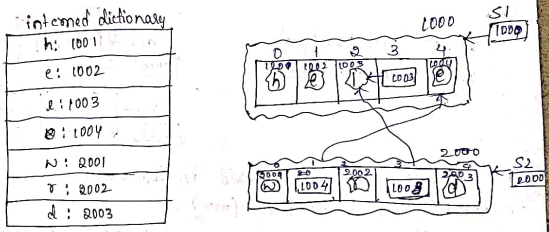
Ex:- S1 = 'A'
S2 = 'A'
print(S1) → A
print(S2) → A

print(id(S1)) → 1000
print(id(S2)) → 1000



Multi character string

S1 = 'hello'
S2 = 'world'
print(S1) → hello
print(S2) → world
print(id(S1)) → 1000
print(id(S2)) → 2000
print(S1[4]) → o
print(S2[1]) → o
print(id(S1[4])) → 1004
print(id(S2[1])) → 1004



Note

* Strings in python are extremely memory efficient because, it makes use of 'string interning'.
* String interning is the process in which the data will be stored within interned dictionary. for individual character. address will be generated and it will be stored in the form

Key-value pairs.

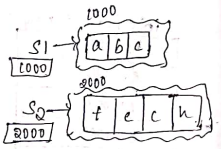
* Multiple occurance of a same character will not be stored within interned dictionary. Rather the address of the previous character will be stored with in string object.

Comparision of Strings

Ex1:-
```
s1 = "abc"
s2 = "tech"
if s1==s2 :
    print ("values are equal")
else :
    print ("values are not equal")
if id(s1) == id(s2) :
    print ("reference are equal")
else :
    print ("reference are not equal")
if s1 is s2 :
    print ("references are equal")
else:
    print ("reference are not equal")
```

O/P

values are not equal
reference are not equal



Note:- * strings can be compared in two ways
→ Comparing the values using (==)
→ comparing the references using id() or is keyword

Ex:-2
```
s1 = "abc"
s2 = "tech"
if s1 is s2:
    print ("value are equal")
else:
    print ("values are not equal")
if id(s1[0]) == id(s2[2]):
    print ("reference are equal")
else:
    print ("reference are not equal")
```

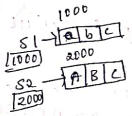
O/P

values are not equal
reference are equal.

Ex3:-
```
s1 = "abc"
s2 = "ABC"
if s1==s2:
    print ("values are equal"):
else:
    print ("values are not equal")
if id(s1[0]) = id(s2[2])
    print ("reference are equal")
else:
    print ("reference are not equal")
```

O/P

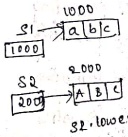
values are not equal
reference are not equal



Ex4:-
```
s1 = "abc"
s2 = "ABC"
if s1==s2.lower() :
    print ("value are equal")
else :
    print ("value are not equal")
if id(s1) == id(s2)
    print ("~~value~~ reference are equal")
else
    print ("value are not equal
            reference")
```

O/P

value are equal
Reference are not equal



here no brand new object are created for the lower function just it took out the character and display in smaller case

s2.lower()
abc

Note: strings are immutable in nature it means that * data stored within interned dictionary cannot be Modified

Ex: s = 'abc'
print (s[1])
s[1] = 'b'
print (s) → error

Few built in functions in Strings

Ex: s = "ABC for tech"
print (s.lower()) → abc for tech
print (s.upper()) → ABC FOR TECH
print (s.capitalize()) → Abc for tech
print (s.title()) → Abc For Tech
print (s.casefold()) → abc for tech
print (s.swapcase()) → abc FOR TECH

Map to replace vowels present within a string

s = "abc for tech"
print (s) → abc for tech
result = str.maketrans ("aeiou", "12345")
print (s.translate (result)) → 1bc f4r t2ch

Ex2:
s = "abc for tech"
print (s) → abc for tech
sp = " for tech"
result = str.maketrans ("aeiou", "12345")
print (s.translate (result)) → 1bc f4r t2ch
(s1.translate (result)) → f4r t2ch

Formating in strings → changing the data however you want
1) Default formating
2) positional formating
3) Keyword formating
4) Binary formating

{} → placeholder

1) default formating
s1 = "abc"
s2 = "for"
s3 = "tech"
print ("{}{}{}".format (s1, s2, s3)) → abcfortech

2) positional formating
s1 = "abc"0
s2 = "for"1
s3 = "tech"2
print ("{2}{0}{1}", format (s1, s2, s3)) → techabcfor

3) Keyword formating
s1 = "abc" arg1
s2 = "for" arg2
s3 = "tech" arg3
print ("{arg1}{arg3}{arg2}".format (arg1=s1, arg2=s2, arg3=s3))
→ abctechfor.

4) Binary formating
→ binary representation
a = 45
b = 10
print ("{a:b} {b:b}".format (a, b)) → 101101 1010

→ pi representation
pi = 3.14678910
print ("{0:0.2f}".format (pi)) → 3.15

→ exponential representation
x = 168.90678787878
print ("{0:.2e}".format (x)) → 1.69e+02

→ percentage representation
y = 0.85
print ("{:.2%}".format (y)) → 25.00%

print ("I : %,}", format (y)) → 2500000000 %

: → digits representation

k = 1000000000

print ("%,d", format (k)) → 1,000,000,000

percentage specifiers in python

%d → integer

%s → string

%f → float

%x → hexadecimal

%o → octal

Ex:- a = False

print ("a = %d" %a) → 0

a = 10.78

print ("a = %d" %a) → 10

hexadecimal

b = 45

print ("b = %x" %b) → 2d hexadecimal 2d

16|45
2|13
2d

octal

b = 45

print ("octal b = %o" %b) octal 55

8|45
5|5

Conversion of languages Using String comparision

Ex:- S1 = "BuSSE"

S2 = "BuBE"

print (S1.lower())

print (S2.lower())

print (S1.casefold())

print (S2.casefold())

If S1.casefold() == S2.casefold():

print ("values are equal")

else:

print ("values are not equal")

If S1.lower() == S2.lower():

print ("values are equal")

else:

print ("values are not equal")

ß → ss in german language

O/P

busse
buße
busse
busse
values are equal
values are not equal

* casefold() is used from to convert from upper case to lower fo all the language.

diff. b/w casefold & lowercase is lowercase understand only english lang. but casefold() under - stands all the lang.

* lowercase is used to convert the string from uppercase to lowercase fo only english lang.

Accessing the individual character in string in forward direction and reverse direction

Ex:- S = "abcdefgtech"

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

for i in reversed(S):
    print(i)
```

O/P

forward direction: a b c d e f g t e c h → reverse direction: h c e t g f e d c b a

Split() in strings

Split() is used to split the string at the specified position

Ex:- S = "abc for tech"

print (S.split())

S = "'abc', 'for', 'tech', 'training'"

print (S.split(','))

S = "'abc', 'for', 'tech', 'training'"

print (S.split(',', maxsplit=2))
=1

O/P

['abc', 'for', 'tech']

["'abc'", "'for'", "'tech'", "'training'"]

["'abc'", "'for'", " 'tech', 'training'"]

["'abc'", "'for', 'tech', 'training'"]

Checking the existence of a str. (start,end) of a string

Ex:- S = "abcfortech"
print (S.startswith ('a')) → True
print (S.startswith ('ab')) → True
print (S.endswith ('h')) → True
print (S.endswith ('f')) → False

Concatination of two strings & merging of strings

There are 4 different ways.

1) + operator
2) format ()
3) join ()
4) f string literal

Ex:- S1 = "abc"
S2 = 'tech'
# using + operator
S3 = S1 + S2
print (S3) → abctech
# using format()
S4 = "{}{}". format (S1, S2) "{} {}" → abc tech
print (S4) → abctech
# using join()
S5 = "".join ([S1, S2]) " " → abc tech
print (S6) → abctech
# using f string literal
S6 = f"{S1}{S2}".
print (S6) → abctech

Aligning a String

Ex:- S = "abc"
print ("|{0:<10}|". format (S)) → left alignment
print ("|{0:>10}|". format (S)) → right alignment
print ("|{0:^10}|". format (S)) → center alignment

O/P
|abc |
| abc|
| abc |

Collection module in python

1) Deque
2) Named tuple
3) Ordered dictionary
4) Default dictionary
5) Chain Map
6) Counter

Deque :-

Deque in a class which is present with in collection modules in order to store the data deque will make use of a data struct - ure called as "double ended queue"

* Deque is used to add the data at both front end as well as rare end efficiently

Ex:- import collections
dq = collections.deque()
print (dq) → deque ([])
print (type (dq)) → <'collections.deque'>
dq.append ([10])
dq.append ([20])
print (dq) → deque ([10, 20])
dq.appendleft (30)
print (dq) → deque ([30, 10, 20])
dq.extend ([40, 50, 60])
print (dq) → deque ([30, 10, 20, 40, 50, 60])
dq.extendleft ([99, 77, 88])
print (dq) → deque ([88, 77, 99, 30, 10, 20, 40, 50, 60])
dq.pop ()
print (dq) → deque ([88, 77, 99, 30, 10, 20, 40, 50])
dq.popleft ()
print (dq) → deque ([77, 99, 30, 10, 20, 40, 50])
dq.remove (10)
print (dq) → deque ([77, 99, 30, 10, 20, 50])

2) Named tuple

Normal tuple

~~normal_tuple~~

```
normal_tup = (55, 100, 75)
print (normal_tup) → (55, 100, 75)
print (type (normal_tup)) → tuple
```

Named tuple

```
import collections
named_tup = collections.namedtuple ('color', ['red', 'green', 'Blue'])
print (named_tup) → class '__main__.color'
print (type (named_tup)) → < class 'type' >
data = named_tup (55, 100, 75)
print (data) → color (red = 55, green = 100, Blue = 75)
print (type (data)) → class '__main__.color'
print (data [0]) → 55
```

* A named tuple is similar to a normal tuple, using named tuple we can provide the meaning for the data present within a tuple.
* Named tuple is a function which is present within collection module & it is used to increase code readability.

3) OrderedDictionary

OrderedDictionary is a class present within a collections module.

* In the versions of python < 3.6 the order of insertion is never preserved. i.e., the data (key-value pairs) will be obtained in unordered manner. So ~~ever~~ in order to overcome from this disadvantage will make use of ordered dictionary
* In the version of python > 3.6 there is no necessity of using ordereddictionary ∴ because normal dictionary is going to preserve the order by insertion.

Python < 3.6

```
d = {1:'a', 2:'b', 3:'c'}
print (d) → {1:'a', 2:'b', 3:'c'}
```

python < 3.6

```
d = {1:'a', 2:'b', 3:'c'}
print (d) → {2:'b', 3:'c', 1:'a'}
```

```
import collections
od = collections.OrderedDict()
print (od) → OrderedDict()
print (type (od)) → < class 'collections.OrderedDict' >
od [1] = 'a'
od [2] = 'b'
od [3] = 'c'
print (od) → OrderedDict ([(1,'a'), (2,'b'), (3,'c')])
print (type (od)) → < class 'collections.OrderedDict' >
```

4) Default Dictionary

Default dictionary is a class present with in a collection module

* it is used to create only keys and in the later stage if you providing the value

Syntax: collections.defaultdict (sequence ([int, tuple, set, dict, str, int...]))

Ex:-

```
import collections
dd = collections.defaultdict (int)
print (dd) → defaultdict (< class 'int' >, {})
print (type (dd) → < class 'collections.defaultdict' >
dd [1]
dd [2]
dd [3]
dd [4]
print (dd) → defaultdict (< class 'int' >, {1:0, 2:0, 3:0, 4:0})
dd [3] = 99
print (dd) → defaultdict (< class 'int' >, {1:0, 2:0, 3:99, 4:0})
```

5) ChainMap

ChainMap is used to merge multiple dictionaries and to store the data in a single platform

Ex:-

```
food = {'biscuits': 100, 'chocolates': 75, 'cake': 200}
toys = {'car': 60, 'bike': 80, 'barbie': 10}
clothes = {'pant': 100, 't-shirt': 90, 'shirt': 30}
import collections
cm = collections.ChainMap (food, toys, clothes)
print (cm)
print (type (cm)) → < class 'collections.ChainMap' >
```

```
print ( cm ['shirt']) -> 30
cm ['shirt']= 100
print ( cm ['shirt']) -> 100
```

O/P chainMap ({'biscuits':100, 'chocolates': 75, 'cake' : 200}, {'car': 60, 'bike': 80, 'barbie': 10} , {'pant': 100, 't-shirt': 90, 'shirt':30})

6}- Counter

Counter is a class present with in collections module

* it is used to count occurence of data
  set is not possible because duplicates not allowed

Ex:-

```
import collections
lst= [10, 10, 20, 30, 40, 10, 20, 30, 30]
c= collections . counter ( lst )
print (c) -> counter ({10:3, 30:3, 20:2, 40 :1 })
print ( type (c)) -> <<class 'collection. counter'>
c= collectione. counter ({'a': 4, 'b': 3, 'c': 9})
print (c) -> counter ({'c':9, 'a': 4, 'b' : 3 }
c= collections. counter (one= 23, two= 7, three= 67).
print (c) -> counter ({'three': 67, 'one'= 23 , 'two'= 7 })
c= collectione . counter ("abcfortech")
print(c) -> counter ({'c': 2 , 'a'=1 , 'b'=1 , 'f'= 1 , 'o' = 1 , 'r' = 1, 't' = 1 , 'e'=1, 'h'
c.update ({'a':9, 'b':6})
print (c) -> ({'a':10, 'b'=7 , 'c': 2 , 'f'=1, 'o'=1 , 'r'=1, 't'=1 , 'e'=1, 'h'
```

wap to access keys & values present with ina dictionary using counter

Regular Expression (Regex)

Regular Expression are used inorder to match the pattern

Notepad++

abc for techtraining
Ms. Bindu
// Python program\\
aa bbcc
ab bb ccc
aaaa bccc
aaaaaaa ccc bbb

Find element {2} find
{2,5} findnext

⊙ regular Expression

a -> literally it matches only a
b -> literally it matches only b
aa -> literally it matches only aa
a|b -> literally it matches only either a or b
. -> it matches all character (period)
\ -> escape the meaning
\. -> matches .
\\ -> Matches \
/ -> matches /
M (or) m -> matches both m & M

Quantifiers

* -> 0 (or) many occurance
+ -> 1(or) many occurance
? -> 0 (or) 1 occurance
{2} -> exactly 2
{2,5} -> b/w 2 to 5
^ -> start with
$ -> ends with

write a regular expression to match airtel nos:

9900217329
9900517369
9900618191
9900767776
99006776

R.E: 9900\d{6}

write a regular expression to identify indian nos

+91- 8553517329
+91- 855376891
+91- 855366362
+91 - 85517736
+91 - 857629199

R.E: \+91-\d{10}

\d -> identifies digits [0-9]
\D -> apart from, identifies all the character

Scanned by CamScanner

2) Ex1: abc for techtraining

123456789

!@#$%^&*()_+-=<>[]:"|;'\|<>?,./

\w → alphabets identifies alphabets, digits & underscore.

\W → special character except underscore and whitespaces

\s → white spaces

\S → not a white spaces

Ex2:- [abc] for tech → [abc]for[abc]

abc for tech → for[abc]

tech [abc] for → [abc]for

tech abc for → for[abc]

[\Babc] [abc\B] [\Babc] [abc\B] [match this not found]

\Babc\b

\b → word boundary

\B → not a word boundary

Matching a pattern using character class

[ ] → character class

[xy] → x or y

[\w\d] → single occurence of alphabets and digits

[\w\d]+ → many occurence of alphabets and digits

3) grouping of regular expression

Ex:- [sandeshs.abc@gmail.com]

[Saudesh&abc-san@yahoo.com]

[sandesh_.@abc]fatech.com]

R.E. [a-z A-Z 0-9 _.\-\$]+@[a-z]+[.][a-z]{3}]+

\w \W

* underscore is not having any special meaning

Raw String :-

Normal string

Ex:- s='\tstring'

print(s)

O/P String (after 4 spaces)

Creation of raw string

s=r'\tstring'

print(s)

\tstring

s='\nstring'

print(s)

O/P saudesh

s=R'\nabc'

print(s)

\nabc

Raw string in python is created using 'r' ' ' and or R' ' it is used to escape the meaning of '\'.

Re module [Regular Expression Module]

In order to perform pattern matching the below mentioned functions may be used.

1) search()
2) Match()
3) sub()
4) findall()
5) compile()

1) Search() : search() is used to search the character at any occurence (beginning and ending)

import re

s='abcxyz'

print(re.search(r'abc',s)) → <re.Match object; span=(0,3), Match='abc'>

print(s[0:3]) → abc

print(re.search(r'xyz',s)) → <re.Match object; span=(3,6), Match='xyz'>

print(s[3:6]) → xyz

print(re.search(r'pqr',s)) → None

2) Match() : Matches is used to search only the character at the beginning.

import re

s='abcxyz'

print(re.match(r'abc',s)) → <re.Match object; span=(0,3), match='abc'>

print(s) → abc

print(re.match(r'xyz',s)) → None

print(s) → xyz

print(re.match(r'pqr',s)) → None

Note:- Search() is used to match a single line string. if you want to match multiline string then we should make use of findall()

2) it matches only 1 line

```
import re
s = '''sandeshhs.abc@gmail.com
Sandesh6460-sau@yahoo.com
sandesh_@abcfoltech.com'''
print(re.search(r'[a-zA-Z0-9_\-\.]+@[a-z]+\.[a-z{3}]+',s))
```

1 O/P <re.match object; span=(0,23), match='sandesh.abc@gmail.com'>

Ex-2

```
import re
S = '''sandeshhs.abc@gmail.com
Sandesh6460-sau@yahoo.com
sandesh_@abcfoltech.com'''
print(re.findall(r'[a-zA-Z0-9\-_\.]+@[a-z]+[a-z{3}]',s))
```

O/P
['sandeshhs.abc@gmail.com', 'Sandesh6460-sau@yahoo.com', 'sandesh_@abcfoltech.com']

* Grouping the data within a raw string

```
import re:
s = '''sandeshhs.abc@gmail.com
Sandesh6460-sau@yahoo.com
sandesh_@abcfoltech.com'''
print(re.search(r'([a-zA-Z0-9_\-\.]+)@([a-z]+\.[a-z{3}])',s))
```

O/P ['sandeshhs.abc', 'gmail.com']

3) Sub()

it is used to substitute the data within string

general syntax sub(pattern, substitution, string)

Ex:-1

```
import re
s = '''sandeshhs.abc@gmail.com
Sandesh6460-sau@yahoo.com
sandesh_@abcfoltech.com'''
s1 = re.sub(r'@[a-z]+\.[a-z{3}]+','@outlook.com',s)
print(s)
print(s1)
```

removing the domain name using sub()

```
s2 = re.sub(r'@[a-z]+\.[a-z{3}]+','',s)
print(s2)
```

O/P

```
sandeshhs.abc@outlook.com
Sandesh6460-sau@outlook.com
sandesh_@outlook.com
sandeshhs.abc@gmail.com
Sandesh6460-sau@yahoo.com
sandesh_@abcfoltech.com
sandeshhs.abc
Sandesh6460-sau
sandesh_
```

Ex-2: Seperating username and domainname using sub()

```
import re
s = '''sandeshhs.abc@gmail.com
Sandesh6460-sau@yahoo.com
sandesh_@abcfoltech.com'''
s1 = re.sub(r'([a-zA-Z0-9._\-]+)@([a-z]+\.[a-z]+)',r'Name:\1 Domain Name:\2',s)
print(s1)
```

O/P

```
Name:sandeshhs.abc Domain Name:gmail.com
Name:Sandesh6460-sau Domain Name:yahoo.com
Name:sandesh_ Domain Name:abcfoltech.com
```

4) Compile()

compile() is used to store regular expression within a variable if in case one regular expression has to be used multiple times then we should make use of variable.

Ex:-

```
import re
s = '''sandeshhs.abc@gmail.com
Sandesh6460-sau@yahoo.com
sandesh_@abcfoltech.com'''
reg = re.compile(r'[a-zA-Z0-9_\-\.]+@[a-z]+\.[a-z{3}]+')
print(reg.search(s))
print(reg.findall(s))
```

2) o/p <re.Match object; span=(0,23), Match='saudeshhs.abc@gmail.com'>

['saudeshhs.abc@gmail.com', 'Saudesh6460-sau@yahoo.com', 'saudesh_-@abcfotech.com']

1) wap to convert a given function name from camelcase to snake case

2) wap to convert a given function name from snakecase to camel case

hint:- 1) bubblesoft → camelcase → java
bubble_soft → snake case → python

2) car_bike_auto_bus → snake case
carBikeAutoBus → camelcase

Split()

import re
s= '''saudeshhs.abc@gmail.com
Saudesh6460-sau@yahoo.com
$audesh_-@gmabcfotech.com'''
print(re.split(r'@',s))

o/p

3) ~~['saudeshhs.abc', 'Saudesh6460-sau', '$audesh_-']~~

['saudeshhs.abc', 'gmail.com\n Saudesh6460-sau', 'yahoo.com\n $audesh_-', 'abcfotech.com']

* Split() is used to split the data at specified value.

general syntax 'split(pattern, string)'

Grouping & Match Object

import re

Ex:-1 s= '''saudeshhs.abc@gmail.com
Saudesh6460-sau@yahoo.com
$audesh_-@abcfotech.com'''

m=re.search(r'[a-zA-Z0-9\.\-_]+@[a-z]+\.[a-z]{3}',s)
print(m) <re.Match object; span=(0,23) match='saudeshhs.abc@gmail.com'>
print(m.start()) → 0
print(m.end()) → 23

print(m.group()) → saudeshhs.abc@gmail.com
print(m.group(0)) → saudeshhs.abc@gmail.com
print(m.group(1)) → error [no such group]
print(m.group(2)) → error [no such group]

* by default the data will be grouped and the index happens to be zero. i.e., the complete regular expression will be grouped.

* In order to overcome from this error we should group the data using regular expression.

Ex:-2 import re
s= '''
m= re.search(r'([a-zA-Z0-9\.\-_]+)@([a-z]+\.[a-z]{3})',s)
print(m.group(1)) → saudeshhs.abc 0
print(m.group(2)) → gmail.com 1 2
(1)@(2)

start() → used to fetch starting index of string
end() → " ending index of string

Matching the multiple occurence of a word present within a string.

s= 'abc for tech abc for'
regex = re.compile(r'\babc\b', re.I)
print(regex.search(s)) → <re.match object; span=(0,3), match='abc'>
print(regex.findall(s)) → ['abc', 'abc']

program:

1) wap to match the initial word present within a string

~~s= 'python abc for tech'~~
~~regex = re.compile(r'[P]+', re.I)~~
~~print(regex.search(s))~~
print(re.

import re
s= 'python abc for tech'
print(re.search(r'\w+',s)) →
<re.Match object; span=(0,5) match='python'>

Note:- ^ → to match initial character \w → word ^\w
$ → to — " — last — " — \w$

Note:- if place holder(x,y) is used we should avoid quantifiers (+,*,?)

2) wap to match initial two character of a word present with in a string.

```
import re
s = 'python abc for tech'
print(re.search(r'\b\w{2}', s)) → ['py', 'ab', 'fo', 'te']
        findall
```

3) wap to match the date present with in a string

```
import re
s = 'python
abc
for
tech
25/10/2019'
print(re.findall(r'\d{2}/\d{2}/\d{4}', s)) → ['25/10/2019']
```

4) wap to validate a valid airtel nos. present with in a list.

```
import re
numbers = [9900567678, 9900567879, 9900676767, 8762223171]
for num in numbers:
    if re.search(r'9900\d{6}', str(num)):
        print("valid Airtel number")
    else:
        print("Invalid Airtel number")
```

O/P
Valid Airtel number
Invalid ——"——
Valid ——"——
Invalid ——"——

5) wap to match a word which starts with vowel

```
import re
s = "being Alive is the greatest gift of life"
print(re.findall(r'\b[aeiouAEIOU]\w+', s))
```

O/P
['alive', 'is', 'of']

6) wap to match whether the entered username is valid or not. username should consists of atleast one number and remaining character should be alphabets the length should be greater than or equal to 5

```
import re
def snakecase(n):
    data = input("Enter the username")
    if re.fullmatch(r'[a-zA-Z0-9]+', data):
        print("valid username")
    else:
        print("Invalid username")
```

7) write a regular expression to match the given pattern and convert the given I/P from snakecase to camelcase and camelcase to snakecase

```
import re
def snakecase(n):
    lst = re.sub(r'[a-z][A-Z]+', r'\1_\2', n)
    print(lst.lower())
name = input("Enter the function name in camelcase:\n")
snakecase(name)
```

O/P
bubble_sort (Camelcase to snake case)
bubble_sort

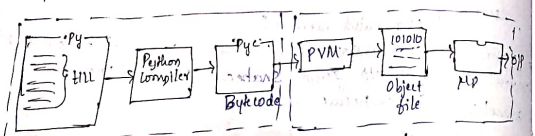
Snakecase to camel case without using regular expression

```
s = input("Enter the function name in snakecase:\n")
lst = s.split('_')
print(lst)
camelcase = ''
for i in range(len(lst)):
    if i == 0:
        camelcase = camelcase + lst[i]
    else:
        camelcase = camelcase + lst[i].title()
print(camelcase)
```

2) Builtin() → builtin → module
search()
match()
findall()
fullmatch()
compile()
split()
sub()

Exception handling in python

print "Hello" → syntax error
open("sandesh.txt") → file not found error



Compilation
Syntax error

Execution
Runtime error
or
Exception

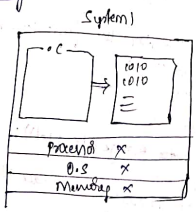
3) web based application → chrome, google
client server application → eshatsapp
Desktop application → VLC player, mx player.

graphical user interface [GUI]

* client Machine server

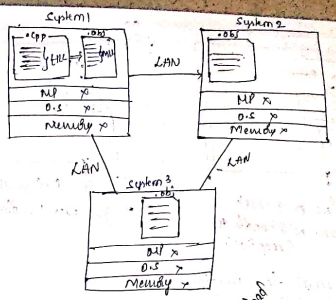
Importance of Exception handling

1990 (C)
Dennis Ritchie
[No exception handling]

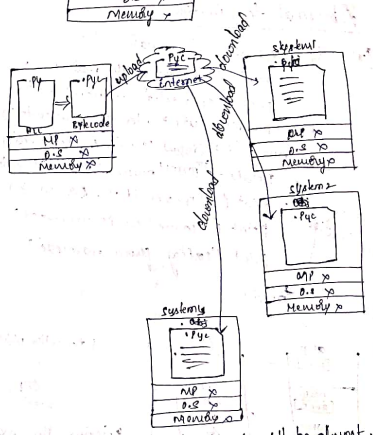


But now a days Exception handling is there because we are using C language in Embedded application

1980 C++
Stroustrup
ten important of Exception handling

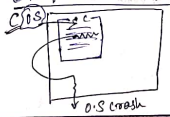


1990 Python
Guido van Rossum
[Exception handling]

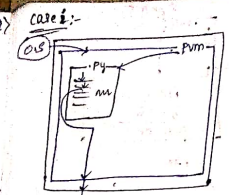


after 1990 the applications that are developed will be almost web based application as well as client server application. So, if we any programming language is used to develop those application should compulsorily support Exception handling.

Exception handling in python

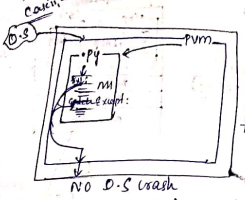


2) case i:-



Abnormal termination No O.S crash (pvm will handle the Exception)

case ii:-



Normal termination

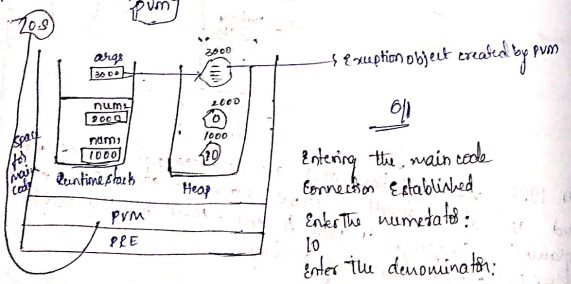
No O.S crash (programmer handled exception)

Abnormal termination

Ex:-
```
if __name__ == '__main__':
    print ("entering the main code")
    print ("connection is established")
    num1 = int(input("enter the numerator"))
    num2 = int(input("enter the denominator"))
    result = num1/num2          (Exception/runtime error)
    print ("the result is :", result)
    print ("connection is terminated")
    print ("exiting from main code")
```

Abnormal termination → pvm

3)



o/p

```
Entering the main code
Connection Established
Enter the numerator:
10
Enter the denominator:
0
```

Exception : ZeroDivisionError : division by zero

Note:- unexpected / unwanted event that occurs during execution time which leads to abnormal termination is called as exception.

if in case user not handling the exception then pvm will handle the exception by creating exception object whose address will be thrown to a line where exception occurred.

Normal termination / Graceful termination

general syntax for exception handling

```
try:
    ==  } Risky code
except:
    ==  } exception handling code
```

```
if __name__ == '__main__':
    print ("entering the main code")
    print ("connection established")
    try:
        num1 = int(input("enter the numerator"))
        num2 = int(input("enter the denominator"))
        result = num1/num2
        print ("the result is:", result)
    except ZeroDivisionError:
        print ("enter a Non zero number")
    print ("connection terminated")
    print ("Exiting the main code")
```

o/p

```
Entering the main code
Connection established
Enter the numerator:
10
enter the denominator:
0
enter a nonzero numerator number
connection terminated
exiting the main code
```

In the above program a single except block is used so the complete clarity about exception will not be provided as well as some other exception occured it has to be handled by pvm so inorder to overcome this we will make use of multiple except block

```
if __name__ == '__main__':
    print("Entering the maincode")
    print("connection established")
    try:
        num1 = int(input("Enter the numerator:\n"))
        num2 = int(input("Enter the Denominator:\n"))
        result = num1/num2
        print(f'The result is: {result}')
    except ZeroDivisionError:
        print("Enter a Nonzero number")
    except ValueError:
        print("Enter the integer value")
    except NameError:
        print("some problem occured\n Enter the valid input")
    except:
        print("some problem occured")
    print("connection terminated")
    print("Exiting the maincode")
```

Grouping Multiple Exceptions within a single except block

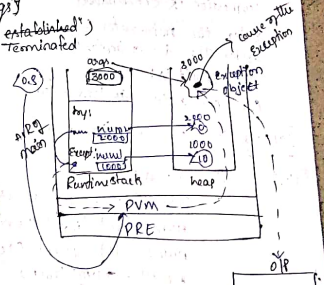
Ex:-
```
if __name__ == '__main__':
    print("entering the main code")
    print("connection established")
    try:
        num1 = int(input("enter the numerator:"))
        num2 = int(input("enter the denominator:\n"))
        result = num1/num2
        print(f'The result is: {result}')
    except(ZeroDivisionError, NameError, ValueError):
        print("Enter a Nonzero number & integer value in valid input")
    except:
        print("some problem occured")
    print("connection terminated")
    print("Exiting the maincode")
```

Aliasing the Exception object and retrieving the cause of an Exception

```
if __name__ == "__main__":
    print("connection is established")
    try:
        num1 = int(input('Enter the numerator'))
        num2 = int(input('Enter the denominator'))
        result = num1/num2
        print(f'The result is: {result}')
    except Exception as e:
        print(e.args)
    print("connection is established/Terminated")
```

o/p

connection is established
Enter the numerator 10
Enter the denominator 0
('division by zero')
connection is terminated



Note:- Exception object will be having an address with in a reference variable args
using that reference variable we can fetch the exact cause of the Exception as shown above.
Aliasing can be done for an exception using the keyword 'as'

Optional else block:

whenever an exception occured some line with in a program will not get executed.
Such line should be placed with in else block, in other words if the user is providing proper i/p the control will be given to else blocks.

```
if __name__ == "__main__":
    print ('connection is established')
    try:
        num1 = int(input ("Enter the numerator"))
        num2 = int (input ("Enter the denominator"))
        result = num1/num2
    except Exception as e:
        print (e.args) → Pvm get control to create Exception object
    else:
        print ("The result is : {result}")
    print ("The connection is terminated");
```

O/P

connection is established
enter the numerator 10
enter the denominator 2
The result is : 5.0
The connection is terminated

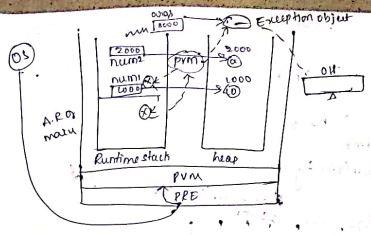
Propogation of Exception object

Ex:-1

```
def alpha():
    print ("connection2 is established")
    num1 = int (input ('enter the numerator'))
    num2 = int (input ('enter the denominator'))
    result = num1/num2
    print (f"The result is : {result}")
    print ("connection 2 is terminated")
print ("connection1 is established")
alpha()
print ("connection1 is terminated")
```

O/P

connection1 is established
connection 2 is established
enter the numerator 10
enter the denominator 0
Traceback (most recent call last):
File "C:\Users\----", line 9
alpha()
File "C:\-------", line 5
result = num1/num2 ZeroDivisionError: division by zero



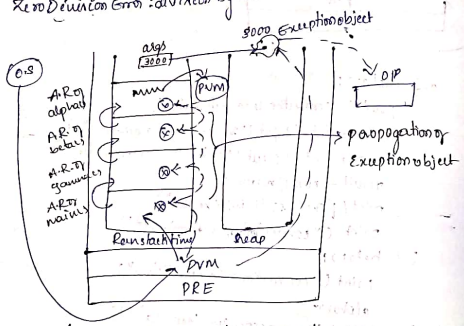
Ex:2

```
def alpha():
    print ("connection 2 is established")
    num1 = int (input ("the enter the numerator"))
    num2 = int (input ("enter the denominator"))
    result = num1/num2
    print (f"The result is: {result}")
    print ("connection 2 is terminated")
def beta():
    print ("connection 1 is established")
    alpha()
    print ("connection 1 is terminated")
def gamma():
    print ("connection 0 is established")
    beta()
    print ("connection 0 is established")
print ("connection 1 is established")
gamma()
print ("connection 1 is terminated")
```

O/P

connection 1 is established
connection 0 is established
connection 2 is established
enter the numerator 10
enter the denominator 0
Traceback (most recent call last):
File "C:\---", line 17
gamma()

File "C:\user - - - ", line 14
beta()
File "c:\user - - - ", line 10
alpha()
File "C:\user - - - ", line 5
3000
result = num/num2
ZeroDivisionError: division by zero



Note:- * whenever an exception occured with in a function and the function is not handling the Exception then the control will be given to pvm, then pvm create an exception object (E.O) and address of E.O. will be thrown to the line where exception occured and again that address (which contain the cause of the exception) encounters pvm. now pvm will check the complete function hierarchy with in stack.

* that means the exception object (address) will propogate below the lower stack of the function is mechanism is referred as propogation of exception object.

Raise keyword in Exception

way to raise an exception using raise keyword if the user if is not entering the odd number which will begin with 2's vowels

Raise keyword is used to raise the Exception after the user not

```
import re
def check_vowel(n):
    pp if re.search('^[aeiouAEIOU]', n):
        print("Entered word starts with vowels")
    else:
        raise Exception("Entered word will not starts with vowels")
n = input("Enter the word")
check_vowel(n)
```

o/p

```
Enter the word saudesh
Traceback (most recent call last):
  File "ex.py", line 9, in <module>
    check_vowel(n)
  File "ex.py", line 6, in check_vowel
    raise Exception("entered the word will not starts with vowel")
Exception: entered word will not starts with vowels.
```

Ex2: The Exception can be raise with in else block, without if as shown below.

NAP to collect the input from the user (odd no.) and convert it into even no. & if the user is entering even no. then raise the exception with in if block.

```
do import re
    def odd_even(n):
        if n%2==0:
            raise Exception('Already the no. is even')
        else
            return n+1
n = int(input("enter the odd no."))
res = odd_even(n)
print('The number is:', res)
```

o/p

```
Enter the odd no. 2
Traceback ("most recent call last):
  File "Ex.py" line 8, in <module>
    res = odd_even(n)
  File Ex.py' line 3, in odd_even
```

raise Exception("Already the no. is even")
Exception: Already the no. is even

Enter the valid no. 3
The number is 4

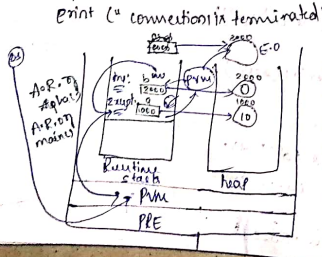
Handling the Exception

There are two ways of Handling the Exception

1. Handling the exception using try except block
2. Rethrowing an exception & Handling the exception using try except finally block.

Ex1:- Handling the exception using try Except block

```
def alpha():
    print("connection 2 is established")
    try:
        print("enter the numerator")
        a = int(input())
        print("enter the denominator")
        b = int(input())
        c = a/b
        print(c)
    except Exception as e:
        print(e.args)
    print("connection 2 is terminated")
print("Connection 1 is established")
alpha()
print("connection 1 is terminated")
```



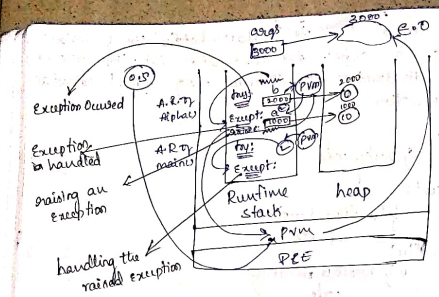
O/P
connection1---
connection2---to
Enter the numerator 10
Enter the denominator 0
(division by zero)
connection 2 terminated
---11---2---

Note:- In the above example alpha() handled the exception. the main() is not notified the about the exception. in orders to overcome the disadvantage we will make use of concept called as rethrowing an exception.

```
try:
    ;; risky code
except:
    ;; handling exception code
finally:
    ;; resource releasing code
```

Ex:2 Handling the Exception using try, except, finally block as well as raising the Exception using raise keyword.

```
def alpha():
    print("connection 2 is established")
    try:
        print("Enter the numerator")
        a = int(input())
        print("Enter the denominator")
        b = int(input())
        c = a/b
        print(c)
    except Exception e:
        print(e.args)
        print('alpha() handled the exception')
        raise e
    finally:
        print("connection 2 is terminated")
if __name__ == '__main__':
    print("connection 1 is established")
    try:
        alpha()
    except:
        print('exception is handled by alpha() and it is notified to main()')
    print('connection 1 is terminated')
```



O/P

~~Error~~ connection 1. is Established
connection 2 is established
Enter the numerator 10
Enter the denominator 0
('division by zero')
Alpha() Handled ~~Exception~~
~~Alpha~~ ~~this~~ ~~Exception~~.
Connection 2 is terminated
Exception is handled by Alpha() and it intimated by main()
connection is terminated.

Stack overflow → Stack overflow is an ~~Error~~ exception which cannot handled by except block.

Java

```
class Exmp
{
  void func()
  {
    func();
  }
}
class Demo
{
  psvm()
  {
    Exmp e=new Exmp();
    try
    {
      e.func();
    }
    catch(Exception e)
    {
      S.O.println("Error in handle");
    }
  }
}
```

O/P → Stack overflow

Python

```
def func():
    func()
if __name__=='__main__':
    try:
        func()
    Except Exception as e:
        print (e.args)
```

O/P maximum recursion depth exceeded.

Java	Python
* Stack overflow is an error in Java	* stack overflow is an exception in python.
* cannot be handled.	* it can be handled

~~Invalid~~

```
try:
    # stinky code
except:
    # handling exception code
else:
    # valid O/P blocks
finally:
    # resource releasing code
```

Invalid application:

```
try:                except:          else:          finally:
except:             except:
else:               finally:
```

Valid Application:

```
try:
except:

try:
finally:

try:
else:

try:
except:
else:

try:
except:
else:
finally:

try:
except:
try:
except:
else:
finally:

try:
except:
try:
except:

try:
except:
except:

try:
except:
else:
try:
except:

try:
except:
try:
except:
else:
finally

try:
finally
try:
except:
```

Local variables

Variables which are declared with in a function is referred as local variables.

* The scope of local variables will be limited function itself.

Ex:-
```
def display_data():
    name = 'saudesh'
    print(f'Good evening {name}')
if __name__ == '__main__':
    # display_data()
    print(name)
```

O/P
Good evening saudesh
Error

name is local variable its scope is till the end of within the function

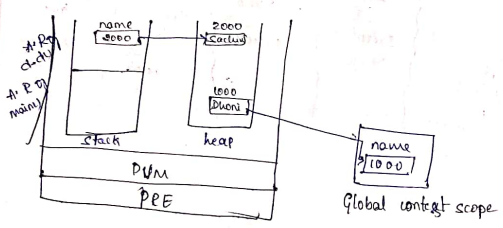
Global variable.

variables which are declared outside the function is called as global variable.

* the scope of global variable will be throughout the program but if there is any local variable with in function. priority will be given for local variable.

Ex:-
```
name = "dhoni"
def display_data():
    name = 'sachin'
    print(name)
if __name__ == '__main__':
    display_data()
    print(name)
```

O/P
Sachin
dhoni



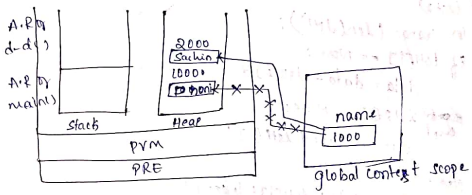
The preference of a global variable within global context scope

Ex2:- In order to access global variable with in a function we will make use of globals() [str]

* In order to declare a global variable within the function we will make use of a keyword called as global

Ex:-
```
name = "dhoni"
def display_data():
    global name
    name = 'sachin'
    print(name)
if __name__ == '__main__':
    display_data()
    print(name)
```

O/p
Sachin
Sachin
bcz



Ex:-3
```
name = 'dhoni'
def display_data():
    name = 'sachin'
    print(name)
    print(globals()['name'])
if __name__ == '__main__':
    display_data()
    print(name)
```

O/P
Sachin
dhoni
dhoni

Nested function

Creating a function within one more function is referred as nested function.

* always the inner function should be called with its outer function

Ex:-
```
def func1():
    print ('function1')
    def func2:
        print ('function2')
    func2()
if __name__ == '__main__':
    func1()
```

O/P
function1
function2

1) Wap to print a following O/P by using lst1 = ['A', 'app', 'a', 'da', 'kee', 'doc', 'a'] lst2 = ['ay', 'fo', 'ps', 'y', None, 'le', 'n']

```
lst1 = ['A', 'app', 'a', 'da', 'kee', 'doc', 'a']
lst2 = ['ay', 'fo', 'ps', 'y', None, 'le', 'n']
lst2 = reverse(lst2)
print(lst1)
print(lst2)
for i in range(len(lst1)):
    if lst1[i] == None:
        data = data + lst2[i] + ' '
    elif lst2[i] == None:
        data = data + lst1[i] + ' '
    else:
        data = data + lst1[i] + lst2[i] + ' '
print(data)
```

O/P
```
lst1 = [
lst2 = [
An Apple a day keeps doctor away
```

Customise Exception:-

There are 2 types of Exception
1) Built-in Exception
2) Customised Exception.

1) Built-in Exception

These are the exception generated by Pvm as of Exception.

Ex:- zero division error, Name error, indentation error, index error, etc.

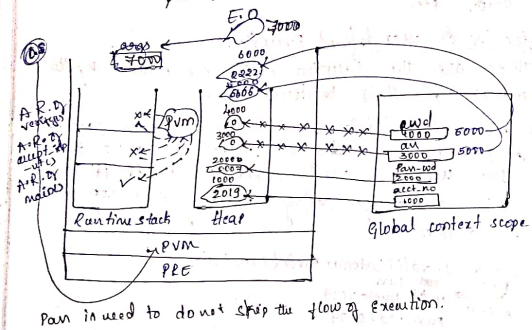
2) Customise or user defined Exception

These are the exception which are created by the programmer for their beneficial.

ex:- Wap inorder to validate the acc_no and pwd. (acct_num = 2019 and pwd = 0007) collect the acc_no. & pwd from the user and verify those details & if it is getting match print a message saying "collect_your money" else give two chances.

```
class InvalidCustomerError(Exception):
    def __init__(self):
        self.acct_num = 2019
        self.pwd = 0007
        self.an = 0
        self.pwd = 0
    def accept_input():
        global an
        global pwd
        an = int(input("Enter the account number"))
        pwd = int(input("Enter the password"))
    def verify():
        if acct_num == an and pwd == pwd
            print("collect your money")
        else:
            print("Invalid customer details")
            raise invalid customer error
if __name__ == '__main__':
    try:
        accept_input()
        verify()
    except:
        try:
            accept_input()
            verify()
```

except:
try:
accept_input()
verify
except:
print ("3 chances were given!!!")



Pass is used to do not skip the flow of execution.

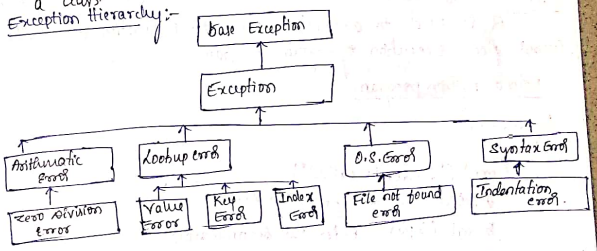
Ex:- a wap to validate the age for marriage as shown below.

class underage Error (Exception):
Pass
class overage Error (Exception):
Pass
age=0
def accept_input():
global age
age = int (input ("enter the current age"))
def verify():
if age < 18:
print ("you are not suitable for marriage")
raise underage Error
elif age > 60:
print ("your age is for Play with children not to marry")
raise overage Error
else:
print ("valid age for marriage")
raise Invalid age Exception Error

if __name__ == '__main__':
try:
accept_input()
verify()
except:
print ("invalid age" "enter valid age" "not valid time")
print e.args
"come with valid age"

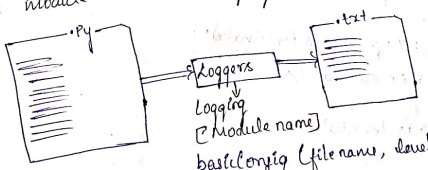
Note:- A customize Exception can be created using following ways
* declare a class in place of class name provide the Exception
* inherit from the base class Exception
* if any msg has to be displayed by the user place it within a class

Exception Hierarchy:-



Loggers:-

It is the process of creating an external file (.txt) inorder to record all the activities of a python program.
In python, inorder to make use of loggers, we have to use the module called as logging.



logging
[module name]
basicConfig (filename, level, filemode)
[function name]

Levels of Loggers

1) INFO
2) DEBUG
3) ERROR
4) CRITICAL
5) WARNING

Note:- In order to make use of loggers we should use a Built in function called as ~~basic config~~ basicConfig() which takes 3 argument

Syntax of basicConfig()

basicConfig (filename, level, filemode)

INFO:- we used to record or control flow of execution of program

It is used to record or store the information about the control flow execution of a python program.

Normal python program

```
def funcs():
    try:
        print ('funcs is called')
        a = int (input ("enter the numerator"))
        b = int (input ("enter the denominator"))
        c = a/b
    except:
        print ("some problem occured")
        print ("funcs handled the exception")
    else:
        print (f"The result is: {c}")
    finally:
        print ('funcs is terminated")
print ('mains is called")
funcs()
print ('mains is terminated')
```

O/P

```
mains is called
funcs is called
enter the numerator 10
enter the denominator 2
The result is : 5.0
funcs is terminated
mains is terminated
```

Python program using INFO Level logger

```
import logging
logging.basicConfig (filename='infolog.txt', level= logging.INFO,
                     filemode='w')
def funcs():
    try:
        logging.info('funcs is called')
        a = int (input ("enter the numerator"))
        b = int (input ("enter the denominator"))
        c = a/b
    except:
        print ("some problem occured")
        logging.info ('funcs handled the Exception')
    else:
        print (f"The result is: {c}")
    finally:
        logging.info("funcs is terminated")
logging.info ("mains is called")
funcs()
logging.info ("mains is terminated")
```

O/P

```
Enter the numerator 10
enter the denominator 2
The result is : 5.0
```

infolog.txt

```
INFO:root:mains is called
INFO:root:funcs is called
INFO:root:funcs is terminated
INFO:root:mains is terminated.
```

1) DEBUG:

It is used to store the input provided by the user.

* using Debug we can collect the information about the values stored within a variable

(usually for example deleted msgs of files/videos any information which we want to store permanently in server then we will use DEBUG ex:- whatsapp msgs stored in servers etc)

Normal python program.

```
def func():
    try:
        a = int(input("Enter the numerator"))
        print(a)
        b = int(input("enter the denominator"))
        print(b)
        c = a/b
        print(c)
    except:
        print("some problem occured")
func()
```

O/P

```
Enter the numerator 10
10
enter the denominator 0
0
5.0
```

Python program using Debug level logger.

```
import logging
logging.basicConfig(filename='debug log.txt', level=logging.DEBUG,
                    filemode='b')
def func()
try:
    print a = int(input("Enter the numerator"))
    logging.debug(f"The value stored within a is : {a}")
    b = int(input("Enter the denominator"))
    logging.debug(f"The value stored within b is : {b}")
    c = a/b
    print(c)
    logging.debug(f"The result obtained is : {c}")
except:
    print("some problem occured")
func()
```

O/P

```
enter the numerator 10
enter the denominator 0
some problem occured
```

(if we run the program again)

```
Enter the numerator 10
Enter the denominator 2
the 5.0
```

debug.txt

```
DEBUG: root: The value stored within a is:10
DEBUG: root: The value stored within b is:0
DEBUG: root: The value stored within a is:10
DEBUG: root: The value stored within b is:2
DEBUG: root: The result stored obtained is:5
```

if we want to display both we will use append → 'a' mode of file

3) ERROR:-

It is used to store the information about the cause of the Exception within a file

Ex:-

```
import logging
logging.basicConfig(filename='Error log.txt', level=logging.ERROR,
                    filemode='a')
def func():
    try:
        a = int(input("Enter the numerator"))
        b = int(input("enter the denominator"))
        c = a/b
        print(c)
    except Exception as e:
        print(f"The cause of the exception is: {e.args}")
        logging.error(e)
func()
```

O/P

```
enter the numerator 10
enter the denominator 0
The cause of the exception is:
('division by zero')
enter the numerator 10
enter the denominator abc
```

error.txt

```
ERROR: root: division by zero
ERROR: root:
```

The cause of exception is:
Invalid

* In the above program using error function only the cause of the exception is fetched stored within .txt file.
* In order to fetch both the exception name as well as the cause of the exception we have to make use of exception() present within logging module as shown below.

```
import logging
logging.basicConfig(filename='errorlog.txt', level=logging.ERR
                    filemode='a')

def func():
    try:
        a=int(input('Enter the numerator'))
        b=int(input('enter the denominator'))
        c=a/b
        print(c)

    Except:
        print('some problem occured')
        Logging.exception('some problem occured')
func()
```

O/P

```
Enter the numerator 10
Enter the denominator 0/abc
some problem occured
```

errorlog.txt

```
ZeroDivisionError : Division by zero
Value Error : Invalid literal for
int()
```

Wap using loggers to stored the details of the variables along with time, filename, functionname

```
import logging
logging.basicConfig(filename='debuglog.txt', level=logging.DEBUG
                    filemode='a', format="time = %(asctime)s, filename=
                    %(filename)s, funcname = %(funcName)s")

def func():
    try:
        a=int(input('Enter the numerator'))
        logging.debug(f'The value of a is:{a}')
        b=int(input('Enter the denominator'))
        logging.debug(f'The value of b is :{b}')
        c=a/b
        print(c)
        logging.exception debug(f'The result is:{c}')
    except:
        print("some problem occured")
        logging.exception("some problem occured")
func()
```

O/P

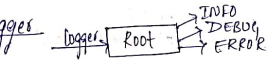
```
Entered the numerator 10
enter the denominator 0
some problem occured
```

debuglog.txt

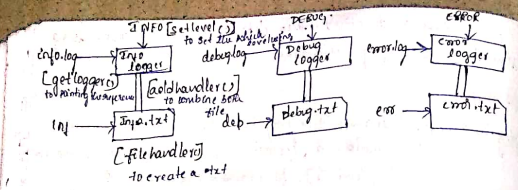
```
time = 2019-11-15 10:41:20,946, filename = demo.py,
funcname = funcname
time = 2019-11-15 10:41:20,947, filename = demo.py,
funcname, funcName
time, 2019-11-15 10:41:22,493, filename: demo.py,
funcname = funcName
```

```
Traceback (most recent call last)
  File "D:\ - - - ": line : 19
    C = a/b
ZeroDivisionError: division by zero
```

Root loggers



It is not possible to use multiple levels of logger using rootlogger so inorder to overcome this we have to created the loggers as shown below




```
import logging
info_log = logging.getLogger('info')
debug_log = logging.getLogger('debug')
error_log = logging.getLogger('error')
info_log.setLevel(logging.INFO)
debug_log.setLevel(logging.DEBUG)
error_log.setLevel(logging.ERROR)
inf = logging.FileHandler('info.txt')
deb = logging.FileHandler('debug.txt')
err = logging.FileHandler('ERROR.txt')
info_log.addHandler(inf)
debug_log.addHandler(deb)
error_log.addHandler(err)

def func():
    try:
        info_log.info('func is called')
        a = int(input('Enter the numerator'))
        debug_log.debug(f'the value of a is: {a}')
        b = int(input('enter the denominator'))
        debug_log.debug(f'The value of b is: {b}')
        c = a/b
    except:
        print('some problem occurred')
        error_log.exception()
    else:
        print(f'The result is: {c}')
        debuglog.debug(f'the value of c is: {c}')
    finally:
        print('func() is terminated')
        info_log.info('func() is terminated')
info_log.info('main() is called
func()
```

Map using loggers to display the warning msg for the user before providing the inputs.

```
import logging
info_log.info("main() is terminated")
```

o/p

```
Enter the numerator 10
Enter the denominator 2
The result is: 5.0
```

info.txt
main() is called
func() is terminated
main() is terminated

debug.txt
The value of a is: 10
The value of b is: 2
The result is c is: 5.0

error.txt

In java, to close the file we use close() - ie. closes

File operation in python

* getLoggers() is used to obtain the levels of loggers.
* setLevel() is used to set the level for the logger which is been created.
* filehandler() is used to create a file (.txt) or log file.
* add_handler() is used to combine the log file (.txt) with the levels of logger.

CRITICAL :-

If is used to display the critical msg: in other words before the exception is been occured. [before the error]

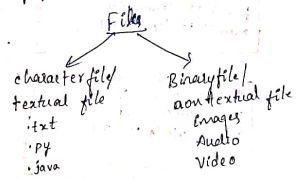
WARNING :-

If is used to display the warning msg regarding the data that to be stored within a variable [before the debug]

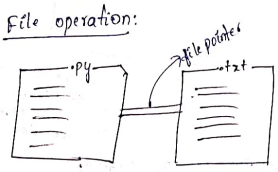
File operations in python or File handling in python.

2) In any programming language, if the data has to be stored permanently then it should support file operations.

* There are two types of files.
  1) character file
  2) Binary file.



File operation:



1) open the file [filepointer = open(file, mode)]
2) perform operations
3) close the file [close()]

* ~~open~~ then open() is used to open a file with in a python file
* close() is used to clone the file (.txt) within a file (.py)

Wap to read the content present with in a file (.txt) using a python program present with in same folder.

```
def read():
    f = open(file = 'sample.txt', mode = 'r')
    print ("file is successfully opened")
    print (f.read())
    f.close()
    print ("file got cloned")

if __name__ == '__main__':
    read_file()
```

Keyword argument
by default mode is read
('sample.txt', mode)
positional argument

O/P

```
file is successfully opened
abc for tech
sandeep
python
file got cloned.
```

Wap to read the content of file (.txt) which is present with in ~~one~~ one more folder using .py file

```
def read():
    f = open ('E:\Python\cricketer.txt')
    print ("file is opened")
    print (f.read())
    f.close()
    print ('file got cloned')
    print ('The file name is:', f.name)
    print ('The default file mode is', f.mode)

if __name__ == '__main__':
    read_file()
```

default argument

O/P

```
file is opened
sachin
virat
file got cloned
The file name is : E:\Python\cricketer.txt
The default file mode is r.
```

Note: By default the file mode happens to be read (r)

Context manager:- is used to close the file without mentioning the close()

general syntax:- with open (file, mode) as file pointer:

→ If a file is opened then compulsorily we have to close the file using close() because of which the complexity of the code will be increased

* by then in order to overcome this we can use context manager as shown below

```
with open('sample.txt') as f:
    print(f.read())
print(f.read()) ---> ValueError: I/O operation on closed file.
```

(∵ after closing the file we cannot read or print a file)

Readlines()

accessing the complete data present within a file

```
with open('sample.txt') as f:
    print(f.read())
```

accessing the first line present within the file

```
with open('sample.txt') as f:
    print(f.readline())
```

accessing the specified data present within the file using index

```
with open('sample.txt') as f:
    print(f.readline(3))
    print(f.readline())
```

O/P

abc fortech
Sandesh
python.
abc fortech → ②
abc
fortech ③

abc fortech
012 3

Cursor positions

* tell() is used to fetch the exact cursor position
* seek() is used to move the cursor at any position by providing the index as argument

```
with open('sample.txt') as f:
    print(f.tell())          → 0
    print(f.readline())      → abc fortech
    print(f.tell())          → 14
    print(f.readline(3))     → san
    print(f.tell())          → 17
    f.seek(0)
    print(f.tell())          → 0
    print(f.readline(2))     → ab
    f.seek(14)
    print(f.readline(7))     → sandesh
```

Note:- In read mode if a file is not present then it will lead to file not found error exception.

```
with open('companyname.txt') as file:
    print(f.read())
```

write mode :- It is used to write the content by creating a new file (.txt)
writing on a file using write() function

Ex:- cases: with open('companyname.txt','w') as f:
    f.write('abc')
    f.write('Infosys')
    f.write('Hcl')

O/P
abcInfosysHcl

Note In the above example (.txt) brand new file (.txt) is considered to perform write operation then the data not be overwritten rather it will get appended from the last cursor index

Ex:- * if an existing file is considered to perform write operation then the previous data present within that file will be over written

```
with open('companyname.txt') as f:
    f.write('wipro')
    f.write('LG')
    f.write('Tata')
```

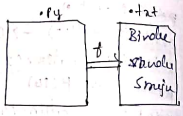
O/P
wiproLGTata

1 Case ii :- performing write operation using print()

```
with open ('names.txt', 'w') as f:
    print ('Sindu\n', file=f)
    print ('vandu\n', file=f)
    print ('sruju\n', file=f)
```

o/p

in a file it get recorded as Sindu
vandu
sruju



writelines()

writelines() is used to write the multiple lines of data within a file

* \n is used just to move the cursor into the next line.

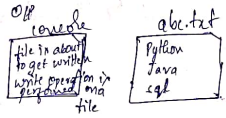
```
with open ('names.txt', 'w') as f:
    f.writelines(['abc\n', 'fdf\n', 'tech\n'])
```

o/p

Imp Wap to perform write operation upon a brand new file (Std. way of writing a file code)

```
def write():
    print ('file is about to get written')
    f = open ('abc.txt', 'w')
    f.write (['Python\n', 'Java\n', 'Sql\n'])
    print ("write operation is performed on a file")
    f.close()

if __name__ == '__main__':
    write()
```

o/p



Different modes under file operations.

Basic modes :-

1) r
2) w
3) a

advanced modes :-

r+
w+
a+

mode / operation	r	w	r+ (read + write)	w+ (write + read)	a (append)	a+ (append + read)
read	✓	×	✓	✓	×	✓
write	×	✓	✓	✓	✓	✓
create	×	✓	×	✓	✓	✓
erase	×	✓	×	✓	×	×
Position	start	start	start	start	start / end	start / end

Exclusive creation mode :- ('x')

using exclusive creation mode we cannot perform any operation on existing file if we try to perform then it leads to file exist error exception.

* we cannot perform read operation using exclusive creation mode.
* we can perform write operation only on a brand new file

why we use exclusive creation mode ?

mode / operation	x
read	×
write	✓
create	✓
erase	×
Position	start

Properties of a file

```
def get properties():
    f = open ('sample.txt', 'w')
    print ("The file name is :", f.name)
    Print ("The filemode is :", f.mode)
```

print ('is the file readable : ', f.readable())
print ('Is the file writable: ', f.writable())
print ('Is the file closed: ', f.closed)
f.close()
print ('Is the file closed : ', f.closed)

if __name__ == '__main__':
    getproperties()

O/P
The file name is : sample.txt
The file mode is : w
Is the file readable : False
Is the file writable : True
Is the file closed : False
Is the file closed : True

Prg2 File to File Transfer :-

Q) Write a python program to read the contents from one file and write those contents into one more file

```
def file_to_file():
    f1 = open ('input.txt', 'r')
    f2 = open ('output.txt', 'w')
    lst = f1.readlines()
    print (lst)
    for i in lst:
        f2.write(i)

if __name__ == '__main__':
    file_to_file()
```

O/P
['abcin', 'fgin', 'techin']

output.txt
abc
fg
tech

copying a binary file and creating the same binary file using file operation

* 'rb mode' used upon a binary file & the o/p will be obtained in the form of bytes.
* In order to convert bytes into actual (image.jpg) formate we have to make use of 'wb mode' as shown below

```
def copy():
    f1 = open ('darshan.jpg', 'rb')
    f2 = open ('newimage.jpg', 'wb')
    bytes = f1.read()
    f2.write(bytes)
    print ('Image Copied')
if __name__ == '__main__':
    copy()
```

O/P
image copied

Application
→ Stand alone (desktop Application
→ client server (python
→ web based

Setter - mutator
getter - accessor
garbage collects
is
when data don't have
reference it is collected by garbage

* constructor is also called as magic method
* when object is created, constructor get activated by default
* constructor is present
* constructor is used to initialize instance Variable by creating class parameter local variable

Object Orientation in Python

Programming language

- Structured programming language → C
- object oriented programming lang.
  - → Java
  - → Python
  - → Java script

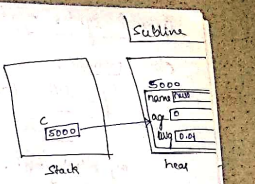
object → anything that has ... - thing & does some...
- has / properties → handled by datatype
- does / behaviour → handled by functions

* class is a blueprint of an object
* properties / variables → The different type of variables are:-
  * Instance variable [fields]
  * static variable [class variables]
  * local variables &
  * global variable
* Methods → types of methods
  * instance method
  * static method [utility method]
  * class method

Ex:-1 Java

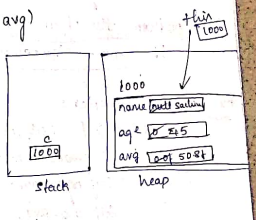
```
class cricketer
{
    String name;
    int age;
    float avg;
}
```
Instance variable

```
void disp() → instance method
{
    s.o.p(name);
    s.o.p(age);
    s.o.p(avg);
}
class CricketApp
{
    psvm(--)
    {
        cricketer c = new cricketer();
        c.disp();
    }
}
```



Ex:-2 class cricketer

```
{
    cricketer(String name, int age, float avg)
    {
        this.name = name;
        this.age = age;
        this.avg = avg;
    }
}
class CricketApp
{
    psvm()
    {
        cricketer c = new cricketer("Sachin", 45, 50.8f);
    }
}
```
this.name / this.age / this.avg → instance variable
name / age / avg → local variable
c → reference variable
"Sachin", 45, 50.8f → value for local variable



Note:-
* class is a blueprint of an object
* in the above ex1. variables declared within a method will be instance variable.
* in the above example default constructor will be used to initialize the properties of an object.

In the Ex-2, constructor is used to initialize the instance variables.

General syntax of a standardized python program.

```
class Example :
    def __init__(self, parameter1, parameter2...) : -> constructor
    ...
    def display(self) : -> instance method
    ...
if __name__ == '__main__' :
    obj = Example()
```

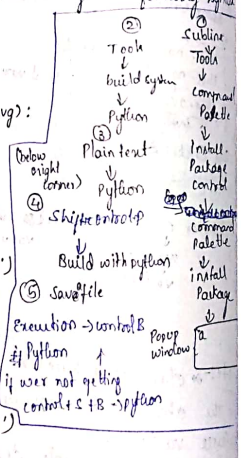
`obj` → reference variable pointing to an object; `Example()` → object

In python the object can be created using the following syntax

`Reference = <classname>()`

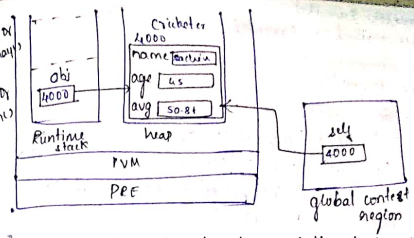
Ex:-1

```
class cricketer :
    def __init__(self, name, age, avg) :
        self.name = name;
        self.age = age;
        self.avg = avg;
    def display(self) :
        print('The name is {self.name}')
        print('The age is {self.age}')
        print('The avg is {self.avg}')
if __name__ == '__main__' :
    obj = cricketer(name='sachin',
                    age='45', avg='50.8')
    obj.display()
```



instance ⇒ an object ∴ state of an object ∴ properties of an object

of above Example is name, age, avg



Note :- ⇒ reference variable is a variable which will be pointing to the object & holds the address of that object

⇒ within any methods the first parameter will be self

⇒ 'self is a variable' which holds the address of the object that has been created.

⇒ self variable will be present within global context region.

Constructor :-

Constructor is a magic method present within python

⇒ always the name of the constructor happens to be '__init__'

⇒ constructor will be executed automatically after object creation.

⇒ the main purpose of constructor is to initialize the instance of object.

⇒ constructor will be executed only once after object creation.

Instance method()
Local variable
self.Instance variable
we can't access L.V (inside the IM())

Java we can create multiple constructors for same object (is called constructor overloading)

Python we can create only one object for same object name with reference

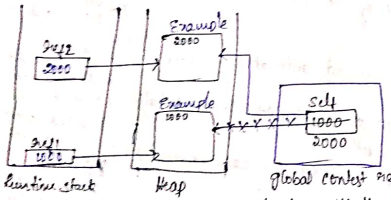
Ex 1:-

```
class Example :
    def __init__(self) :
        print('constructor() executed')
    def instance_method(self) :
        print('instance method() executed')
if __name__ == '__main__' :
    ref1 = Example()
    ref1.instance_method()
```

O/P constructor executed
instance method() executed

Ex:-2

```
class Example:
    def __init__(self):
        print('constructor() executed')
    def instance_method(self):
        print('instance method() executed')
if __name__ == '__main__':
    ref1 = Example() → constructor() executed
    print(id(ref1)) → 1000
    ref1.instance_method() → instance method() executed
    ref2 = Example() → constructor() executed
    print(id(ref2)) → 2000
    ref2.instance_method() → instance method() executed
    print(id(ref1)) → 1000
    print(id(ref2)) → 2000
```



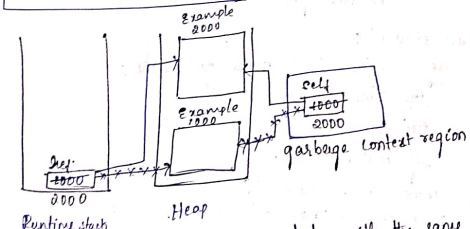
* we can create multiple objects with the same name as shown in the above example.
* self is a variable which will hold the address of current object only a single copy will be present within global context region
*

Ex3:-

```
class Example:
    def __init__(self):
        print('constructor() executed')
    def instance_method(self):
        print('instance method() executed')
if __name__ == '__main__':
    ref = Example()
    print(id(ref))
    ref.instance_method()
    ref = Example()
    print(id(ref))
    ref.instance_method()
    print(id(ref))
```

O/P

```
constructor() executed
1000
instance method() executed
constructor() executed
8000
instance method() executed
8000
```



* if multiple objects are created with the same name with the same reference initially the address of 1st object will be stored later it will be replaced by the address of 2nd object within that ref

The difference between Instance() & Constructor()

Instance()	(constructor())
* The name of the instance can be anything.	* The name of the constructor should be __init__
* Instance() will not be called automatically rather it should be called using the reference which pointing to the object	* constructor() will called automatically by pvm after object creation.
* For object instance() can be called n number of time	* per object constructor() will be called only once
* Instance() is used to initialize the business logic	* Constructor() is used to initialize the instance of object

Properties / Variables

① Instance Variables:

Instance Variables are such variables whose value keeps changing from object to object.

(or)

Instance variables are such variables which are present within a method & enclosed to within a defined class.

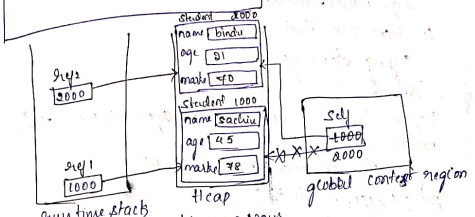
* using "__dict__" we can fetch the state of an object.

Ex:-1

```
class student:
    def __init__(self, name, age, marks):
        self.name = name;
        self.age = age;
        self.marks = marks;
if __name__ == '__main__':
    ref1 = student('sachin', age=45, marks=78)
    print(ref1.__dict__)
    ref2 = student('bindu', age=21, marks=70)
    print(ref2.__dict__)
```

O/P

{'name': 'sachin', 'age': 45, 'marks': 78}
{'name': 'bindu', 'age': 21, 'marks': 70}



Adding instance variable in 2 ways

① Adding the instance variables dynamically to the objects using the reference pointing to the object

Ex:-

```
class student:
    def __init__(self, name, age, marks):
        self.name = name
        self.age = age
        self.marks = marks
if __name__ == '__main__':
    ref1 = student('sachin', age=45, marks=78)
    print(ref1.__dict__)
    ref1.gender = 'male'
    print(ref1.__dict__)
```

(initially it was LV but after it is IV)

O/P

{'name': 'sachin', 'age': 45, 'marks': 78}
{'name': 'sachin', 'age': 45, 'marks': 78, 'gender': 'male'}

② Adding the instance variable using Instance method.

We can add instance variable using instance() by passing the argument by calling that instance method.

Ex:-
```
class student:
    def __init__(self, name, age, marks):
        self.name = name
        self.age = age
        self.marks = marks
    def instance_method(self, gender):
        self.gender = gender
if __name__ == '__main__':
    ref = student('name', 'sachin', age = 45, marks = 78)
    print(ref.__dict__)
    ref.instance_method(gender = 'male')
    print(ref.__dict__)
```

O/P

{'name': 'sachin', 'age': 45, 'marks': 78}
{'name': 'sachin', 'age': 45, 'marks': 78, 'gender': 'male'}

Modification of data which is present within instance variable

There are two ways to modify existing data within instance variable as shown below.

```
class student:
    def __init__(self, name, age, marks):
        self.name = name
        self.age = age
        self.marks = marks
    def instance_method(self, age):
        self.age = age
if __name__ == '__main__':
    ref = student('name', 'sachin', age = 45, marks = 78)
    print(ref.__dict__)
    ref.name = 'suhas' # using reference pointing to the object
    print(ref.__dict__)
    ref.instance_method(age = 20) # using instance method
    print(ref.__dict__)
```

O/P

{'name': 'sachin', 'age': 45, 'marks': 78}
{'name': 'suhas', 'age': 45, 'marks': 78}
{'name': 'suhas', 'age': 20, 'marks': 78}

Accessing the instance variable

* within the instance method if we want to access instance variable then we should make use of self which is pointing to the object.
* If we want to access the instance variable outside the class then we have to make use of the reference which is pointing to the object.
* by using the reference which is pointing to the object we can access the instance variable within the instance method

Ex:- ①
```
class student:
    def __init__(self, name, age, marks)
        self.name = name
        self.age = age
        self.marks = marks
    def instance_method(self):
        print(self.marks)
if __name__ == '__main__':
    ref = student('name', 'sachin', age = 45, marks = 78)
    print(ref.marks) → 78
    ref.instance_method()
```

Deletion of instance variable

* within the instance method we can delete the instance variable by using two ways
  → using self keyword
  → using the reference which is pointing to the object.
* outside the class the instance variable can be deleted using the reference pointing to the object.

ex:-
```
class student:
    def __init__(self, name, age, marks, gender):
        self.name = name
        self.age = age
        self.marks = marks
        self.gender = gender
    def display(self):
        del self.marks  # using self keyword
        del ref.age  # using reference which is pointing to the object
if __name__ == '__main__':
    ref = student('name': 'sachin', age: 45, marks: 78,
                  gender = 'male')
    print(ref.__dict__)
    ref.display()
    print(ref.__dict__)

    del ref.name  # deleting the IV outside the class.
    print(ref.__dict__)
```

O/P

```
{'name': 'sachin', 'age': 45, 'marks': 78, 'gender': 'male'}
{'name': 'sachin', 'gender': 'male'}
{'gender': 'male'}
```

STATIC Variables

Static variables are variables whose value will not change from object to object

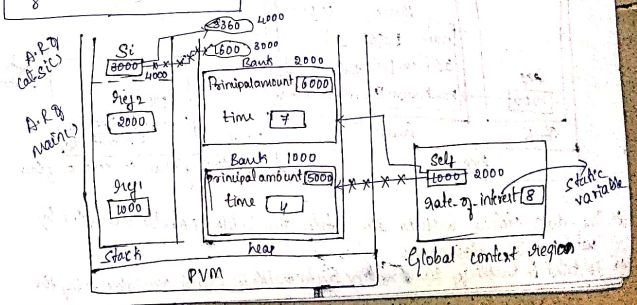
- variables which are declared within the defined class and outside the method is referred as static variable
- static variable can be accessed using i) class name
  ii) reference which is pointing to the object
- the memory for static variable will be allocated with in global context region & a single copy will be present. so that memory will be utilized efficiently.

ex:-1

```
class Bank:
    rate_of_interest = 8 -> static variable
    def __init__(self, principalamount, time):
        self.principalamount = principalamount
        self.time = time
    def cal_si(self):
        si = (principalamount * self.time * Bank.rate_of_interest) / 100
        print('simple interest is :', si)

if __name__ == '__main__':
    ref1 = Bank(principalamount=5000, time=4)
    ref1.cal_si()
    ref2 = Bank(principalamount=6000, time=7)
    ref2.cal_si()

    print(Bank.rate_of_interest)  # accessing static variable using classname
    print(ref1.rate_of_interest)  # accessing sv using I object reference
    print(ref2.rate_of_interest)  # accessing sv using II object reference
```

O/P

```
simple interest is : 1600
simple interest is : 3360
8
8
8
```



Note:- * Always the memory for static variable will be allocated first within any program.

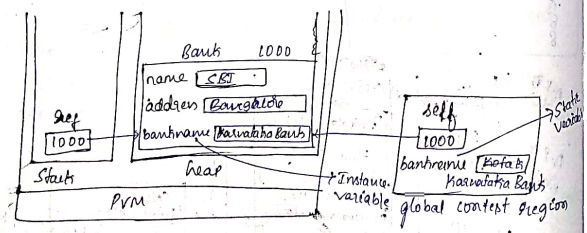
Imp

Modifying the Existing Value present within the static variable

Ex:-
```
class Bank:
    bankname = 'Hotak'
    def __init__(self, name, address):
        self.name = name
        self.address = address
if __name__ == '__main__':
    ref = Bank(name = 'SBI', address = 'Bangalore')
    print(ref.__dict__)
    print(ref.bankname)
    ref.bankname = 'Karnataka Bank' # new IV will be created
    print(ref.bankname)
    print(ref.__dict__)
    print(Bank.bankname)
    Bank.bankname = 'ICICI' # static variable will be modified using class name
    print(Bank.bankname)
```

O/P

```
{'name': 'SBI', 'address': 'Banglore'}
Hotak
Karnataka bank
{'name': 'SBI', 'address': 'Bangalore', 'bankname': 'Karnataka Bank'}
Hotak
ICICI
```



* The data stored within the static variable can be modified by using only class name.
* It tried to modify the existing data within static variable using the object reference then a brand new instance variable will be added to the object
* If the data has to be stored within any variable as a fixed data then we can make use of static variable.

Local variables and Global variables

Local variable:- variables which are declared within a user defined function and not enclosed within a defined class is referred as local variables

* The scope of local variable will be limited within a function, outside the function we cannot access local variable.
* `locals()['variablename']` is used to fetch the data present within local variable.

Global variable:- variables which are declared outside the user defined function is not enclosed within a defined class.

* `globals()['variablename']` is used to fetch the data stored within global variable. to make any variable as global variable we can use global keyword.
* The scope of global variable will be throughout the program.

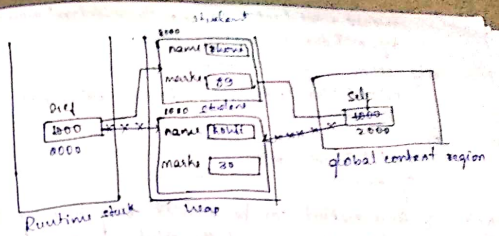
Ex:-
```
x = 77
def func():
    global x
    x = 99
    print(x) -> 99
    y = 45
    print(y) -> 45
    print(locals()['y']) -> 45
    print(globals()['x']) -> 99
if __name__ == '__main__':
    func()
    print(x) -> 99
    print(globals()['x']) -> 99
    print(locals()['y']) -> Error
```

Methods

1) Instance methods :- methods which are declared within a defined class & the 1st parameter happens to be self is referred as instance method.

* instance methods are used to write the business logic
* using the reference which is pointing the object we can call instance methods.

ex:-
```
class student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def display_data(self):
        print('The student name is:', self.name)
        print('The student marks is:', self.marks)

    def display_grade(self):
        if self.marks >= 70:
            print('FCD')
        elif self.marks >= 60:
            print('FC')
        elif self.marks >= 35:
            print('second class')
        else:
            print('fail')

if __name__ == '__main__':
    num = int(input('Enter the numbers of students'))
    for i in range(num):
        n = input('Enter the name of the student')
        m = int(input('Enter the marks of the student'))
        ref = student(name = n, marks = m)
        ref.display_data()
        ref.display_grade()
```



ex2 :-
```
import sys
class Customer:
    def __init__(self, name, balance = 0.0):
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        self.balance = self.balance + amount
        print('The balance after deposit is:', self.balance)

    def withdraw(self, balance):
        if amount > self.amount: balance
            print('insufficient funds!!')
            sys.exit(0)
        else:
            self.balance = self.balance - amount
            print('The balance after withdraw is:
                  {self.balance}')

if __name__ == '__main__':
    n = input('enter the customer name:')
    ref = customer(name = n)
    while True:
        print('select your choice \n D/D-Deposit \n
              W/W - withdraw \n E/E-Exit')
        option = input('Enter your option:')
        if 'd' == option or option == 'D':
            amount = int(input('Enter the deposit
                               amount'))
            ref.deposit(amount)
        elif 'w' == option or 'W' == option
```

```
        amount = int(input('Enter the withdraw amount \t'))
        ref.withdraw(amount)
    elif option == 'e' or option == 'E':
        print('Thanks for using banking application')
        sys.exit(0)
    else:
        print('Invalid option')
        sys.exit(0)
```

Note :- instance method can be accessed only by using the reference which is pointing to the object

* using classname we cannot access the instance methods

O/P

```
Enter the customer name  Bindu
Enter your option choice
d/D - Deposit
w/W - Withdraw
e/E - Exit
Enter your option d
Enter the amount to deposit 5000
The balance after deposit is: 5000
Enter the ch
```

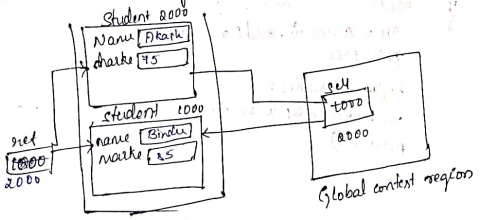
Setters & getters

Setters are used to add the properties to an object using instance methods.

getters are used to fetch the value from properties or an object

Ex:-

```
class Student:
    def setName(self, name):
        self.name = name
    def setMarks(self, marks):
        self.marks = marks
    def getName(self):
        return self.name
    def getMarks(self):
        return self.marks
if __name__ == '__main__':
    n = int(input('Enter the no. of student \t'))
    for i in range(n):
        ref = Student()
        name = input('Enter the name \t')
        ref.setName(name)
        marks = int(input('Enter the marks \t'))
        ref.setMarks(marks)
        print('The name student name is: ', ref.getName())
        print('The student marks is: ', ref.getMarks())
        print(ref.__dict__)
```



O/P

```
Enter the no. of student : 2
Enter the name  Bindu
Enter the marks  85
The student name is :  Bindu
The student marks is :  85
{'name': 'Bindu', 'marks': 85}
```

~~Bindu : 85~~

② Static Method

These are the methods which will be inherited using the class. @staticmethod.

* With in this method implementation we should not use IV, CV or SV.
* With in this method always make use of local variables
* static methods can be accessed using two ways.
  1) classname 2) reference which is pointing to the object

Ex:

```
class Abcmath:
    @staticmethod
    def add(x,y):
        return x+y
    @staticmethod
    def sub(x,y):
        return x-y
if __name__ == '__main__':
    res = Abcmath.add(x=10, y=20)
    print(res)
    ref = Abcmath()
    res = ref.sub(x=20,y=10)
    print(res)
```

O/P

```
30
10
```

③ class method

* class methods are such methods. that are inherited from one class @classmethod
* The first parameter within the class method happens to be cls.
* within the class method implementation we should not use instance variables. Rather we have to make use of class variable or static variable.
* there static variable can be accessed in two ways
  1) using the classname 2) using the parameter cls.

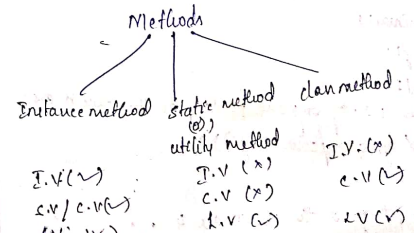
```
class Animal:
    legs = 4
    @classmethod
    def walk(cls, name):
        print(f'The animal {name} walks with {cls.legs} legs')

if __name__ == '__main__':
    ref = Animal()
    ref.walk(name='lion')
    ref = Animal()
    ref.walk('tiger')
```

O/P

```
The animal lion walks with 4 legs
The animal tiger walks with 4 legs.
```

Wap to count the no. of objects present with in the ~~program~~ without using the constructor

```
class Animal:
    no_of_objects = 0
    @classmethod
    def display(cls):
        cls.no_of_objects = cls.no_of_objects + 1
if __name__ == '__main__':
    ref1 = Animal()
    Animal.display()
    ref2 = Animal()
    Animal.display()
    ref3 = Animal()
    Animal.display()
    print(f'{Animal.no_of_objects}')
```



Wapython program to count no. of object present with in the program using constructor.

```
class count:
    no_of_objects = 0
    def __init__(self):
        count.no_of_objects = count.no_of_objects + 1
    @classmethod
    def display(cls):
        print("no of objects present with the prog is", cls.no_of_objects)
if __name__ == '__main__':
    ref1 = count()
    ref2 = count()
    ref3 = count()
    ref1.display()
    ref4 = count()
    ref5 = count()
    ref1.display()
```

O/P

no. of objects present with in the prog is 3
" " " " " " " is 5

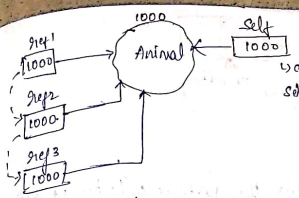
Wap to count the no. of references pointing to the object

Note:- In order to count the reference we can use getrefcount present with in the sys module

```
import sys
class Animal:
    pass
if __name__ == '__main__':
    ref1 = Animal()
    ref2 = ref1
    ref3 = ref2
    print("the no. of reference pointing to the objects are:",
          sys.getrefcount(ref1))
```

O/P

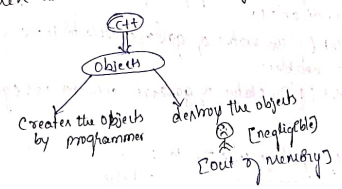
The no. of reference pointing to the objects are 4 ...



→ default constructor is present
self is a keyword which holds the reference of current object

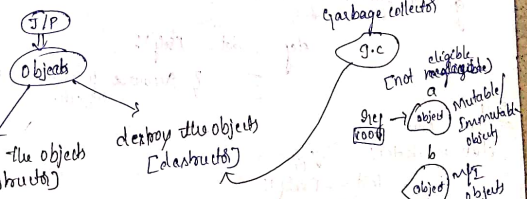
garbage collector :-

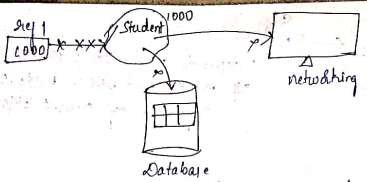
In case of programming language like c++ programmer can create the objects, as well as he like to destroy that objects. → beg of the negligence of a programmer if the objects are not destroyed. then it will lead to memory.



In case of programming language like Java/Python the programmer can creates the objects then inorder to destroy the objects it will make use of garbage collector.

- if the objects are not having any reference then it will be eligible to get destroyed by garbage collector.
- constructor is used to initialize the objects
- destructor is used to terminate the activities of the objects, with other entities.





Program to check whether the garbage collector is active or not

```
import gc
if __name__ == '__main__':
    print ("The status of garbage collector is :", gc.isenabled())
    gc.disable()
    print ("The status of garbage collector is :", gc.isenabled())
    gc.enable()
    print ("The status of garbage collector is :", gc.isenabled())
```

O/P

The status of garbage collector is : True
" " " : False
" " " : True

destructor.

It's a magical method used to write the resource termination code of the objects.

* The name of this happens to be __del__.
* once the object is not having any reference pointing to it, then garbage collector will destroy that objects but before destroying the objects this destructor method will be invoked

general syntax of destructor

```
def __del__(self):
    ==  } resource termination code to objects.
```

Ex:1

```
import time
class cricketer:
    def __init__(self):
        print ('constructor got executed')
        print ('initializing the objects')
    def __del__(self):
        print ('destructor got executed')
        print ('fullfilling the last wish')
        print ('cleaning up the activities of the objects')
if __name__ == '__main__':
    print ('stating the application')
    ref = cricketer()
    time.sleep(5)
    ref = None
    time.sleep(5)
    print ('application got closed')
```

O/P

```
Starting the application
constructor got executed
initializing the objects
after 5sec  destructor got executed
            fullfilling the last wish
            cleaning up the activities of the objects
after 5sec  application got closed
```

Ex2:-

If multiple references are pointing to the objects, then destructor got executed if no references are pointing to the object.

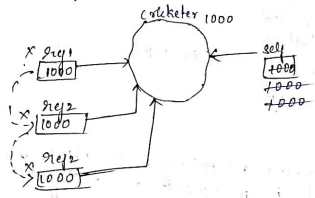
```
import time
class cricketer:
    def __init__(self):
        print ('constructor got executed')
    def __del__(self):
        print ('destructor got executed')
if __name__ == '__main__':
    print ('starting the application')
    ref1 = cricketer()
    ref2 = ref1
    ref3 = ref2
    del ref1
    print ('ref1 reference got deleted')
    time.sleep(3)
    del ref2
    print ('ref2 reference got deleted')
```

```
time.sleep(3)
print('ref3 reference got deleted')
def ref03
time.sleep(3)
print('application got closed')
```

O/P

starting the application
ref constructor got executed
ref1 reference got deleted
ref2 — 11 —
ref3 — 11 —
destructor got executed
application got closed

In python everything like int, function etc. are objects

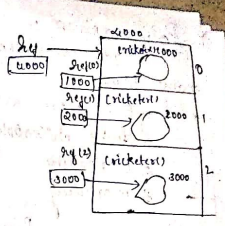


Note:- * within an object we can create multiple objects.
* if the reference of the objects which holds the reference of other objects is deleted then all those reference pointing to the respective objects will get deleted.

```
import time
class Cricketer:
    def __init__(self):
        print('constructor executed')
    def __del__(self):
        print('destructor got executed')
if __name__ == '__main__':
    print('starting the application')
    ref = [Cricketer(), Cricketer(), Cricketer()]
    time.sleep(3)
    del ref
    time.sleep(3)
```

print('application got closed')

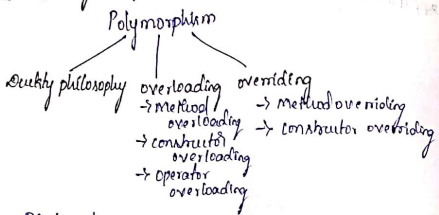
starting the application
constructor executed
— 11 —
— 11 —
destructor got executed
— 11 —
— 11 —
application got closed.



Polymorphism :-

Its mechanism of exhibiting a single entity in many forms.

* In python there are 3 levels of polymorphism
  1) Duckty philosophy
  2) overloading
  3) overriding



Ducktype Philosophy :-

Python is a dynamically typed programming lang. bcz the type of the object will be decided at the execution place bcz. it follows the principle of ducking

* Since, python follows ducking principle we can achieve the Polymorphism using Duck type philosophy.

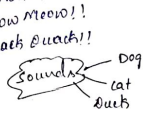
Definition :- Duck type philosophy is a process of exhibiting a single instance method name in many forms by providing by the method implementation as per the programmer needs.

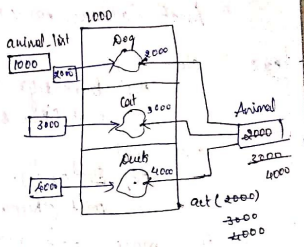
Ex :- 1.

```
class Dog:
    def sounds(self):  # Instance method
        print("Bow Bow!!!")
class cat:
    def sounds(self):  # IM
        print("Meow Meow!!")
class Duck:
    def sounds(self):  # IM
        print("Quack Quack")
if __name__ == '__main__':
    def act(parameter):
        parameter.sounds()
    animal_list = [Dog(), cat(), Duck()]
    for animal in animal_list:
        act(animal)
```

O/P

```
Bow Bow!!
Meow Meow!!
Quack Quack!!
```



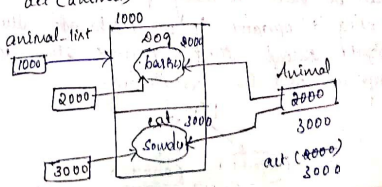


Ex 2 :-

```
class Dog:
    def bark(self):
        print("Bow, Bow!!")
class cat:
    def sound(self):
        print("Meow, Meow!!")
if __name__ == '__main__':
    def act(parameter):
        # if hasattr(parameter, 'bark'):
            parameter.bark()
        # if hasattr(parameter, 'sound'):
            parameter.sound()  # Attribute error
    Animal_list = [Dog(), cat()]
    for animal in animal_list:
        act(animal)
```

O/P

```
Bow Bow!!
Attribute error
```



In Duck type philosophy if the method is used with the different values, then it will lead to attribute error.

* Inorder to overcome this we will make use of hasattr() It will take two argument as shown below.

hasattr (address, name of the method)

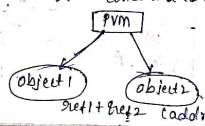
Ex:-
```
class dog:
    def bark(self):
        print ("Bow Bow!!")
class cat:
    def sound(self):
        print ("Meow Meow");
if __name__ == '__main__':
    def act (parameter):
        if hasattr(parameter, 'bark'):
            parameter.bark()
        if hasattr(parameter, 'sound'):
            parameter.sound();
    animal_list = [dog(), cat()]
    for animal in animal_list:
        act(animal)
```

O/P
Bow Bow!!
Meow Meow!!

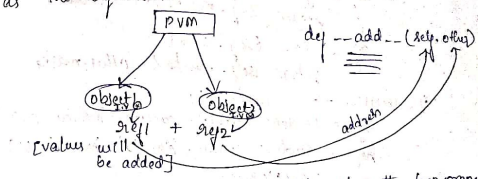
Overloading

① operator overloading :- It is a mechanism where one operator can be used in many forms

* using '+' operator if we try to add the properties of an object through their references. then instead of addition the reference will be concatinated.



* inorder to overcome this we will make use of magical methods. i.e., inorder to perform addition b/w the properties of an object we will make use of __add__() which takes the addresses of an object as the argument.

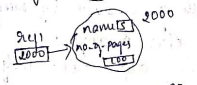


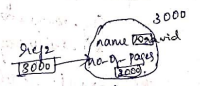
* a python program to perform the addition b/w the two properties of an objects using operator overloading. (+)

```
class student:
    def __init__(self, name, no_of_pages):
        self.name = name
        self.no_of_pages = no_of_pages
    def __add__(self, other):
        return self.no_of_pages + other.no_of_pages
if __name__ == '__main__':
    ref1 = student ('name': 'sachin', 'no_of_pages' = 100)
    ref2 = student ('name' = 'dravid', 'no_of_pages' = 200)
    print ('The total no of pages :', (ref1+ref2))
```

→ we can give as many arguments we need

O/P
The total no of pages : 300





Napython program to perform operator overloading upon relational operators (>, <, >=, <=)

```
class student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks
    def __gt__(self, other):
        return self.marks > other.marks
```

```
def __lt__(self, other)
    return self.marks < other.marks
def __gt__(self, other)
    return self.marks > other.marks
def __le__(self, other)
    return self.marks <= other.marks
if __name__ == '__main__':
    ref1 = student('name': sachin, marks: 80)
    ref2 = student('name': david', marks: 60)
    print (ref1 > ref2)
    print (ref1 < ref2)
    print (ref1 >= ref2)
    print (ref1 <= ref2)
    print (56 < 82)
    print (56 < 82)
```

O/P
True
False
True
False
False
True

wap to calculate the salary of the employee based upon the given data by using operator overloading (or)

```
class Employe:
    def __init__(self, name, dailywages):
        self.name = name
        self.dailywages = dailywages
    def __mul__(self, other):
        return self.dailywages * other.no_of_days
class Timesheet:
    def __init__(self, time, no_of_days):
        self.time = time
        self.no_of_day = no_of_days
if __name__ == '__main__':
    num = int(input('Enter the name of the Employee'))
    for i in range(num):
        name = input('Enter the name of the employee')
        dailywages = int(input('Enter the dailywages of the employee'))
        ref1 = Employee(name = name, dailywages = dailywages)
        ref2 = Timesheet(time = 10, no_of_days = 30)
        print('The salary of the employee is:', (ref1, ref2))
```

O/P
Enter the name of the Employe: sachin
Enter the dailywages of the Employe: 5000
The salary of the Employee is: 1,50,000
Enter the name of the Employe: chaithra
Enter the dailywages of the employee: 8000
The salary of the Employee is: 2,40,000

wap passing the properties present within one class to another class

wap to calculate the salary after providing hike using the below mentioned details.

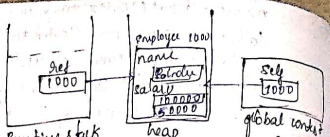
Employee

Name	Salary
Bindu	100000

Hike
50000

```
class Employe:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def display_data(self):
        print('Name:', self.name)
        print('Salary:', self.salary)
class Hike:
    def give_hike(self, ref):
        ref.salary = ref.salary + 50000
if __name__ == '__main__':
    ref = Employe(name = 'Bindu', salary = 100000)
    ref.display_data()
    Hr.give_hike(ref)
    ref.display_data()
```

O/P
Name - Bindu
Salary = 100000
Name - Chandu
Salary = 150000



Prog to calculate the hike of respective employee based upon data given below.

Employee

name	salary
Bindu	1500000
Vandu	2000000
Chaithra	8000000

HIKE

Range	Hike
1000000 to 1500000	10,000
1500000 to 1800000	20,000
> 2000000	5,00,000

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def display_data(self):
        print('name:', self.name)
        print('salary:', self.salary)
    def display_new_data(self):
        print('The Employee', self.name, 'salary after hike:',
              self.salary)

class hr:
    def give_hike(self, hike):
        self.salary = self.salary + hike
if __name__ == '__main__':
    num = int(input('Enter the no. of employee'))
    for i in range(num):
        name = input('enter the name of employee')
        salary = int(input('Enter the salary of the employee'))
        hike = int(input('Enter the hike amount'))
        ref = Employee(name, salary)
        ref.display_data()
        ref.give_hike(hike)
```

ref.display_new_data()

O/P
Enter the no of Employee 3
Enter the name of Employee Bindu
Enter the salary of Employee 15000000
Enter the hike amount 1000000
The salary of the Employee in 1600000

Vandu	10000000
"	20000000
"	10000000
Chaithra	20000000
"	5000000
"	25000000

Different magic methods used to perform operator overloading

Operator	magic method
+	__add__(self, other)
-	__sub__(self, other)
*	__mul__(self, other)
/	__div__(self, other)
//	__floordiv__(" ,")
%	__mod__(" ")
<	__lt__(" ")
>	__gt__(" ")
>=	__ge__(" ")
<=	__le__(" ")
==	__eq__(" ")
+=	__iadd__(" ")
!=	__ne__(" ")

Sir

Method overloading

In python the mechanism of method overloading is not possible.

* Method overloading is a process of creating a multiple methods with the same name but with different no. of parameters.

* In case of python if we try to perform m.o. then the method which is present at the last in the method hierarchy will gets executed as shown below.

Ex:-

```
class calculate:
    def add(self,a):
        return a+10
    def add(self,a,b):
        return a+b
    def add(self,a,b,c):
        return a+b+c
if __name__=='__main__':
    ref=calculate()
    sum=ref.add(a=10)
    sum=ref.add(a=10,b=20,c=30)
    print('The result is:',sum)
```

O/P

The result is: 60

In case of python method overloading can be handled indirectly using the following approach.

```
class calculate:
    def add(self,a=None,b=None,c=None)
        if a!=None and b!=None and c!=None:
            return a+b+c
        elif a!=None and b!=None:
            return a+b
        elif a!=None:
            return a+10
        else:
            print('Enter atleast one argument')
if __name__=='__main__':
    print('The result is:',ref.add(a=10))
    print('The result is:',ref.add(a=10,b=20))
    print('The result is:',ref.add(a=10,b=20,c=30))
    print('The result is:',ref.add())
```

O/P

The result is 20 (D3)
The result is 30
The result is: 60

Constructor overloading:-

In python there is no mechanism of constructor overloading.

-> If we try to perform constructor overloading then the last constructor present within constructor hierarchy will executed as shown below.

Ex:-

```
class student:
    def __init__(self,name):
        self.name=name
    def __init__(self,name,marks):
        self.name=name
        self.marks=marks
    def __init__(self,name,marks,age):
        self.name=name
        self.marks=marks
        self.age=age
    def display_data(self):
        print('The name:',self.name)
        print('The marks:',self.marks)
        print('The age:',self.age)
if __name__=='__main__':
    ref=student(name='sachin')->missing 2 required argument error
    ref=student(name='sachin',marks=40,age=36)
    ref.display_data()
```

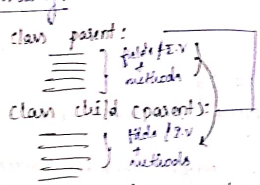
this constructor get will execute.

O/P
Name : Sachin
marks: 40
age : 5+

In case of python constructor overloading can be handled using the following approach.

* Refer to the constructor overloading program in to write here.

Method overriding:



The process of inheriting properties & methods present within parent class into child class changing the implementation of that method as per the child class is expressed as method overriding.

* In case of python we can achieve method overriding as shown below.

```
class parent :
    def property (self):
        print ('cash + land + gold')
    def marry (self):
        print ('anupama')
class child (Parent):
    def marry (self):
        print ('sunny leone')
if __name__ == '__main__':
    ry = child()
    ry.property()
    ry.marry()
```

O/P
cash + gold + land
by sunny leone

Constructor overriding:

Ex:1

```
class Parent:
    def __init__(self):
        print ("parent class constructor")
class child (parent):
    def __init__(self):
        print ("child class constructed")
if __name__ == '__main__':
    ry = child()
```

O/P
child class constructor

Ex: 2

```
class Parent:
    def __init__(self):
        print ("parent class constructed")
class child(parent):
    pass
if __name__ == '__main__':
    ry=child()
```

→ default constructor created by pvm

O/P
parent class constructor

* In Ex1 within the child class there is a defined constructor so if a child object is created then the defined constructor will get executed.

* In Ex2 there is no define constructor within the child class so if the child object is created then the constructor present within parent class get executed. So within the child class pvm will create a default constructor within that call will be made to the parent class constructor.

Ex3:

```
class parent:
    def __init__(self):
        print ("parent class constructor")
class child (Parent):
```

```
    def __init__():
        super().__init__()
        print("The child class constructor")
if __name__ == '__main__':
    ref = child()
```

O/P

Parent class constructor
child class constructor

Ex:-4

```
class Parent:
    def __init__(self):
        print("parent class constructor")
class child(Parent):
    def __init__(self):
        print("child class constructor")
        super().__init__()
if __name__ == '__main__':
    ref = child()
```

O/P

child class constructor
parent class constructor

* we can call the parent class constructor within child class constructor using super()
* using super() we can call parent class constructor in any line of program within child class constructor

Initializing the properties of parent class without creating Parent object.

* In the below example parent class (person) it having two properties and the child class have inherited parent class, this child class is having two properties.
* while creating the child object we can provide the values for all the properties but inorder to initialize two properties (name, age) we should call parent class constructor with in child class constructor using super()
* while calling that parent class constructor we should pass two arguments.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
class Student(Person):
    def __init__(self, name, age, marks, backlog):
        super().__init__(name=name, age=age)
        self.marks = marks
        self.backlog = backlog
    def display_data(self):
        print(f'Name: {self.name}')
        print(f'Age: {self.age}')
        print(f'marks: {self.marks}')
        print(f'backlog: {self.backlog}')
if __name__ == '__main__':
    ref = student(name='kushal', age=28, marks=35
                  backlog=6)
    ref.display_data()
```

O/P

```
Name: kushal
age: 28
marks: 35
backlog: 6
```

Inner class

Its a mechanism of creating one class within another class. there is a dependency within is there within a object then we should make use of inner class.

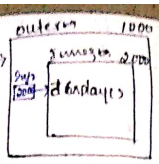
class Book:
    class page:

class human:
    class brain:

class university:
    class college:

Eg:- class Outer:

class Inner:



class Outer:
    def __init__(self):
        print("outer class constructor")

    class Inner:
        def __init__(self):
            print("Inner class constructor")

        def display(self):
            print("Inner class method")

if __name__ == '__main__':
    ref1 = Outer()
    ref2 = ref1.Inner()
    ref2.display()
    print()
    ref3 = Outer().Inner()
    ref3.display()
    print()
    Outer().Inner().display()

→ I way of accessing

→ II way of accessing

→ IV way of accessing

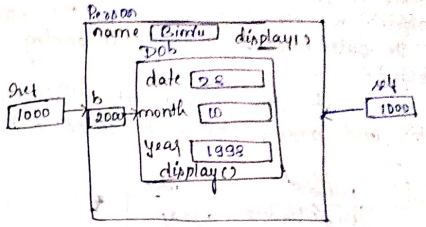
O/P

outer class constructor
Inner " "
Inner " method

outer class constructor
Inner " "
Inner " method

Outer class constructor
Inner " ...
" " method

Eg:1 : Create the date of birth object within the person object & the person name happens to be Bindu and the dob happens to be 28/10/1998



class Person:
    def __init__(self, name, date, month, year):
        self.name = name
        self.b = self.DOB(date, month, year)

→ In order to create the object of DOB

    def display(self):
        print(f'Name: {self.name}')
        self.b.display()

class DOB:
    def __init__(self, date, moth, year):
        self.date = date
        self.month = month
        self.year = year
    def display(self):
        print(f'Date of birth: {self.date}/{self.month}/{self.year}')

if __name__ == '__main__':
    ref = person('name', 'Bindu', date=28, month=10, year=1998)
    ref.display()

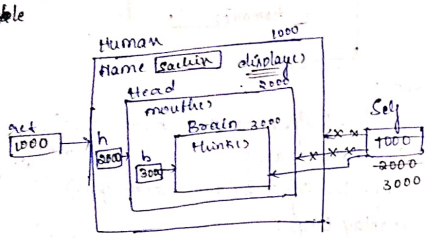
O/P

Name: Bindu
Date of Birth: 28/10/1998

* In order to create the object of innerclass we should make use of the reference variable pointing to the outer object (There are two ways: self, reference variable)

* In order to call the instance method present within inner object we should make use of the reference pointing to the outer object, the reference pointing to the inner object.

* We can create the inner object within outer object constructor and address has to be stored with in instance variable




```
class Human:
    def __init__(self, name):
        self.name = name
        self.h = self.head()
    def display(self):
        print(f'Name: {self.name}')
        self.h.mouth()
        self.h.b.think()

class Head:
    def __init__(self):
        self.b = self.Brain()
    def mouth(self):
        print('using mouth we can eat')

class Brain:
    def think(self):
        print('using brain we can think')

if __name__ == '__main__':
    ref = Human('sachin')
    ref.display()
```

O/P
Name : sachin
using mouth we can eat
using Brain we can think.

Inner function:-

Creating a function another function is referred as inner functions. the innerfunction, have to be called with in outer function itself.

Aliasing a function:-

In order to alias a function we just assign the function name or stored the function name within one more variable.

Ex:-

```
def outer():
    print('outer function')
    def inner():
        print('inner function')
    i = inner # aliasing the function
    i()
if __name__ == '__main__':
    o.outer()
    o()
# inner() -> error
```

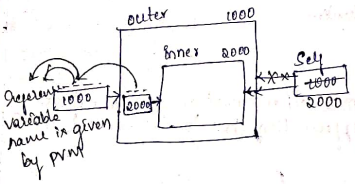
we can create an object with out creating a class

O/P
outer function
Inner function

* A function can return o/p in many ways as shown below using the keyword return

function

- return a single o/p
- return multiple o/p
- returns functions as o/p [lambda function]
- return sequence as o/p [list, tuple...]



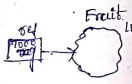
Abstraction

Abstract method :-

a method without any implementation and which has the first parameter has self. It should have inherited the decorator @ abstract method, it should belongs to module called as abc then it is referred as abstract method

Ex:-1

```
from abc import *
class Fruit:
    @abstractmethod
    def taste(self):
        pass
if __name__ == '__main__':
    ref = Fruit()
    print(id(ref))
    ref.taste()
```



→ 1000

* In the above example the class Fruit is not an abstract class.

Ex 2:- within a class when is any abstract method then we can create the instance method also.

```
from abc import *
class Fruit:
    @abstractmethod
    def taste(self):
        pass
    def season(self):
        print('season fruits')
if __name__ == '__main__':
    ref = Fruit()
    print(id(ref))
    ref.taste()
    ref.season()
    # Fruit.season() -> we cannot call instance method using class name (Error)
    # Fruit.taste() -> Error
```

O/P

season fruits

* using the classname we cannot access instance method & abstract method

Abstract class

If a class consists of atleast one abstract method it should have inherited from the base abstract class called as 'ABC' and present within the module abc in required as abstract class.

Ex:- 1

```
from abc import *
class Fruit(ABC):
    pass
if __name__ == '__main__':
    ref = Fruit()
    print(id(ref))
```

O/P

1000

* In the above example the class fruit is not an abstract class bcz. there is no abstract method present in it. so we can create object of that class.

Ex2:- For an abstract class we cannot create the object if we try to object it will leave the exception called TypeError as shown below

```
from abc import *
class Fruit(ABC)
    @abstractmethod
    def taste(self):
        pass

if __name__ == '__main__':
    ref = Fruit() #-> TypeError
    print(id(ref)) -> want to check the whether object is created or not
```

O/P

TypeError: can't instantiate the abstract class

Ex3:- * if a class inherits the abstract class in order to create the object of that class we should provide the implementation for all the abstract method present with in abstract class.

* after providing the body for the abstract method with in a child class we can only create the object for that child class not for parent class.

```
from abc import *
class Fruit(ABC):
    @abstractmethod
    def taste(self):
        pass

    def season(self):
        print("tasty fruit");

class Mango(Fruit):
    def taste(self):
        print("summer season fruit")

if __name__ == '__main__':
    ref = Mango()
    print(id(ref))
    ref.taste()
    ref.season()
# ref = Fruit() #->Error
```

O/P

1000
summer season fruit
tasty fruits

Ex4:-

```
from abc import *
class Animal(ABC):
    @abstractmethod
    def eat(self):
        pass
    @abstractmethod
    def sleep(self):
        pass

    def breathe(self):
        print('Animals breathe')
class Tiger(Animal):
    def eat(self):
        print("Tiger eats")

    def sleep(self):
        print('Tiger sleeps')*
if __name__ == '__main__':
    ref = Tiger()
    ref.eat()
    ref.sleep()
    ref.breathe()
```

O/P

Tiger eats
Tiger sleeps
Animals breathe

ex5:- * we can convert the instance method into abstract method has shown below.

```
from abc import *
class Animal(ABC)
```

```
    @abstractmethod
    def eat(self):
        pass
    def sleep(self):
        print("Animals sleeping")
    def breathe(self):
        print("Animals breathe")

class Lion(Animal):
    def eat(self):
        print("Lion hunts & eats")
    @abstractmethod
    def sleep(self):
        pass

class Malelion(Lion):
    def sleep(self):
        print("Malelion sleeps till 8hrs")

if __name__ == '__main__':
    ref = Malelion()
    print(id(ref))
    ref.sleep()
    ref.eat()
    ref.breathe()
```

o/p

```
1500
Malelion sleeps till 8hrs
Lion hunts & eats
Animals breathe
```

Ex:-6

```
from abc import *
class Vehicles(ABC):
    @abstractmethod
    def wheels(self):
        pass
    def drive(self):
        print("to drive GL in compulsory")
class Bus(Vehicles):
    def wheels(self):
        return "bus consists of 6 wheels"
class Car(Vehicles):
    def wheels(self):
        return "car consists of 4 wheels"
if __name__ == '__main__':
    ref1 = Bus()
    ref1.drive()
    res = ref1.wheels()
    print(res)

    ref2 = Car()
    ref2.drive()
    print(ref2.wheels())
```

o/p

```
to drive GL in compulsory
bus consists of 6 wheels
to drive GL in compulsory
car consists of 4 wheels
```

Note:- Abstraction is used to achieve code reusability

Interface

If a class consists of only abstract method then such class is prepared as interface

Ex: from abc import *
class DBInterface(ABC):
@abstractmethod
def db_connect(self):
pass
@abstractmethod
def db_close(self):
pass
→ interface

class Oracle(DBInterface):
def db_connect(self):
print("connected to oracle database")
def db_close(self):
print("disconnected from oracle database")

class Mysql(DBInterface):
def db_connect(self):
print("connected to mysql database")
def db_close(self):
print("disconnected from mysql database")

if __name__ == '__main__':
print("select the given database as oracle/mysql")
databasename = input("select the database name")
# print(globals())
classname = globals()[databasename] → globals() is used to convert string into variable
ref = classname()
ref.db_connect()
ref.db_close()

O/P
Select the given database.
oracle
mysql
select the database name
oracle
connect to oracle database
disconnected from oracle database

Note:- * In the above example the collected input from the user will be stored in the form of string. so we cannot create the object.
* Inorder to overcome this will make use of 'globals()' to convert the string data into a variable. So that we can create object for the variable.

Ex:- a python program to fetch the data present with in textual file(.txt) using the concept of interface

from abc import *
class Printer(ABC):
@abstractmethod
def start(self):
pass
@abstractmethod
def stop(self):
pass

class Hpprinter(printer):
def start(self,text):
print("connected to Hpprinter")
print(text)
def stop(self)
print("disconnected from Hpprinter")

class Elipsonprinter(printer):
def start(self,text)
print("connected to Elipsonprinter")
Print(text)
def def stop(self):
print("disconnected from Elipson Printer")

if __name__ == '__main__':
with open(file='printer.txt', mode='r') as f:
printername = f.readline()
classname = globals()[printername]
ref = classname()

```
ref.start(text: 'printing python program')
ref.stop()
```

O/P

connect to tlpprinter
printing python program
disconnected from tlpprinter.

temporary → non persistent
permanent → persistent

data is stored on the HDD in the form of Bytes
∴ in order to execute after some long time in RAM also stored in the form of Byte.

Pickling and UnPickling

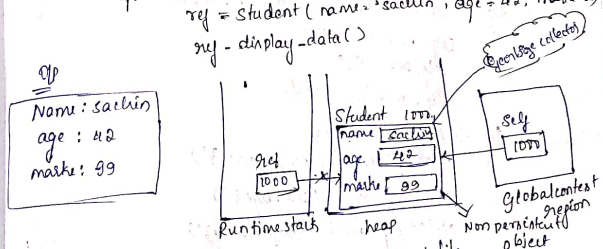
Non persistent Object :-

Ex:-1

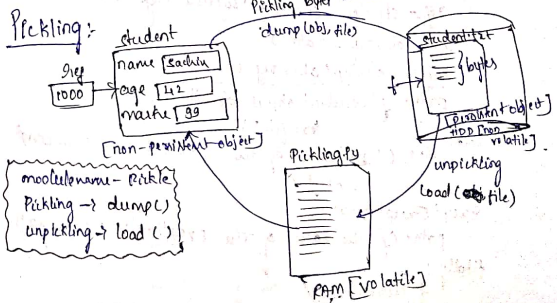
```
class Student:
    def __init__(self, name, age, marks):
        self.name = name
        self.age = age
        self.marks = marks
    def display_data(self):
        print(f'Name: {self.name}')
        print(f'age : {self.age}')
        print(f'marks: {self.marks}')
if __name__ == '__main__':
    ref = Student(name='sachin', age=42, marks=99)
    ref.display_data()
```

O/P

```
Name: sachin
age : 42
marks: 99
```



Pickling :-



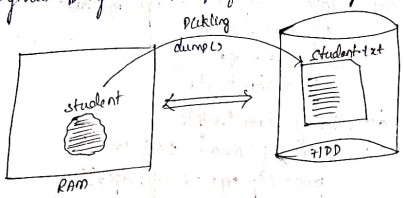

```
module/name - Pickle
Pickling → dump()
unpickling → load()
```

* pickling is a process of writing the state of object into a file.

(or)

converting non-persistent object into persistent object.

→ a python program to perform pickling operation.

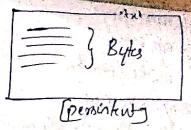



```
import pickle
class Student:
    def __init__(self, name, age, marks):
        self.name = name
        self.age = age
        self.marks = marks

    def display_data(self):
        print(f'name : {self.name}')
        print(f' age : {self.age}')
        print(f' marks : {self.marks}')
if __name__ == '__main__':
    try:
        f = open(file = 'student.txt', mode = 'wb')
        ref = Student(name = 'sachin', age = 42, marks = 99)
        ref.display_data()
        pickle.dump(obj = ref, file = f)
        print('student object details is stored in the
                text file')
    except:
        print('pickling of the student object is comp
            -leted')
    Except Exception as e:
        print('the cause of the exception is :', e.args)
    finally:
        f.close()
```

Op

Name : Sachin
age : 42
marks : 99
student object details is stored in the text file
Pickling of the student object is completed

[non-persistent]



[persistent]

Note:- while performing pickling operation the states of object have to be stored within textual file(.txt) in the form of bytes. bcz,

→ while retrieving the data if we want to display it on RAM it should be in the form of bytes. bcz with in the RAM, the data will be stored in the form of bytes.

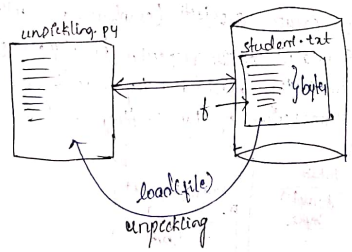
→ in order to provide security for the properties of an object.

Unpickling

It is the process of reading the states of object from the textual file within the python program (RAM).

In order to perform unpickling we make use of load() present with in pickle module.

→ In order to perform any operation upon a file we also -uld make use of file pointer pointing to the file.



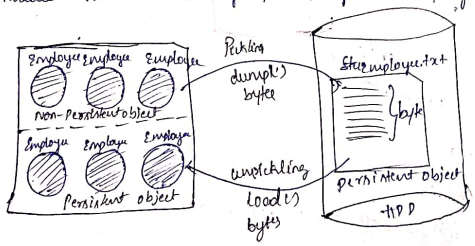

```
- import pickle
  class student:
      def __init__(self, name, age, marks):
```

```
        self.name = name
        self.age = age
        self.marks = marks
    def display_data(self):
        print(f'Name: {self.name}')
        print(f'Age: {self.age}')
        print(f'marks: {self.marks}')
if __name__ == '__main__':
    with open(file='student.txt', mode='rb') as f:
        sau = pickle.load(file=f)
        print('states of object after unpickling')
        sau.display_data()
        print('unpickling completed')
    except Exception as e:
        print(f'the cause of the exception is: {e.args}')
```

O/P

States of object after unpickling
Name: Sachin
Age : 42
Marks: 99
unpickling completed

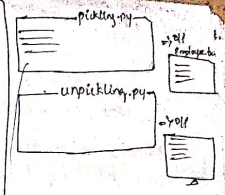
* Wapython program to create 3 objects (employee) and collect the name, address & salary of the respective employees & perform pickling & unpickling operation. by importing the module which has 3 properties of the employees.



Employeeinfo.py

```
class Employee:
    def __init__(self, name, addr, salary):
        self.name = name
        self.addr = addr
        self.salary = salary
    def display_data(self):
        print(f'Name: {self.name}')
        print(f'Address: {self.addr}')
        print(f'salary: {self.salary}')
```

file(fi) module



Pickling program

```
import pickle, Employeeinfo
~~class Employee:~~
if __name__ == '__main__':
    try:
        myfile = open(file='employee.txt', mode='wb')
        num = int(input('enter the no of employee:'))
        for i in range(num):
            name = input('Enter the name of the employee:')
            addr = input('Enter the address ---- :')
            salary = input('Enter the salary ---- :')
            ref = Employeeinfo.Employee(name=name, addr=addr,
                                        salary=salary)
            ~~pickle.dump(file obj='Employee.txt', mode=~~
            pickle.dump(obj=ref, file=myfile)
    except Exception as e:
        print(f'The cause of the exception is: {e.args}')
    else:
        print('pickling completed')
    finally:
        myfile.close()
```

we should create ~~creating~~ the object during module name

O/P

it is in the .txt file

```
enter the no. of employee 2
enter the name of employee Sachin
enter the address of the employee Bangalore
enter the salary of the employee
100000
```

unpickling program

```
import pickle, employeeinfo
if __name__ == '__main__':
    try:
        with open (file='employee.txt', mode='rb') as f:
            sau = pickle.load (file=f)
    except Exception as e:
        print ('The cause of the exception is:', e.args)
    else:
        while True:
            try:
                print ('Employee details after unpickling is')
                sau.display_data()
                print()
            except EOFError:
                print ('The unpickling is completed')
                break
```

O/P

```
Employee details after unpickling
Name : sachin
addr : Bangalore
salary : 100000
The unpickling is completed
```

Note:- In the above example while reading the data from .txt file if the controller encounter the last line then exception has to be collected within except block & the name of exception happens to be EOFError (end of file)
in order to make use of class present within a module you should make use of module name in order to use instance method you need not to use module name but it should be imported

Modifiers in python :-
(Access specifier)

There are 3 modifiers in python as shown below:
1) public → ∞ declare a variable
2) private → __ declare a variable
3) protected → _ declare a variable

Note:- There is no such modifiers like default modifiers in python

```
import pickle, employeeinfo
if __name__ == '__main__':
    try:
        my_file = open ('employee.txt', 'rb')
    except Exception as e:
        print ('The cause of the exception is:', e.args)
    else:
        print ('employee details are:\n')
        while True:
            try:
                ref = pickle.load (my_file)
                ref = display_data()
                print()
            except EOFError:
                print ('The unpickling is completed')
                break
    finally:
        my_files.close()
```

O/P

```
Employee details are
Name: sachin
addr: Bangalore
salary: 100000
Name: Bindu
addr: Bangalore
Salary: 200000
The unpickling is completed
```

Ex 1:-

```
class Abc:
    x = 10      # public
    _y = 20     # protected
    __z = 30    # private
    def display(self):
        print(Abc.x)  -> 10
        print(Abc._y) -> 20
        #print(Abc.__z) -> Attribute error (when we try to access the private variable outside the class)
if __name__ == '__main__':
    ref = Abc()
    print(Abc.x) -> 10
    print(Abc._y) -> 20
    print(Abc.__z)
    ref.display() ------> 10
                          20
                          30
    print(ref._Abc__z) -> 30
    ref._Abc__z = 60
    print(ref._Abc__z) -> 60
```

Note:- * using the classname we can access the private variable only within the class

* inorder to access the private variable outside the class we should make use of syntax shown below.

Reference variable._classname__privatevariablename

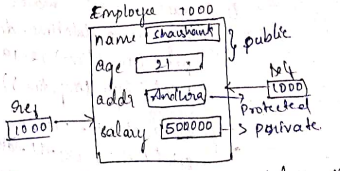
* If a class inherited from one more class then we cannot access the private variable using that classname instead we should make use of the reference which is pointing to that variable.

Ex:-2

```
class Abc:
    def __init__(self, name, age, addr):
        self.name = name
        self.age = age
        self.addr = addr

class Employee(Abc):
    def __init__(self, name, age, addr, salary):
        super().__init__(name, age, addr)
        self.__salary = salary
    def display_data(self):
        print(f'Name : {self.name}')
        print(f'age : {self.age}')
        print(f'addr : {self.addr}')
        print(f'salary : {self.__salary}')
if __name__ == '__main__':
    ref = Employee(name='shashank', age=21, addr='andhra',
                   salary=500000)
    ref.display_data()
    print(ref._Employee__salary)
    ref._Employee__salary = 600000
    print(ref._Employee__salary)
```

O/P

```
Name = Shashank
age : 21
addr : andhra
Salary : 500000
500000
600000
```



__str__ :- whenever the brand new object got created in python magic method called constructor will get executed in order to print the data properties of object we can use the magic __str__ &

__repr__

Ex:-

```
class Employee:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __str__(self):
        return f'The name of the employee is:{self.name}
                and the age is:{self.age}'
if __name__ == '__main__':
    ref1 = Employee(name='shashank', age=21)
    ref2 = Employee(name='sachin', age=22)
    print(f'{ref1}')
```

pr print (f'The details of 2nd employee is{a}{z}')

~~The details of~~

O/P

The name of the Employee : shashank and the age is : 21

The details of 2nd employee :-

The name of the Employee : sachin and the age is : 01

str() and repr() :- * both the functions are used to convert the given object into string object.

* we can fetch server time using now() present within the class datetime which is there inside module called as datetime

repr() :-

```
import datetime ;
~~class datetime:~~
if __name__ == '__main__':
    my_time = datetime.datetime.now()
    print (my_time) -> 2019-12-11 04:49:41.375093
    print (type (my_time)) <class 'datetime.datetime'>

    string_object = repr(my_time)
    print (string_object) datetime.datetime(2019,12,11,4,49,41,375093)
    print (type (string_object)) -> <class 'str'>
    # print (globals()) {'string_object': 'datetime.datetime(2019,12,11,4,49,41,375093)'}

    date_object = eval (string_object) #20
    print (date_object) -> 2019-12-11 04:49:41.375093
    print (type (date_object)) -> <class 'datetime.datetime'>
```

str() :-

```
import datetime
if __name__ == '__main__':
    my_time = date.time.datetime.now()
    print (my_time) -> 2019-12-11 04:57:17.222
    print (type (my_time)) -> <class 'datetime.datetime'>

    string_object = str(my_time)
    print (string_object) -> 2019-12-11 04:57:17.222956
    print (type (string_object)) <class 'str'>
    # print (globals()) {'string_object': 2019-12-11 04:48:08}
```

```
# date_object = eval (string_object) -> error
# print (date_object)
# print (type (date_object))
```

Note :-

- date object —(eval())→ string object —(str())→ string object
- date object —(eval())→ string object —(repr())→ string object

Difference b/w str() & repr()

str()	repr()
* is used to convert any given object into string object.	* is used to conver any given object into string object.
* if the objects are converted into string object using str() then again if we try to convert that string object into original form of object using eval() then it leads to error.	* if the objects are converted into string object using repr() then again if we try to convert that string object into original form of object using eval() then it will be converted.

Public, Protected, Private variables (without using inheritance)

- Within the class
  - → using class name (we can)
  - → using self (we can)
- outside the class
  - → using class name (Private cannot be accessed)
  - → using reference variable (private cannot be accessed)

Public, protected, Private variables (access using inheritance)

- within the class
  - using self (we can)
  - using classname (P,P,P) ×
- outside the class
  - → reference variable (private cannot be accessed)
  - → classname (P,P,P) ×

Encapsulation :-

* It is the process of providing controlled access the most important data present within the class.
* in order to provide the security you should declare the variable as private

ex:- python program without encapsulation.

```
class Uc:
    def __init__(self)
        self.maxprice = 35000
        print (self.maxprice) → 35000
        #print (Uc.maxprice) → error
if __name__ == '__main__':
    ref = Uc()
    print (ref.maxprice) → 35000
    #print (Uc.maxprice) → error
    ref.maxprice = 10000
    print (ref.maxprice) → 10,000
```

(we cannot access instance variable using class name)

in the above example the security is not provided to the most important data within the class, so it can be acheived using encapsulation as shown below.

ex2:-

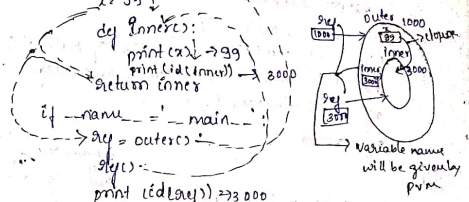
```
class Uc:-
    def __init__(self):
        self.__maxprice = 35000
    def set_maxprice(self, price):
        self.__maxprice = price
    def get_maxprice(self):
        return self.__maxprice
if __name__ == '__main__':
    ref = Uc()
    print (ref._Uc__maxprice) → 35000
    ref._Uc__maxprice = 80000
    print (ref._Uc__maxprice) → 30000
    ref.set_maxprice(40000)
    print (ref.get_maxprice()) → 40000
```

2nd Closures

It is a process of providing the access to the variable present with in the outer function within the inner()

ex:- 1

```
def outer():
    x = 99
    def inner():
        print (x) → 99
        print (id(inner)) → 3000
    return inner
if __name__ == '__main__':
    ref = outer()
    ref()
    print (id(ref)) → 3000
    print (id(outer)) → 1000
```



→ once the access is provided for the variable present within outer outer() into inner() using closures even if the outer() is deleted the access cannot be taken as shown below.

ex:-2

```
def person():
    x = 'sachin'
    def player():
        print(x)
    return player
if __name__ == '__main__':
    r = person()
    r()
    del person
    r()
```

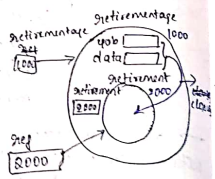
O/P

sachin
sachin

Write a python program to collect the year of birth and the retirement age by the user & display his current age & also how many years are left out to get retired using the concept of closures.

```
import datetime
def retirementage(yob):
    data = 'years left out to get retired
```

```
def retirement(ra):
    age = int(input(datetime.datetime.now().year) - yob
    print(f'The age is {age}')
    print(f'{ra-age} {data}')
    return retirement
if __name__ == '__main__':
    yob = int(input("Enter the year of birth"))
    ra = int(input("Enter the retirementage"))
    ref = retirementage(yob, yob)
    ref(ra=ra)
```



O/P

Enter the year of birth 1998
Enter the retirement age 60
The age is : 21
38 years left out to get retired

Ex:-

```
def Outerfunc(job):
    def innerfunc(name):
        if job == 'cricketer':
            print(f'{name} plays cricket')
        elif job == 'java':
            print(f'{name} teaches java')
        elif job == 'python':
            print(f'{name} teaches python')
        elif job == 'prog':
            print(f'{name} teaches programming')
        else:
            print(f'{name} teaches something else')
    return innerfunc
if __name__ == '__main__':
    ref = outerfunc(job='cricketer')
    ref(name='sachin')
    ref = outerfunc(job='java')
    ref(name='bheemesh')
    ref = outerfunc(job='python')
    ref(name='sandesh')
    ref = outerfunc(job='prog')
    ref(name='somanna')
```

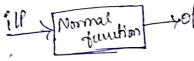
O/P

sachin plays cricket
bheemesh teaches java
sandesh teaches python
somanna teaches programming

Imp

Decorator:- is a function which takes the function name as i/p and returns the modified function with some additional functionalities as o/p.

Python Program using only functions.



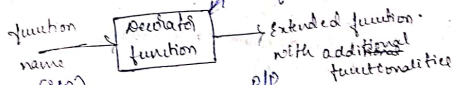

```
def with(name):
    print('hey', name, 'good morning')
if __name__ == '__main__':
    with('sachin')
    with('dhoni')
```

→ Return example, irrespective of inputs provided but the o/p remains same, so inorder to add some additional functionalities we can make use of decorators

O/P

hey sachin good morning
hey dhoni good morning

Python program using decorator function:



Ex:-

```
def outer(func):
    def inner(name):
        if name == 'dhoni':
            print(f'hey {name} good afternoon')
        elif name == 'kohli':
            print("hey, {} good evening".format(name))
```

```
        else:
            func(name)
    return inner
def wish(name):
    print('hey', name, 'goodmorning')
if __name__ == '__main__':
    ref = outer(func = wish)
    ref(name = 'sachin')
    ref(name = 'dhoni')
    ref(name = 'kohli')
```

X (~~In the above example, irrespective of input in given but o/p is same add some additional functionalities we can make~~) X

O/P

hey sachin goodmorning
hey dhoni goodafternoon
hey kohli good evening

Ex2:-

```
def outer_decorator(d):
    def inner(x,y):
        if y==0:
            print('not possible to perform division')
        else:
            d(x,y)
    return inner
def division(a,b):
    print(f'The result is: {a/b}')
if __name__ == '__main__':
    div = outer_decorator(d = division)
    div(x = 10, y = 5)
    div(x = 10, y = 0)
```

O/P

The result is: 2.0
not possible to perform division

@decorator method property:-

This will be used while declaring the independent function by using the decorator function name.
at this instance of time if we call that independent function then by default pvm would make a call to the outer function as well as inner function.

Ex:-1

```
def outer_decorator(d):
    def inner(x,y):
        if y==0:
            print("not possible to perform division")
        else:
            d(x,y)
    return inner
@outer_decorator
def division(a,b):
    print(f'The result is: {a/b}')
if __name__ == '__main__':
    division(x = 10, y = 2)
    division(x = 10, y = 0)
```

O/P

The result is: 5.0
not possible to perform division.

Ex:- Note:- using a single decorator function we can decorate multiple independent function as shown below.

Ex:-2

```
import logging
logging.basicConfig(filename = 'decorator.txt', level = logging.
    DEBUG, filemode = 'w')
def outer(func):
    def inner(x,y):
        logging.debug(f'The data stored within x:{x}')
        logging.debug(f'The data stored within y:{y}')
        func(x,y)
    return inner
@outer
def add(x,y):
    print(f'The added result is: {x+y}')
    logging.debug(f'The added result is: {x+y}')
```

```
def sub(x,y):
    print("The subtracted result is :",x-y)
    logging.debug("The subtracted result is :",x-y)
def mul(x,y):
    print("The multiplied result is :",x*y)
    logging.debug("The multiplied result is :",x*y)
def div(x,y):
    print("The divided result is :",x/y)
    logging.debug("The divided result is :",x/y)
if __name__=='__main__':
    add(x=10,y=20)
    sub(x=20,y=10)
    mul(x=10,y=10)
    div(x=100,y=10)
```

O/P

```
The added result is : 30
The subtracted result is : 10
The multiplied result is : 100
The divided result is : 10.0
```

decorator1 :
The data stored within x : 10
The data stored within y : 20

Decorator chaining

* It's a mechanism where a single independent function can be decorated using multiple decorators.
* If a multiple decorators are used then the first decorator used to decorate independent function will get executed first and the remaining decorators will be chained. This process is called as decorator chaining.

Ex:-

```
def decorator1(func):
    def inner(name):
        print("decorator1 is executed")
        func(name)
    return inner
def decorator2(func):
    def inner(name):
        print("decorator2 is executed")
        func(name)
    return inner
```

```
@decorator2
@decorator1
def wish1(name):
    print("hey",name,"how are you??")
@decorator1
@decorator2
def wish2(name):
    print("hey",name,"how r you??")
if __name__=='__main__':
    wish1('sachin')
    wish2('david')
```

O/P

```
decorator2 is executed
decorator1 is executed
hey sachin how are you??
decorator1 is executed
decorator2 is executed
hey david how r you??
```

Ex2:-

```
def decorator1(func):
    def inner():
        data = func()
        print(data)
        return data*2
    return inner
def decorator2(func):
    def inner():
        data = func()
        print(data)
        return data*data
    return inner
@decorator1
@decorator2
def num():
    return 5
@decorator2
@decorator1
```

```
def num1()
    return 10
if __name__ == '__main__':
    print(num1())
print(num1())
```

O/p
5
25
50
10
20
400

Note:- In decorator chaining if return statement is present then the order of execution would be executed in upward direction. of the decorator.

* If the decorator function doesn't have any return statement in the order of execution of the decorator would be executed in downward direction

diff b/w decorator function and Normal function

decorator function	Normal function
* The decorator method would be called automatically by the PVM provided if we call the independent function	* Normal function has to be called explicitly by the programmer.
* we can extend the functionality of the independent function without modifying independent function.	* we cannot extend the functionality of any independent function.

Note:- * using a single decorator we can decorate multiple independent function.

* using multiple decorators we can decorate a single independent function.

Iterator:-

1. It a process of iterating over the containers (list, tuples, ~~brackets~~).
* using iter() we can create iterable object.
* using the method (magic method → __iter__()) we can create the iterable object without using iter().
* in order to fetch the data with in iterable objects they have to declare magic method called as __next__.

Containers and non-containers

Containers are the data types which can hold the address of multiple object.

In other words, containers can hold multiple elements

Ex:-
```
lst = [10, 20, 30]
tup = (10, 20, 30)
set = {1, 2, 3}
dict = {1:'a', 2:'b'}
```

non-containers :- non containers are the data types which cannot hold the address of multiple objects.

In other words, using non containers we cannot store multiple elements.

Ex:-
```
int → a = 10
float → b = 10.5
complex → c = 3+1j
boolean → d = True
```

Converting the containers into iterable objects.

a containers can be converted into iterator objects using 'iter()'.

in order to fetch the data stored within the iterator object we should use '__next__()'.

Ex1:-
```
lst = [1,2,3]
print(lst) → [1,2,3]
print(id(lst)) → 1000
x = iter(lst)
print(x) → <list_iterator object at 0x01DEBCE0>
print(id(x)) → 2000
print(x.__next__()) → 1
print(x.__next__()) → 2
print(x.__next__()) → 3
# print(x.__next__()) → stopiteration
```

Iterating over a container and iterable object

for loop

```
lst = [1,2,3]
print (lst)
x = iter(lst)
for i in lst:
    print (x.__next__())
```

Op:
1
2
3

Note:- using non container if we try to iterate over a container then default steps happens to be 1.

- using non container if we try to iterate on iterable object then by default the steps happens to be 2.

for loop

```
lst = [1, 2, 3]
x = iter(lst)
for in x:
    print (x.__next__())
```

Op: 2
StopIteration

Converting the container into iterable objects & iterate using for loop.

```
lst = [1, 2, 3]
print (lst)
x = iter (lst)
print(x)
i=0
while i< len(lst):
    print (x.__next__())
    i = i+1
```

op
[1, 2, 3]
<lst_iterator object at 0x0031BF70>
1
2
3

Itertool module :-

(i) islice() :- general syntax :- islice (iterable, start, stop, step)
(Sequence) inclusive exclusive default = 1

Case(i): providing sequence as iterable to islice()

Ex:-
```
from itertools import *
lst = [1,2,3,4,5,6,7]
x = iter(lst)
for i in islice (lst, 0, 7):
    print (i)
```
& (lst, 0, 6) -> Op 1 2 3 4 5 6

```
for i in islice (lst, 0, 7):
    print (x.__next__())
```

Op
1 1
2 2
. .
. .
7 7

Note:- If the sequence is provided as iterable then we can perform iteration using n no. of for loops.

Case ii: providing the iterable objects as iterable to islice()

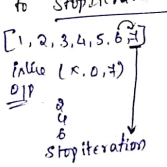
Ex- islice (iterable, start, stop, step)
(iterable object) inclusive default=2

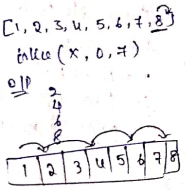
```
from itertools import *
lst = [1, 2, 3, 4, 5, 6, 7]
x = iter(lst)
for i in islice (x, 0, 7)
    print (i)
for i in islice (x, 0, 7)
    print (x.__next__())
```

Op
1 end of stopiteration
3 (bez. first for loop get executed & multiple for loop are not allowed)
5
7

Note:- If the iterable object is passed as iterable the multiple objects cannot be used to perform iteration.

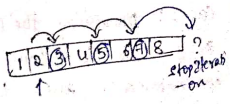
- an iterable object provided as an iterable if we try to fetch the value using __next__() then for the first iteration the step will be one & from the second iteration onwards the step will be 2.
- while iterating if there is no element after the iteration then it leads to StopIteration.





Ex:- 2

```
from itertools import *
lst = [1,2,3,4,5,6,7,8]
x = iter(lst)
for i in islice (x,1,7):
    print (x.__next__())
```



Op
2
4
6
StopIteration

(ii) Cycle()

Case(i):- paring sequence as iterable

general syntax :- cycle (iterable)
↓ sequence

Start } inclusive
Stop }

Cycle takes only one argument

Ex:1

```
from ertertools import *
lst = [1,2,3]
x = iter(lst)
for i in cycle(lst):
    print(i)
```

O/P

```
1
2
3
1
2
3
1
2
3
1
.
.
continuous...
```

Note:- In the above example sequence is pass as an argument to the cycle() while fetching the data if we use non container() then the cycle keeps on continuing.

Ex:2:- In the below example sequence is passed as an argument to the cycle() but while retrieving the data if we use __next__(). then the cycle will stop it there is no value to iterate.

* In other words it leads to stop iteration as shown below

```
from itertools import *
lst = [1,2,3]
x = iter(lst)
for i in cycle(lst):
    print(x.__next__())
```

O/P

```
1
2
3
Stop Iteration
```

Case2:- providing the iterable objects as an argument in the cycle()

Ex:-1 In the below example if the data is retrieve using non container() then the steps happens to 1 and the cycle keeps on continuing.

```
from itertools import *
lst = [1,2,3,4]
x = iter(lst)
for i in cycle(x)
    print(i)
```

O/P

```
1
2
3
4
continue...
```

Ex2:- In the below example the data in been fetched using iterable object (__next__()) then the cycle stops if there is no value for iteration as shown below.

```
from itertools import *
lst = [1,2,3,4,5,6,7]
t = iter(lst)
for i in cycle(x)
    print(x.__next__())
```

→ it is used to iterate on Iterable object

O/P

```
1 2
2 4
3 6
StopIteration).
```

Iterating over the containers without using iterator

```
if __name__ == '__main__':
    my_list = ['ABC', 'for', 'Technology']
    for data in my_list:
        print(data)

    my_tuple = ('Java', 'Python', 'testing', 10)
    for data in my_tuple:
        print(data)
```

O/P

```
ABC
for
Technology
Java
Python
testing
10
```

Converting the container into iterable object via. iterable object.

* In order to convert the container into iterable object using iter() .
* If the object is converted into i.o by default pvm wants to execute magic method called as __iter__().

Ex:-
```
class Abc:
    def __init__(self, x):
        self.x = x
    def __iter__(self):          # to convert into Iterable object without using iter
        return iter(self.x)
if __name__ == '__main__':
    ref = Abc(['ABC', 'for', 'Technology', 'Training'])
    for data in ref:
        print(data)
```

O/P
ABC
for
Technology
Training

Ex2:-
```
class Alleven:
    def __init__(self):
        self.x = 0
    def __iter__(self):
        return self
    def __next__(self):
        self.x = self.x + 2
        return self.x
if __name__ == '__main__':
    All = Alleven()
    for data in iter(range(0, 10)):
        print(data)
```

O/P
2 10 20
4 12
6 14
8 16
18

Napp to obtain the fibonacci series using the concept of iterator.

```
class Fibo:
    def __init__(self):
        self.prev = 0
        self.curr = 1
        print(prev)
        print(curr)
    def __iter__(self):
        return self
    def __next__(self):
        self.prev, self.curr = self.curr, self.prev + self.curr
        return self.curr
if __name__ == '__main__':
    f = Fibo()
    for i in range(iter(f, 0, 5)):
        print(i)
```

O/P
0
1
1
2
3
5
8

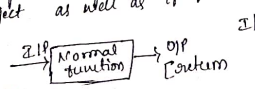
Napp to print the data from 11 to 15 using concept of iterator

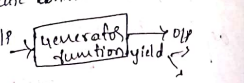
```
class Test:
    def __init__(self):
        self.limit = 15
    def __iter__(self):
        self.x = 10
        return self
    def __next__(self):
        self.x = self.x + 1
        return self.x
if __name__ == '__main__':
    t = Test()
    for i in t:
        print(i)
```

O/P 11
12
13
14
15

Generator

Generator is a function which is responsible to ...
- the sequence of objects
- In case of generator() to return the o/p we should ...
- 'yield' key word
- yield key word returns the o/p by creating generator object as well as it will take control back.





- In order to fetch the data present within generator object we should make use of next()
- We can iterate over a generator() so generator object is an iterable object.
- while iterating over generator object if there is no value to iterate it would result in StopIteration.

Normal function:-

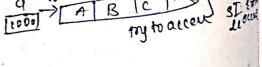
```
def Alphabets():
    return 'A'
    return 'B'
    return 'C'
if __name__=='__main__':
    a=Alphabets()
    print(a)
```

O/P
A

Generator function:-

```
def Alphabets():
    yield 'A'
    yield 'B'
    yield 'C'
if __name__=='__main__':
    a=Alphabets()
    print(next(a))
    print(next(a))
    print(next(a))
    #print(next(a))
```

O/P
A
B
C
StopIteration



Difference b/w Normal function and generator function

Normal function

```
def func():
    num=1
    print(num)
    return num

    num=2
    print(num)
    return num
if __name__=='__main__':
    print(func())
```

O/P
if we use return keyword once it return the o/p it won't be back.

Ex:-

generator function

```
def func():
    num=1
    print(num)
    yield num
    num=2
    print(num)
    yield num
    print(num)
    yield 3
if __name__=='__main__':
    a=func()
    print(next(a))
    print(next(a))
    print(next(a))
    print(next(a))
```

O/P
<generator object func at ...>
1
1
2
2
3
3
1st print
2nd print
3rd print
yield is executed when we print(next()) we didn't use next() o/p will not be retrieve
StopIteration (way to prove generator is a iterable) if we print(next(a)) more

Ex:-2

```
def display():
    name='sandeep'
    print(name)
    yield name
    yield name
    print(name)
    name='subas'
    print(name)
    yield name
    yield 'akash'
if __name__=='__main__':
    d=display()
    print(next(d))
    print(next(d))
    print(next(d))
```

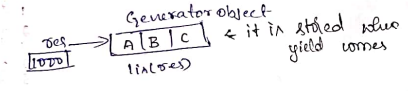
```
print(next(a))
print(next(a))
```

O/P
Sandesh
Sandesh
Sandesh
Sandesh
Sandesh
Suhas
Suhas
akash

generator function is an iterable object so we can provide generator object as an I/p to the container as shown below

Ex:-
```
def alphabets():
    yield 'A'
    yield 'B'
    yield 'C'
if __name__=='__main__':
    res=alphabets()
    l=list(res)
    for i in l:
        print(i)
```

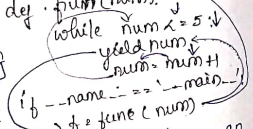
O/P
A
B
C



Program 3

using generators generate series from 1 to 5 (using loops)

```
def fun(num):
    while num<=5:
        yield num
        num=num+1
if __name__=='__main__':
    f=fun(num)
    for i in f:
        print(i)
```



O/P 1 2 3 4 5

using generators obtain the series of prime no till 15 (using loops)

using generator generate series till 50 where in which the no. is divisible 5.

```
def fun(num):
    while num<=50:
        yield num
        num=num*5
```

Program 4

write a python program to generate the squared no. of the given sequence using the concept of normal function & generate function (use yield keyword)

Normal function
```
def func():
    lst=[1,2,3,4,5]
    new_lst=[]
    for i in lst:
        new_lst.append(i*i)
    return new_lst
if __name__=='__main__':
    f=func()
    for i in f:
        print(i)
```

generate function
```
def func():
    lst=[1,2,3,4,5]
    for i in lst:
        yield i*i
if __name__=='__main__':
    f=func()
    for i in f:
        print(i)
```

O/P: [1,4,9,16,25]

O/P: 1 4 9 16 25

Generator expression

In general, if we try to perform tuple comprehension it will generate generator object without using yield keyword

using generator expression obtain the squared numbers for given sequence [1, 2, 3, 4, 5]

```
def func():
    lst=[1,2,3,4,5]
```

```
new_lst = (i*i for i in lst)   → generator expression
f = func(new_lst)
for i in f:
    print(i)
```

o/p: 1, 4, 9, 16, 25 (8)

```
def func():
    lst = [1,2,3,4,5]
    g = (i*i for i in lst)
    for i in g:
        print(i)
if __name__ == '__main__':
    func()
```

Wap to generate fibonacci series till 10 using the concept of generator.

```
def fun(fib0, fib1)
    while fib0 < 10
        yield fib0
        fib0, fib1 = fib1, fib0+fib1
if __name__ == '__main__':
    f = fun(fib0=0, fib1=1)
    for i in f:
        print(i)
```

Difference b/w normal function & generator function

normal function	generator function
* in order to return the o/p within the n.f we should make use of return keyword	* in order to return the o/p we should make use of yield keyword
* whenever return statement get executed the function would be terminated	* whenever yield keyword executed the generator function it will generate generator object the control goback to yield

If the n.f is finished the execution it would not raise the any exception.

* whenever the funct^n is called then the execution begins immediately

* whenever the generator function executes till last element & if try to fecth the data using next() stop iteration exception is generated
* generator funct^n is called after the execution generator object will be created using
  - → yield keyword
  - → generator expression

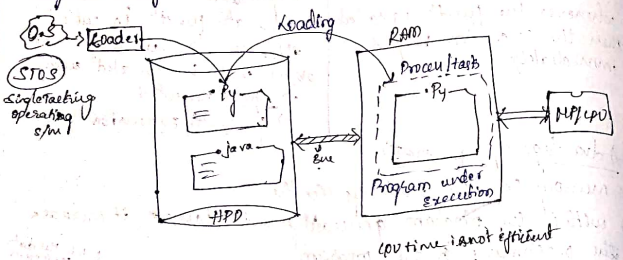
Advantage of generator

* memory can be used effectively
* with in the program generator funct^n is used it increase the performance by the program.
* generator function is used in the field of data scrolling & data scraping, fetch the data & use the data (fetch the data / for scrapping use)
* generator should be used when we work with large amount of data to generate series using the given set of i/p

pid
Post no.
who will
quit
→ loaded
present
with in
operating
sim.
HTTP & HTTPS
process

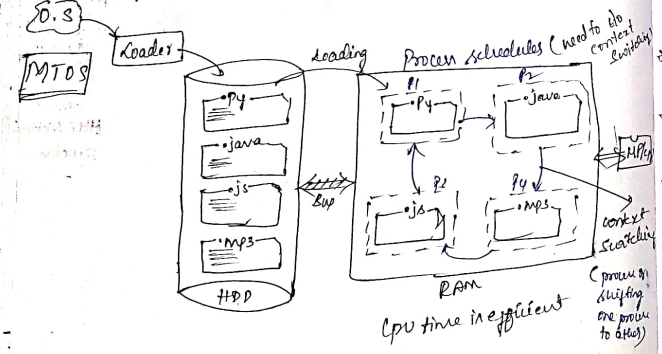
Multithreading

Single tasking



* Single tasking is executing one task at a time
* If on the RAM if single process/task is getting executed then it is referred as 'single tasking'

Multi tasking

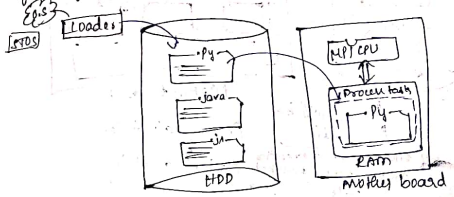


Note:- * On a RAM, if multiple process & tasks is getting executed simultaneously then it is referred as 'Multi tasking'

* In case of multitasking the process of switching from one process to another is referred as 'context switching'
* process scheduler present within operating system is responsible for context switching activities.
* basically a process/task is a program under execution.
* In case of, single tasking cpu time is not utilized properly
* In case of multitasking cpu time is properly utilized.

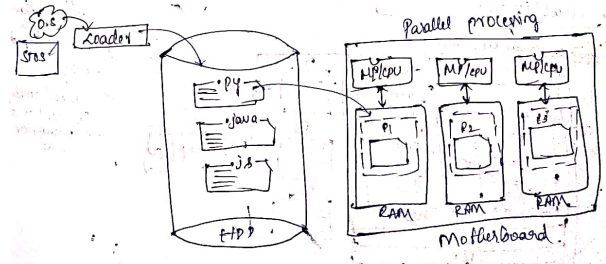
Single tasking operating S/m [1970's]

* In case of, single tasking o/s cpu time is not utilized efficiently and the process will get executed efficiently.



Single tasking operating S/m [1980's]

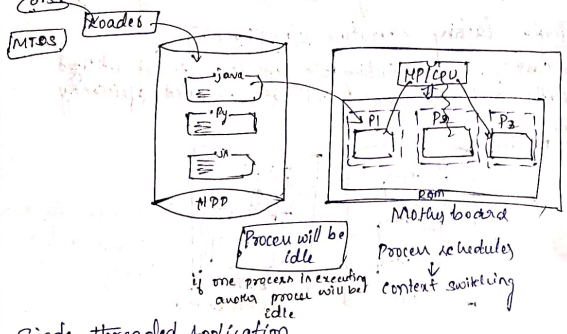
* In case of the below diagram in order to execute multiple files multiple microprocessor and multiple RAMs will be used.
* but in some of the files if these in mutual exclusive resources, then cpu will be at idle state.
* In this case execution time of processes will be done in the efficient manner and the cost of the s/m will be increased



Mutual exclusive resources

Multi tasking operating sys

* In this case, the CPU time will be utilized properly by multiple process will be running simultaneously on the RAM and equal chances (or time) will be given for all the process via context switching



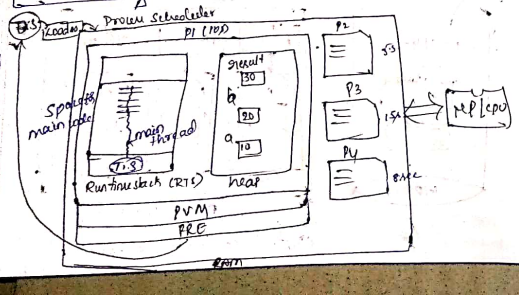
Single threaded Application

Ex 1:-

```
if __name__ == '__main__':
    a = int(input('enter the first operand:'))
    b = int(input('enter the second operand:'))
    res = a+b
    print('The result is :',res)
```

O/P

```
enter the first operand : 10
enter the second operand : 20
The result is : 30
```



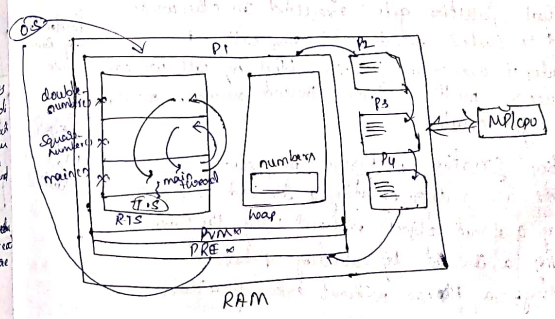
* thread is used to execute single line program
* PVM will give control to the thread scheduler
* Multithreading - eg: more than one line of a program independently
* Thread scheduler will create main thread

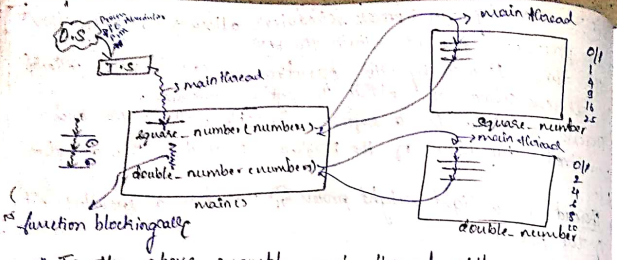
Note:- operating sys (process scheduler) allocate a specific time to the processes present within the RAM

* the time shown by the operating sys (PS) will be collected by thread scheduler within the python program.
* Thread scheduler is a software which will control and coordinate the lines of the program present within a python file.
* 'Thread' is a light weight process (ie) Thread is a smallest unit of the process

Ex2:-

```
import time
def square_number(numbers):
    for num in numbers:
        time.sleep(2)
        print(num * num)
def double_number(numbers):
    for num in numbers:
        time.sleep(2)
        print(num)
if __name__ == '__main__':
    numbers = [1,2,3,4,5]
    square_number(numbers)
    double_number(numbers)
```





* In the above example main thread will execute main followed by lines present within a main function
* within a main() there two functions that are getting called.
* Untill ~~evaluvation~~ the function get executed the next function will be in blocking state this mechanism is referred as function blocking call.

Multi threaded Application

In case of single thread application cpu time in not utilized efficiently because if one function calls execution then the other function has to wait till that function gets executed. In other words it is called function blocking call.

In order to over come function blocking call we can make use of multi function thread means, multiple stacks get created.

Creating a thread in python.

There are 3 ways of creating a thread in python.

1) creating a thread object without using the class
2) creating a thread by extending a class.
3) creating a thread without extending a class

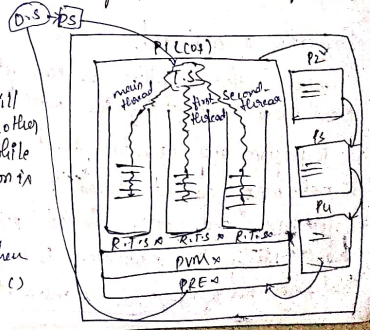
* in java thread can be create using extending thread class & implementing Runnable interface

Creating a thread object without using the class

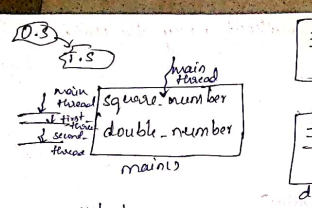
```
import time
from threading import *
def square_number(numbers):
    for number in numbers
        time.sleep(2)
        print(number*number)
def double_number(numbers):
    for number in numbers:
        time.sleep(2)
        print(2*number)
if __name__=='__main__':
    numbers = [1,2,3,4,5]
    begin_time = time.time()
    first_number = Thread(target=square_number, args=(numbers,))
    second_number = Thread(target=double_number, args=(numbers,))
    first_thread.start()
    second_thread.start()
    first_thread.join()  -> just for delay
    second_thread.join()
    end_time = time.time()
    print('time taken by cpu in :', end_time - begin_time)
```

O/P
1
4
9
16
25
2
4
6
8
10
10.08

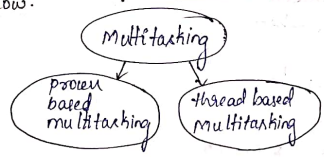
(because, we use sleep() hence the it will jump to another function while other function is sleeping)



multiple stacks created when we use start()



Multitasking can be achieved using two ways as shown below.



Multitasking → O.S.
Multithreading → Thread Scheduler

* Process Scheduler is responsible for multitasking.
* Thread Schedules is responsible for multithreading

Process based Multitasking

It is the process of executing several tasks simultaneously where each task is seperate independent process.

Thread based Multitasking

It is the process of executing several tasks simultaneously where each seperate independent part of same program. that independent part are referred as Threads

Properties of a thread.

using setters & getters we can get the name of current thread we can change the name of thread

* to perform any operation w.r.to current thread we have to make use of current_thread().
* for each and every thread pvm would allocate a unique identification number. to to fetch that number we can make use of "ID" Ident & statements

Ex:-
```
from threading import *
if __name__ == '__main__':
    print ('the current thread is :', current_thread().getName())
    current_thread().setName(name = 'mythread')
    print('the current thread is: ', current_thread.getName())
    print ('the id no of current thread: ', current_thread().ident)
```

O/P
The current thread is : main thread
" " : my Thread
The id no. of current thread : 8520

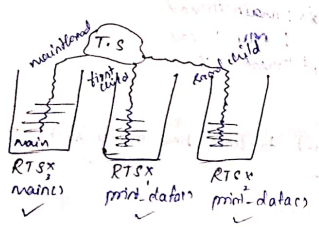
active_count():-
- this function is used to check how many stacks (threads) are present within one process.

Ex:-
```
from threading import *
import time
def print_data():
    print ('current_thread().getName() got started!!')
    time.sleep(3)
    print ('current_thread().getName() got ended!!')
if __name__ == '__main__':
    print ('the no. of threads that are active:
            active_count()')
    first_thread = Thread (target = print_data, name = 'first child')
    second_thread = Thread (target = print_data, name = 'second child')
    first_thread.start()
    second_thread.start()
    print ('the no. of threads that are active: active_count()')
    time.sleep(5)
```

print(f'the no of active threads : {active_count()}')

O/P

The no. of active threads that are active : 1
first child got started!!
The no. of threads that are active : 3
second child got started!!
first child got ended!
second child got ended!!
The no. of active threads : 1



enumerate() :-

The information of the thread (stacks) will be stored within 'enumerate()' in the form of list.

* while iterating over the 'list' specifically we want to fetch the name(value) then we have to make use of the statement called as 'name'.

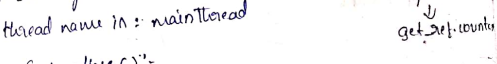
Ex:-

```
from threading import *
import time
def print_data():
    print(f'{current().getName()} got started')
    time.sleep(3)
    print(f'{current().getName()} got ended')
if __name__=='__main__':
    my_threads = enumerate()
    for thread in my_threads:
        print(f'The thread name : {thread.name}')
```

```
first_thread= Thread(target = print_data,name = 'firstchild')
second_thread = Thread(target= print_data, name='secondchild')
first_child.start()
second_thread.start()
my_threads = enumerate()
for thread in my_threads:
    print(f'The thread name is : {thread.name}')
time.sleep(5)
my_thread = enumerate()
for thread in my_threads:
    print(f'The thread name is : {thread.name}')
```

O/P

first child got started
the thread name is : mainthread
the thread name is : firstchild
the thread name is : second child
second child got started
first child got ended
second child got ended
The thread name is : mainthread



sys module
↓
getrefcount

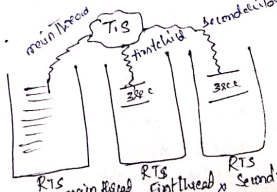
isAlive() & is_alive() :-

X 3.0 X 3.0 version

this function is used to check the active status of threads.

* whenever we create a thread by default python will give the name for those threads. (Thread-1, Thread-2 ---).

* If a thread is in active state is_alive() is going to return in the o/p as true & if the thread is in dead state then the o/p will be False



Ex:- from threading import *
import time
def print_data():
print(f'{current_thread().getName()} got started!!')
time.sleep(3)
print(f'{current_thread().getName()} got ended!!')
if __name__ == '__main__':
print(f'{current_thread().name} isAlive: {current_thread().isAlive()}')
first_thread = thread(target=print_data, name='first child')
second_thread = thread(target=print_data, name='second child')
first_thread.start()
second_thread.start()
print(f'{current_thread().name} isAlive: {current_thread()!!isAlive()}')
print(f'{first_thread.name} isAlive: {first_thread.isAlive()}')
print(f'{second_thread.name} isAlive: {second_thread.isAlive()}')
time.sleep()

O/P
main thread isAlive: True
first child got started!!
Second child got started
main thread isAlive: True
first child isAlive: True
second —"— : True
first child got ended!!
second child got ended!!
main thread isAlive: True
first child isAlive: False
Second child isAlive: False

join():
→ this function is used to wait till one thread completes its operation
→ join(seconds) is used to give the proper time to wait till one thread completes its operation.

Ex:- from threading import *
import time
def db_connect:
from data in range(5):
print("connected to database!!')
time.sleep(2)
if __name__ == '__main__':
db_thread = thread(target=db_connect, name='dbcpnt')
db_thread.start()
db_thread.join()
for data in range(5):
print('dis connected from database!!')

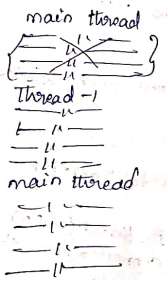
O/P
connected from database!!
dis connected from database!!

db_thread.join(5)
Connected from db
dis
connected from db
connected from db
(Bcoz, within 5 seconds db_connect should execute otherwise main thread will get executed)

Creating a thread by extending Thread class
→ If a thread has been created by extending Thread class then with in the class programmer should override run() (Instance method).
→ In order to execute the run() programmer should call start().

Ex:- from threading import *
class 'My_task (Thread):
def run(self):

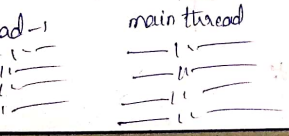
```
        for data in range(5):
            print(my_thread.name)
if __name__=='__main__':
    print(current_thread().name)
    my_thread=mythread task()
    my_thread.start()
    my_thread.join()
    for data in range(5):
        print(current_thread().name)
```



3) Creating the thread without extending thread class

```
Ex:- from threading import *
     class mytask:
         def display(self):
             for data in range(5):
                 print(my_thread.name)
     if __name__=='__main__':
         rej=mytask()
         my_thread=Thread(target=rej.display)
         my_thread.start()
         my_thread.join()
         for data in range(5):
             print(current_thread().name)
```

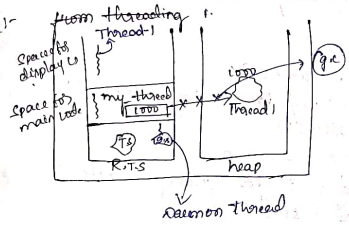
O/P



Daemon thread:- Daemon threads are the helper threads which should be running on the background to support non daemon thread so that a user can execute the application/program in a birway

* the main objective of daemon thread is to support non-daemon thread
* within the python program - best example for daemon thread happens to be garabage collector.
* If the object is having any reference then the object will be destroyed by one thread be which belongs to the garbage collector. that thread name happens to be daemon thread.

Ex:-



Here: the There is a daemon thread in garbage collector which will delete the object

to check

* whether the thread is daemon or not in two way
  - → isDaemon
  - → daemon property
* using set daemon() we can provide daemonic property to the thread by assigning true.
* by default every thread is considered as non-daemon thread.

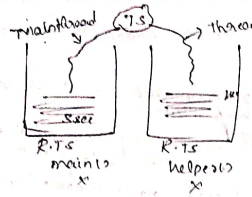
Ex:- 1 setting the daemon property to the running thread

```
from threading import*
if __name__=='__main__':
    print(f'{current_thread().name} is a daemon thread:
          {current_thread().isDaemon()}')
    print(current_thread().daemon)
    current_thread().daemon(True)
    print(f'{current_thread().name} is a daemon thread:
          {current_thread().isDaemon()}')
```

main thread is a daemon : False
False

Runtime error : cannot set daemon status of active thread

Ex2:-




```
from threading import *
import time
def helper():
    for data in range(5):
        print('hey iam a thread')
        time.sleep(1)
if __name__ == '__main__':
    my_thread = Thread(target = helper)
    my_thread.start()
    time.sleep(3)
```

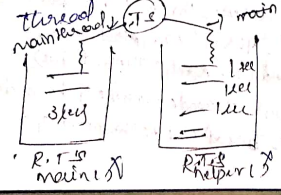
O/P

```
hey iam a thread
hey iam a thread
hey iam a thread
```

Note:- the above example the stack which is allocated for thread-1 it is not dependent on stack allocated for main thread

* main() will wait untill helper() completes its task.

Ex:- in providing Daemonic property for user defined thread




```
from threading import *
import time
def helper():
    for data in range(5):
        print('hey iam Daemon')
        time.sleep(1)
if __name__ == '__main__':
    my_thread = Thread(target = helper)
    my_thread.setDaemon(True)
    my_thread.start()
    time.sleep(3)
```

O/P

```
hey iam Daemon
hey iam Daemon
hey iam Daemon
```

* In the above example thread-1 is Daemon thread & it will be acting as Helper thread for main thread.
* once the main thread completes its activities then thread-1 (Daemon) will stop its execution
* in other words. If the main runtime stack is deleted for main thread by default the runtime stack of Daemon thread will be deleted.

Ex:-

```
from threading import *
import time
def mytask():
    print(current_thread().isDaemon())
    while True:
        print('autosuggestions!!')
        time.sleep(1)
if __name__ == '__main__':
    my_thread = Thread(target = mytask, name = 'birdy')
    my_thread.setDaemon(True)
    my_thread.start
    for data in range(5)
```

```
    print ('typing python code in sublime')
    time.sleep(1)
print ('typing completed-- program is saved')
```

O/P :-

```
True
autosuggestions
typing python code in sublime.



typing completed--- program is saved
```

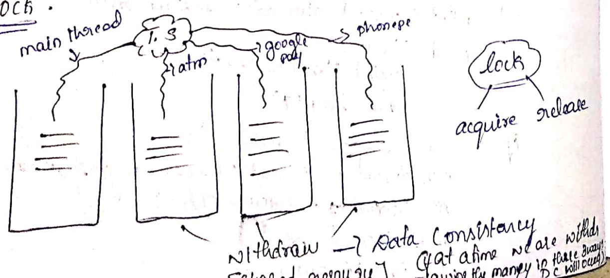
* in the above example autosuggestions thread is Daemon thread it runs on the background support main thread.
* by creating the Daemon thread always we should write a logic in infinite loop.
* if RTS of main thread is deleted the Runtime stack of Daemon thread should compulsarily deleted
* Imp: using setDaemon() we can set the Daemonic property for user defined thread. but incase of main thread we can-not set as Daemon thread bcz. the main thread will-be created by thread scheduler

Impv. Synchronization

Whenever multiple threads acts upon same shared resource than it would lead to data inconsistancy in order avoid this data inconsistency it will make use of synchronization

* In case of python there are 3 levels of synchroonizatn

1) Lock.



Ex:- In the below example withdraw() happens to be the shared resource for 3 threads. so in order to avoid datainconsistency Synchronizations has to be applied for the resource

```
from threading import *
import time
my_lock = Lock()
def withdraw():
    my_lock.acquire()
    for data in range(3):
        print(f'withdrawl of amount is done through:
              {current_thread().name}')
        time.sleep(2)
    my_lock.release()
if __name__ == '__main__':
    atm_thread = Thread(target = withdraw, name = 'atm')
    googlepay_thread = Thread(target = withdraw, name = 'googlepay')
    phonepe_thread = Thread(target = withdraw, name = 'phonepe')
    atm_thread.start()
    googlepay_thread.start()
    phonepe_thread.start()
```

O/P

```
withdrawl of amount is done through: atm
______ " ______
______ " ______
______ " ______ : googlepay
______ " ______
______ " ______
______ " ______ : phonepe
______ " ______
______ " ______
```

Applying Synchronization upon the main function

```
from threading import *
if __name__ == '__main__':
    my_lock = Lock()
    my_lock.aquire()
    print('main code is been locked!!')
    my_lock.release()
```

O/P
main code is been locked

Note:- whenever the lock is applied multiple times using lock for any function then the thread will go to waiting state

* whenever a lock is applied to the resource if we try to acquire the resource using acquire() eventhough if we don't release from resource the resource will be executed.
* whenever a lock is applied to the resource if we try to release from the resource without acquiring the resource it lead to runtimeError exception. (releasereunlock)

```
Locks)
    From importing Thread-*
    if __name__ == '__main__':
        my_lock = Lock
        # my_lock.acquire() -> main lock been locked
        print ("main lock to been locked
        my_lock.release() -> error. (release unlocked key)
```

Eg:-
```
from threading import *
if __name__ == '__main__':
    my_lock = Lock()
    my_lock.acquire()
    print ('main code been locked')
    my_lock.release()
    my_lock.release()
```

O/P:- main thread is in blocking state no o/p on the console

inorder to overcome the above error we have to make use of synchronization level i.e., RLock

Ex:-1
```
from threading import *
if __name__ == '__main__':
    my_lock = RLock()
    my_lock.acquire()
    my_lock.acquire()
    print ('main code is been locked!!')
    my_lock.release()
    my_lock.release()
```

O/P main code is been locked!!

Ex:-2
```
from threading import *
import time
my_lock = RLock()
def fact(number):
    my_lock.acquire()
    if number == 0:
        return 1
    else:
        return number * fact(number-1)
    my_lock.release()
def print_number(number):
    print ("The factorial of", number, "is:", fact(number))
if __name__ == '__main__':
    my_thread = Thread(target = print_number, args=(5,),
                       name = 'factorial')
    my_thread.start()
```

O/P
The factorial of 5 is :120

Difference b/w Lock and RLock

Lock	RLock
* It is referred as simple Lock	* It is referred as Reentrant lock
* It will be applicable only for simple logic	* It will be applicable for both simple and complex logic
* It is not applicable for recursive (or) nested function call	* It will be applicable for recursive (or) nested function call
* once the lock object is obtained as the owner it should be applied only once otherwise the thread would reach the blocking state.	* once the RLock object is obtained as the owner of the thread we can apply the lock n no. of times.

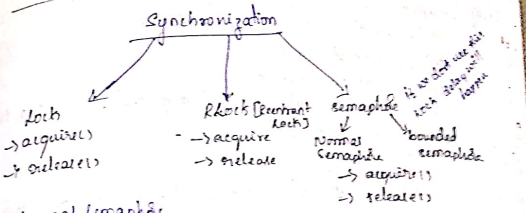
Semaphore

* If the multiple threads are accessing the same shared resources present within the server then the server is not able to respond for all the threads (clients) without any delay.
* In such situation if we apply Lock (or) RLock then that resource could be executed by one thread at a time, the next thread has to wait till the first thread completes its operation.
* So in order to overcome this disadvantage we have to make use of semaphore.

Ex:
```
from threading import *
import time
my_semaphore = Semaphore(2)
def getresult():
    my_semaphore.acquire()
    for data in range(3):
        print(current_thread().name, 'is accessing the result')
        time.sleep(1)
    my_semaphore.release()

if __name__ == '__main__':
    Sachin_thread = Thread(target=getresult, name='Sachin')
    David_thread = Thread(target=getresult, name='David')
    Kohli_thread = Thread(target=getresult, name='Kohli')
    Sachin_thread.start()
    David_thread.start()
    Kohli_thread.start()
```

O/P
```
Sachin is accessing the result
David  is accessing the result
Sachin is accessing the result
David  is accessing the result
Sachin is accessing the result
David  is accessing the result
```

Kohli is accessing the result
" "



Normal Semaphore

Normal semaphore is a unlimited semaphore, means we can acquire the lock as well as we can release the lock many times.
* In case of Normal semaphore there is no restriction for the lock.

Ex:
```
from threading import *
import time
if __name__ == '__main__':
    my_semaphore = Semaphore(2)
    my_semaphore.acquire()
    my_semaphore.acquire()
    print('main thread acquired the lock 2 times')
    my_semaphore.release()
    my_semaphore.release()
    my_semaphore.release()
    print('main thread acquired the lock 3 times')
```

O/P :- main thread acquired the lock 2 times
" " 3 times

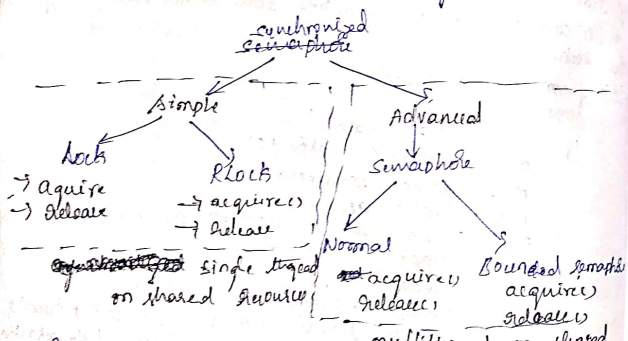
Bounded Semaphore :-

Bounded semaphore is a limited semaphore means the lock we can have to be released based upon how many times the lock has been acquire.
* In case of bounded semaphore there is a restriction to access the lock in other words the lock should be acquired and released simultaneously.

```
from threading import *
import time
if __name__ == '__main__':
    my_semaphore = BoundedSemaphore (2)
    my_semaphore.acquire()
    my_semaphore.acquire()
    print ('main took threaded acquired the lock 2 times')
    my_semaphore.release()
    ---- " ----
    print ('main threaded aequired the lock 3 times')
```

O/p

value error: Semaphore released toomany times



Inter thread communication

It is the mechanism where one thread will communicate with other thread

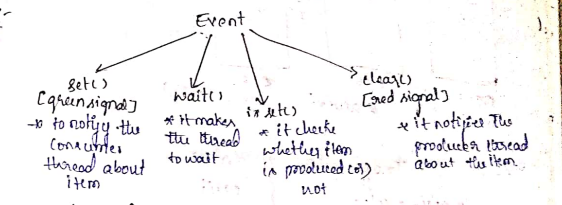
In case of python inter thread communication can be achieved using 3 ways

1) Event
2) condition
3) Queue

1) Event

It is a mechanism of communicating b/w two threads using signals

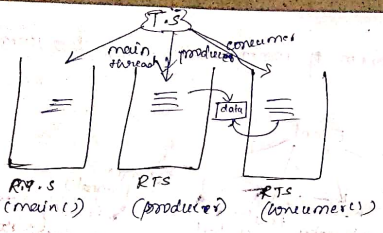
Ex:-




```
from threading import *
import time
def producer():
    time.sleep(10)
    print ('producer has produced the item')
    print ('producer thread has notified the consumer thread')
    my_event.set
def consumer():
    print ('consumer thread is waiting for the notification from prog producer')
    wait my_event.wait()
    print ('consumer thread got notified from producer thread')
    print ('consumer thread consumed the item')
if __name__ == '__main__':
    my_event = Event() # object is created
    producer_thread = Thread (target = producer)
    consumer = Thread = Thread (target = consumer)
    producer_thread.start()
    consumer_thread.start()
```

O/p

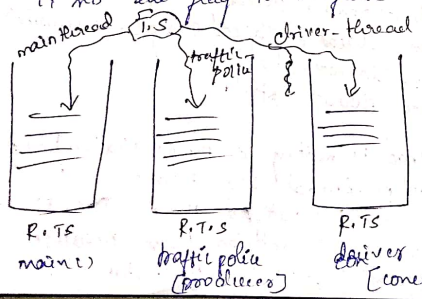
consumer thread is waiting for the notification from producer
producer has produced the item
producer thread has notified the consumer thread
consumer thread got notified from producer thread
consumer thread consumed the item



Ex2:-

Traffic police	drivers
greensignal → set()	wait()
redsignal → clear()	isset()

Note:- * if the threads want to communicate each other via signals then have to make use of event object
* functions used under event objects are
1) set() :- it is used to set the flag as true in other words it will notify the thread which is waiting for notification.
2) wait() :- this function is used to make the thread to be in waiting state until it will get the notification
3) isset() :- this function is used to check the whether the signal is coming from one thread to another thread.
4) clear() :- this function is used to notify the thread that the signal is red In other words internally it set the flag to be false




```
from threading import *
import *

def trafficpolice():
    while True:
        time.sleep(15)
        Print ('traffic police gave green signal')
        my_event.set()
        time.sleep(20)
        print ('traffic police gave red signal')
        my_event.clear()

def driver():
    number=0
    while True:
        print ('driver is waiting for the signal from
                traffic police')
        my_event.wait()
        print ('traffic police gave green signal to driver')
        print ('traffic signal is green... vehicle can move..')
        while my_event.isset():
            number=number+1
            print ('vehicle no:',number,'is moving')
            time.sleep(2)
        print ('traffic police gave redsignal -- vehicles should
if __name__ == '__main__':
    my_event = Event() # event object is created
    trafficpolice_thread= Thread(target = trafficpolice)
    driver_thread = Thread(target=driver)
    trafficpolice_thread.start()
    driver_thread.start()
```

o/p

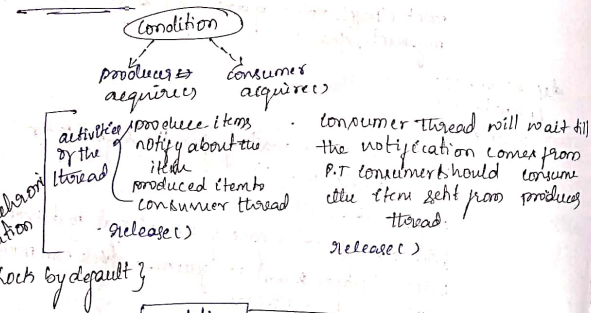
driver is waiting for the signal from traffic police
traffic police gave green signal
traffic police gave green signal to drivers.
traffic police gave green --- vehicle can move---

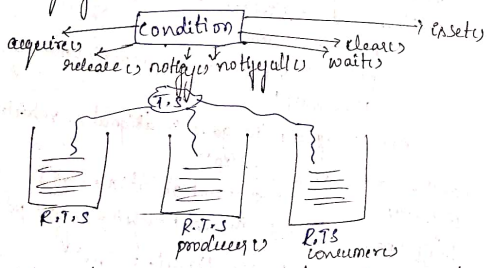
Vehicle no1 moving
vehicle no 2 moving
⋮
— " —— 10 — "

traffic police gave red signal
traffic police gave red signal .. vehicle should stop

~~Condition~~

Condition





Condition object is the updated version of event object to achieve inter thread communication.
threads are communicating

A condition object would represent some state of change in any application like producing the item, consuming the item.

If one of the other thread is executing upon the same share resources then one more thread as to wait till the task is completed.

* In order to overcome the above disadvantage while executing the thread first to we should execute consumer thread.
* Internally condition object will make use of synchronization (RLock).
within condition object some function are used as follows
1) acquire():- this method can be used to acquire the lock present within condition object.
2) Release():- It is used to release the lock on the shared resource
3) notify():- this method is used to send the notification from one thread to another thread.
4) notifyall():- This method is used to notify many threads from one thread.
5) wait():- This function is used to make a thread to wait till it will get the notification.

When to go for Condition object?
* If multiple thread acts upon the same shared resource then it would lead to data inconsistency.
* In order to avoid the data inconsistency b/w the threads we have to use synchronization
* using event object we can't provide synchronization but by default condition object will be having synchronization (RLock)

Ex:-
```
from threading import *
import time
def producer(condition):
    condition.acquire()
    time.sleep(5)
    print('producer thread produced the item!!')
    print('producer thread notified consumer thread!!')
    condition.notify()
    condition.release()
def consumer(condition):
    condition.acquire()
```

```
        print ('consumer thread is waiting for the item')
        condition.wait()
        print ('consumer thread got notified')
        print ('consumer thread consumed the item:')
        condition.release()
if __name__ == '__main__':
    condition = condition()
    producer_thread = Thread (target = producer, args = (condition,))
    consumer_thread = Thread (target = consumer, args = (condition,))
    consumer_thread.start()
    producer_thread.start()
```

o/p

```
consumer thread is waiting for the item
producer thread produced the item.
producer thread notified consumer thread
consumer thread got notified.
consumer thread consumed the item
```

Ex:- using random we can generate random integer & this function will be available within random module

```
from threading import *
import time
import random
lst = []
def producer (condition):
    while True:
        condition.acquire()
        data = random.randint(1,10)
        print ('The producer thread produced the item:', data)
        lst.append(data)
        print ('producer thread is notifying')
        condition.notify()
        condition.release()
        time.sleep(3)
```

```
def consumer (condition):
    while True:
        condition.acquire()
        print ('consumer thread is waiting')
        condition.wait()
        print ('consumer thread got notified')
        print ('consumer thread is consuming the item:
                ', lst.pop())
        condition.release()
        time.sleep(3)

if __name__ == '__main__':
    condition = Condition()
    producer_thread = Thread (target = producer, args=(condition,))
    consumer_thread = Thread (target = consumer, args =
                (condition,))
    consumer_thread.start()
    producer_thread.start()
```

o/p

```
Consumer thread is waiting
The producers thread produced the item 9
producer thread is notified
consumer thread is got notified
Consumer thread is consuming the item 9
```

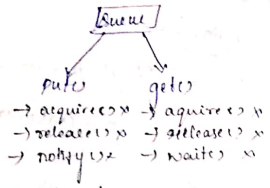
3) Queue

* In order to perform inter thread communication the advanced version will be used & it is referred as Queue.
* In order to create the Queue object we have need queue module.
* within Queue object the logic will be written in the abstract level.
* within Queue object two function will be need.

i) put() : this function is used to send the data item from producer to consumer.

internally within the put function the lock will be acquired & released using acquire() & release()
If the consumer thread is to be notified then it will make use of notify() internally.

2) get(): It is used to get the data item from the producer internally it makes use of acquire function & release() to achieve synchronization as well as whenever the consumer thread is to wait it make use of wait()

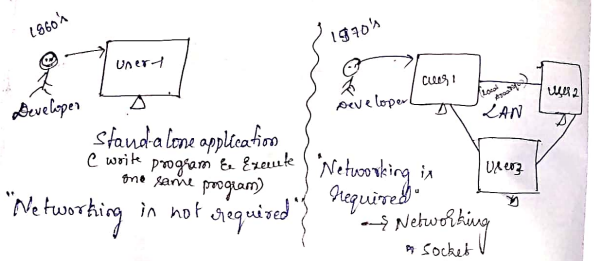


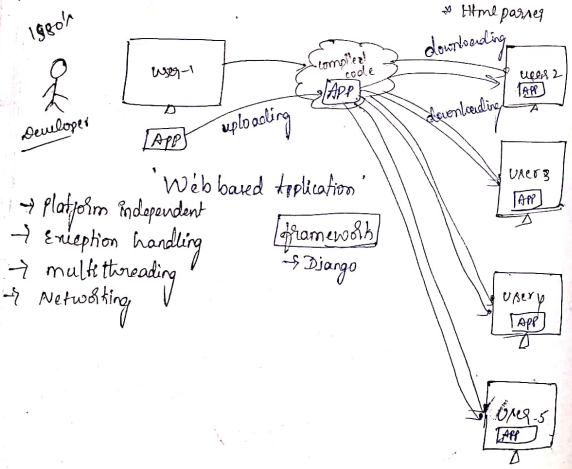
Ex:-
```
from threading import *
import time
import random
import queue
lst = []
def producer(queue):
    while True:
        data = random.randint(1,10)
        print('The producer thread produced the item
              :{data}')
        queue.put(data)
        print('producer thread is notifying')
        time.sleep(7)

def consumer(queue):
    while True:
        print('consumer thread is waiting')
        data = queue.get()
        print('consumer thread got notified')
        print('consumer thread is using the item:{data}')
        time.sleep(1)
if __name__ == '__main__':
    queue = queue.Queue()
    producer_thread = Thread(target=producer, args=(queue,))
    consumer_thread = Thread(target=consumer, args=(queue,))
    consumer_thread.start()
    producer_thread.start()
```

Networking

It's mechanism of connecting two computers. In order to establish the networking b/w two computers following libraries will need:

1) Socket
2) URLlib
3) Html parser



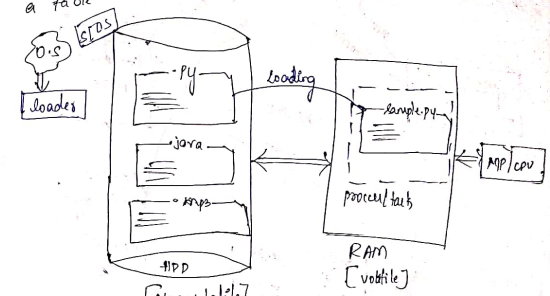


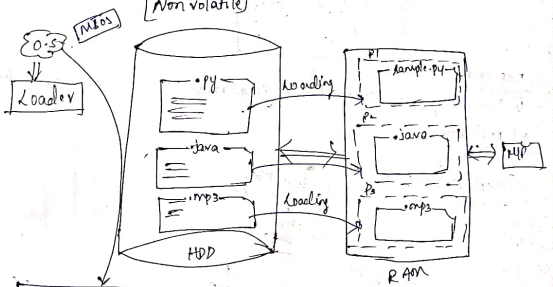
Port no:-

Port no. is the address given by the operating sys (process wise) for the process getting executing on the RAM.

Port no. is also referred as pid of process no.

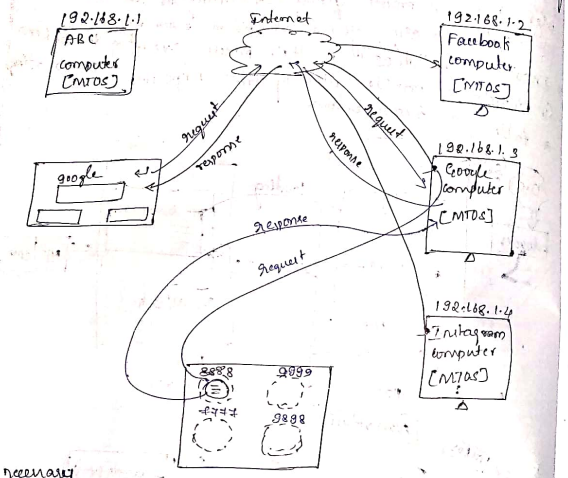
* Incase of multiple OS (tasking) multiple process executing on the RAM. So inorder to provide time of control (CPU time) PS creating a table as shown below.





process name	Process ID / port no. / pid
.py	8888
.java	9090
.mp3	7070

Necessary information required to establish a Network



necessary

1) Name of the computer & IP address
2) port no.

protocol → rules

Socket Module

Networking

Socket

TCP [Transmission control protocol]
→ http [80]
→ https [443]
→ urllib

UDP [User datagram protocol]

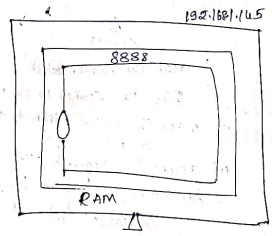
ip addresses
IPv4 IPv6
(internet protocol version)

NAP to create a socket and binding the socket

```
import socket
if __name__ == '__main__':
    my_soch = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print('socket got created successfully')
    ip_addr = '192.168.1.145'  # ip_addr = 'localhost'
    port_no = 8989
    address = (ip_addr, port_no)
    my_soch.bind(address)
    print('socket got binded to the ip address : ', ip_addr,
          'running on the port no : ', port_no)
    print('The port no used : ', port_no)
    print('The ip address which is used is : ', ip_addr)
```

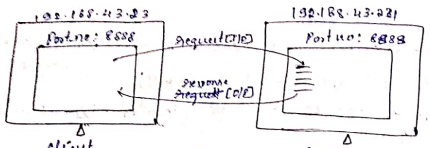
O/P

Socket got created successfully
socket got binded to the ip address : 192.168.1.145 running on the port no : 8989
The port no used : 8989
The ip address which is used is : 192.168.1.145



create socket object
Socket.Socket(Socket.
ipaddr ← AF_INET, (IPv4) Socket.SOCK_STREAM)
(protocol) TCP

client - server Architecture



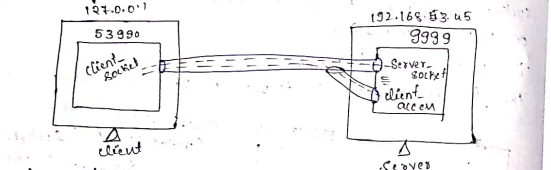
[needs the information]

1. create the socket
2. connect the socket
3. receive the information

[provides the information]

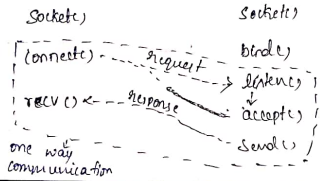
1. Create the socket
2. Bind the socket
3. make the socket to listen for request
4. Send the information

One-way Communication



1) create the socket
2) connect the socket
3) receive the response

1) create the server socket
2) Bind the socket to the process
3) make the socket listen to the request
4) accept the request (I/P)
5) Respond to the request by sending response (O/P)



Server program

```
import socket
if __name__ == '__main__':
    try:
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print('Socket got created at the server end')
        ip_addr = 'localhost'
        port_no = 8888
        address = (ip_addr, port_no)
        server_socket.bind(address)
        print('server socket got binded to {ip_addr} running on
              {port_no}')

        backlog = 5
        server_socket.listen(backlog)
        print('The server socket in ready to listen for backlog
              request')
        client_accept, (ip_addr, port_no) = server_socket.accept()
        print('server socket got connected to client whose
              ip addr {ip_addr} running on {port_no}')
        data = 'hey client ... tq for connecting'
        client_accept.send(data.encode('utf-8'))

    except socket.error as msg:
        print('The cause of the exception is : {msg}')

    finally:
        client_accept.
        Server_socket.close()
        print('socket got closed at the server end')
```

O/P

```
socket got created at the server end
Server socket got binded to localhost running on 8888
Server socket is ready to listen for 5 requests
Server socket got connected to client socket whose
ip address 127.0.0.1 running on 63108
Socket got closed at server end..
```

client program:

```
import socket:
if __name__=='__main__':
    try:
        client_socket = socket.socket(socket.AF_INTE, socket.
                                      SOCK_STREAM)
        print('socket got created at the client')
        ip_addr = 'localhost'
        port_no = 8888
        address = (ip_addr, port_no)
        client_socket.connect(address)
        print('the client socket got connected to the server
              socket')
        buf_size = 1024
        data = client_socket.recv(buf_size).decode('utf-8')
        print('The information received from the server:
              {data}')
    except socket.gaierror as msg:
        print('The information cause of the exception is:
              {msg}')
    Except socket.error as msgg:
        print('The cause of the exception is: {msgg}')
    finally:
        print('the socket got closed at the client end')
```

O/P

socket got created at client end
the client socket got connected to the server socket
the information received from the server : hey client--
tq for connecting.
the socket got closed at the client end...

Note:-

socket() :- this function is used to create the socket

bind() :- this function is used to bind the socket to the process
→ while binding socket to the process two information has to be provided ip-addr & port no.

connect() :- this function is used to bind the socket to the process & it will establish the connection from one computer to another computer.

listen() :- this function is used to make the socket to listen to for the request sent from the client.

accept() :- this function is used to create the socket internally the socket binded to the process
* basically this function is used to send the response to the client.
* the information of client ipaddress and portno will be stored within this function.

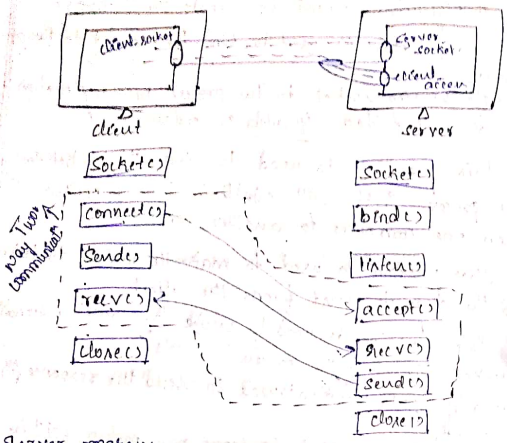
recieve(): this function is used to receive the information
→ while receiving the information should be decoded using UTF-8 format.

Send() :- this function is used to send the information.
→ while sending the information it has to be encoded using UTF-8 format.

close() :- this function is used to close the socket

gaierror →

Two-way Communication :-



Server program

```
import socket
if __name__ == '__main__':
    try:
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print ('server socket got created')

        ip_addr = 'localhost'
        port_no = 8989
        address = (ip_addr, port_no)
        server_socket.bind (address)
        print ('binding')
        backlog = 5
        server_socket.listen(backlog)
        print ('ready to listen')

        client_accept, (ip_addr, port_no) = server_socket.accept()
        print ('server socket got connected to client whose ip addr
            {ip_addr} running on {port_no}')
        buf_size = 1024
        data = client_accept.recv(bufsize).decode('utf-8')
        print (f'The msg recieved from the client: {data}') hi
        data = input ('enter the msg to client') hlo
        client_accept.send(data.encode('utf-8'))
    except socket.error as msg:
        print (f'The cause of the exception: {msg}')
    finally:
        client_accept.close()
        print ('socket got closed')
```

client program

```
import socket
if __name__ == '__main__':
    try:
        socket_client_socket = socket.socket(socket.AF_INET,
                                             socket.SOCK_STREAM)
        print ('socket got connected')
            created
        ip_addr = 'localhost'
        port_no = 8989
        address = (ip_addr, port_no)
        client_socket.connect (address)
        print ('connecting')
        data = input ('enter a msg to server') hi
        client_socket.send (data.encode('utf-8'))
        buf_size = 1024
        data = client_socket.recv(buf_size).decode('utf-8')
        print (f'The msg recieved from the client: {data}') hlo
    except socket.gaierror as msg:
        print (f'The cause for the exception is: {msg}')
    except socket.error as msg:
        print (f'The cause for the exception is: {msg}')
    finally:
        print (client_socket.close())
            print ('socket got closed')
```

Multi-way Communication:-

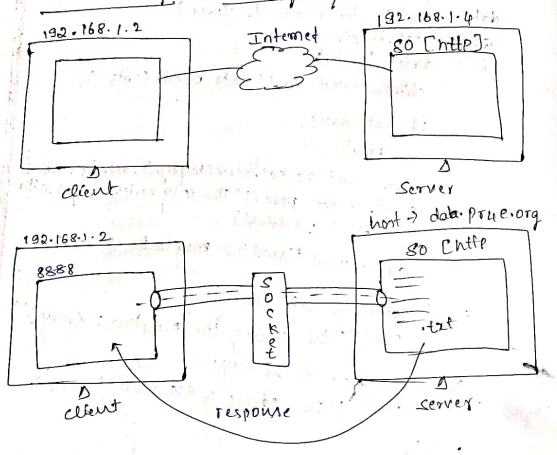
Server

```
import socket
if __name__ == '__main__':
    try:
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print('The socket got ~~connected~~ created')
        ip_addr = 'localhost'
        port_no = 8989
        address = (ip_addr, port_no)
        server_socket.bind(address)
        backlog = 5
        server_socket.listen(backlog)
        client_access, (ip_addr, port_no) = server_socket.accept()
        buf size = 1024
        while True:
            data = client_access.recv(bufsize).decode('utf-8')
            if not data:
                break
            print(f'The msg received from client :{data}')
            data = input('enter a msg to client')
            client_access.send(data.encode('utf-8'))
    except socket.error as msg:
        print(f'The cause of the exception is : {msg}')
    finally:
        client_access.close()
        print('socket got closed')
```

Client

```
import socket
if __name__ == '__main__':
    try:
        c_socket = socket.socket(Socket.AF_INET, socket.SOCK_STREAM)
        print('got created')
        ip_addr = 'localhost'
        port_no = 8989
        address = (ip_addr, port_no)
        client_socket.connect(address)
        print('got connected')
        data = input('enter a msg to server')
        while data != 'quit':
            client_socket.send(data.encode('utf-8'))
            if not data:
                break
            data = client_socket.recv(buffsize).decode('utf-8')
            print('The msg received from the client :', data)
            data = input('send the msg to the server!!')
    except socket.gaierror as msg:
        print(f'The cause of the exception : {msg}')
    except socket.error as msg:
        print(f'The cause of exception : {msg}')
    finally:
        client_socket.close()
        print('socket got closed')
```

Establishing networking v hypertext transfer protocol (http) through



Note:-

* In below example using http we have performed web scraping that is romeo.txt is fetched from the server.
* while making a web request if you want to get the information 'GET' upload the information on the server 'post' statement.
* while sending the information to the server we have to specify the complete path of the file. we have to tell server type (HTTP). and no. of files (1.0)

\r → return carriage

whenever we perform webscraping using HTTP along with the desired o/p the complete server details get executed in order to avoid this we have to use urllib module.

```
import socket *
if __name__ == '__main__':
    try:
        client_socket = socket.socket(socket.AF_INET, socket.
                                      SOCK_STREAM)
```

Urllib

Using built in funct" called as urllib.request" the type of webrequest will be decided as well as the connection will be established from client to server. Then it will return the o/p in the form of an object

* 'strip()' is used to eliminate white space

Wapython program to perform webscraping using urllib

```
import urllib.request
if __name__ == '__main__':
    url = 'http://data.pr4e.org/romeo.txt'
    data = urllib.request.urlopen(url)
    print(data)
    print(type(data))
    for i in data:
        print(i.decode().strip())
```

o/p

wap prgm to perform webscraping to count the occurrance of the words present within the data & store the result in the form of dictionary

```
import urllib.request
if __name__ == '__main__':
    url = 'http://data.pr4e.org/romeo.txt'
    data = urllib.request.urlopen(url)
    print td,
    for i in data:
        k = i.decode().split()
        print(k)
        for j in k:
            count[j] = count.get(j,0)+1
    print(count)
```

wap prgm to perform web-scraping on the binary file

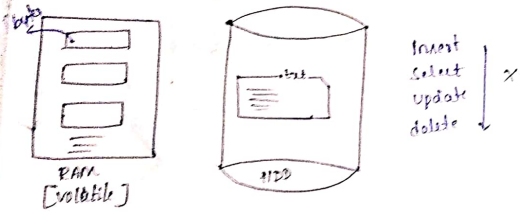
```
import urllib.request
if __name__ == '__main__':
    url = 'http://data.pr4e.org/cover.jpg'
    data = urllib.request.urlopen(url).read()
    my_file = open('new.jpg', mode = 'wb')
    new_data = my_file.write(data)
    print(data)
    print('image got scrapped!!')
```

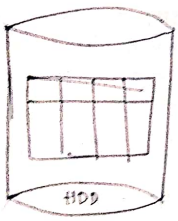
PYTHON TO DATABASE CONNECTIVITY (PDBC)

Database

In order to store huge amount of data nowadays they make use of database bcz to perform the operations like inserting the data, selection of data, update & deletion of data can be done in a simplest manner. by using some database vendors.

Initially they used to store the data in text file but it is very difficult to manage the data so ~~they~~ data will be stored in the form of table (rows & column) within the database





Database vendors

- oracle
- MySql
- Sybase
- mongoDB
- microsoft

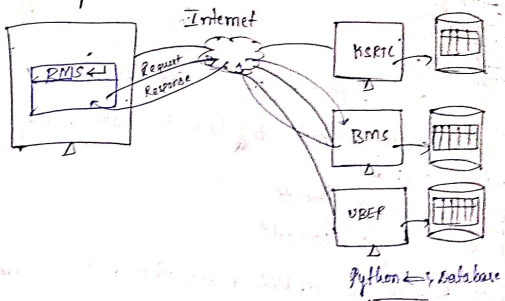
insert ⟷ CREATE
select ⟷ READ
update ⟷ UPDATE
delete ⟷ DELETE

"CRUD" operation

If the data is been stored in the database in order to perform some operation on data we have to make use of some sql query

if the application ass developed using programming language like java, python then the programming language should be able to connect the database

how some of the operation on database.
Pgm
sql query to perform crud operation.

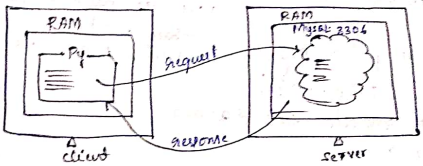


Insert ⟷ Create → insert into tablename(columnname) value(--)
Select ⟷ Read → select * from tablename
Update ⟷ update → update tablename set columnname = 'value' where columnname = 'value'
delete ⟷ delete → delete tablename where columnname = '--'

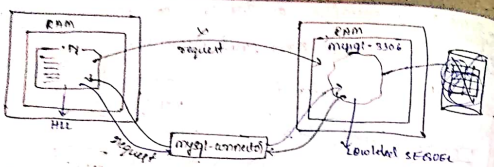
Necessary information required to connect python program with the database.

* In order to establish the connection b/w the python pgm & database we have to make use of the mediator mysql-connector.

* while connecting python pgm to database certain protocol has to be provided shown below



ex:-
host = 'localhost'
port = "3306"
username = 'root'
password = 'root123'
database = 'tutmeta base'



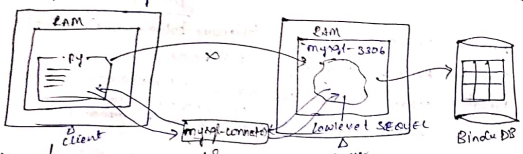
to install mysql-connector we have to make use of following command

→ pip install mysql-connector

to check installed or not

⇒ pip show mysql-connector

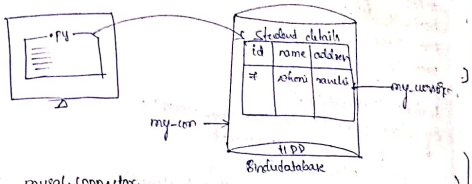
Wapython program to establish a connection with the database




```
import mysql.connector
if __name__ == '__main__'
    try:
        host = 'localhost'
        port = 3306
        user = 'root'
        password = 'root123'
        database = 'bindudatabase'
        my_con = mysql.connector.connect(host=host, port=port
                 user=user, password=password,
                 database=database)
        print(f'the python prgm got connected to database
              :{database}')
except mysql.connector.Error as msg:
    print(msg)
finally:
    my_con.close()
```

1) CRUD operations:- (C) Create operation

Ex:- wapython program to establish a connection with database and insert the data within a table




```
import mysql.connector
host = 'localhost'
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def close_connection(my_con, my_cursor):
    if my_con.is_connected():
        my_cursor.close()
        print('cursor object got closed')
        my_con.close()
        print('connection got closed from database')

def insert_records():
    try:
        my_con = mysql.connector.connect(host=host, port=port, user
                 =user, password=password, database=database)
        print(f'python program got connected to database')
        sql_insert_query = """insert into 'studentdetails' ('id','name',
                           'address') values (81,'dhoni','ranchi')"""
        my_cursor = my_con.cursor()
        print('cursor object got created')
        my_cursor.execute(sql_insert_query)
        my_con.commit() → the changes has to be displayed on front
        print('changes has been updated within the database')
    except mysql.connector.Error as msg:
        print(f'the cause of the exception is:{msg}')
    except e
    finally:
        close_connection(my_con, my_cursor)
if __name__=='__main__':
    insert_records()
```

Ex2:- wapp to establish the connection with DB & insert multiple data into the dB (table)

```
import mysql.connector.
host = 'localhost'
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def close_connection (my_con, my_cursor):
    if my_con.is_connected():
        my_cursor.close()
        print ('cursor object got closed')
        my_con.close()
        print ('connection got closed from database')
def insert_multiple_records():
    try:
        my_con = mysql.connector.connect(host=host, port=port, user=user,
                                         password=password, database=database)
        print (f'python program got connected to database :{database}')
        sql_insert_query = """Insert into 'studentdetails'('id','name','address')
                              Values (%s,%s,%s)"""
        my_cursor = my_con.cursor()
        print ('the cursor object got created')
        records_data = [(18,'Bindu','chitradurga'),(17,'chaithra','BTM'),
                        (16,'Vandu','Kondaballi')]
        my_cursor.executemany(sql_insert_records, records_data)
        print ('sql query got inserted into table: studentdetails')
        my_con.commit()
        print ('changes has been updated within the database')
    Except mysql.connector.Error as msg:
        print (f'the cause of the exception is :{msg}')
    finally:
        close_connection (my_con, my_cursor)
if __name__=='__main__':
    insert_multiple_record()
```

2) Read operation

Ex3:- wapp to establish the connection with dB & fetch the single data from the table.

```
import mysql.connector
host = 'localhost'
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def close_connection (mycon, my_cursor):
    my_cursor.close()
    print ('The cursor object got closed')
    my_con.close()
    print ('The connection got closed from database')
def select_records:
    try:
        my_con = mysql.connector.connect(host=host, port=port, user=user,
                                         password=password, database=database)
        print (f'The connected to the database: {database}')
        Sql_select_query = """select * from studentdetails"""
        my_cursor = my_con.cursor()
        print ('cursor object got created')
        my_cursor.execute (sql_select_query)
        record = my_cursor.fetchone()
        print (record)
        print (f'id:{record[0]}')
        print (f'name:{record[1]}')
        print (f'address:{record[2]}')
    Except mysql.connector.Error as msg:
        print (f'The cause of the exception is:{msg}')
    finally:
        close_connection (mycon, my_cursor)
if __name__=='__main__':
    select_record()
```

Ex 28: WAP to establish the connection with the DB & fetch all the data present within the table.

```
import mysql.connector
host = 'localhost'
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def close_connection(my_con, my_cursor):
    my_cursor.closed()
    print ("The cursor object got closed")
    my_con.close()
    print ('The connection got closed')
def select_records():
    try:
        my_con = mysql.connector.connect(port=port, host=host, user=user,
                                         password=password,
                                         database, database)
        print (" connection established")
        my_cursor = mysql.my_con.cursor()
        print (" cursor object got created")
        select_sql_select_query="""select * from 'studentdetail'"""
        my_cursor.execute(sql_select_query)
        for my_con.commit()
        records_data = my_cursor.fetchall()
        print (records_data)
        for record in records_data:
            print (f' id : {record[0]}')
            print (f' name: {record[1]}')
            print (f'address: {record[2]}')
    except mysql.connector.query as msg:
        print (f'The cause of the exception is :{msg}')
    finally:
        my_close_connection (myconn, my_cursor)
if __name__ == '__main__':
    select_records()
```

Update operation

Ex 29: WAP to establish the connection with the DB & update the existing data

```
import mysql.connector
host = 'localhost'
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def close_connection(my_con, my_cursor):
    my_con.close()
    print ('The connection got closed')
    my_cursor.close()
    print ('The cursor object got closed')
def update_query:
    try: my_con = mysql.connector.connect(host=host, port=port, user=user,
                                          database=database, password=password)
        update my_cursor = my_con.cursor()
        sql_update_query = """update 'studentdetail' name='virat' where id=18"""
        my_cursor.execute(sql_update_query)
        my_con.commit()
        sql_select_query = ""select * from 'studentdetail' where id=18"""
        my_cursor.execute()
        records = my_cursor.fetchall()
        print (records)
    except mysql.connector.Error as msg:
        print (f'the cause of the exception is {msg}')
    finally:
        print (try) close_connection (my_con, my_cursor)
if __name__ == '__main__':
    update_query()
```

4) Delete operation

Ex:- Wap to establish the connection with the db & Delete the ~~data~~ complete details present with in the row.

```
import mysql.connector
host = 'localhost'
port = 3306
user = 'root'
password: 'root123'
database = 'bindudatabase'
def close_connection (my_con, mycursor):
    my_cursor.close()
    my_con.close()
def delete_query():
    try:
        my_con = mysql.connector.connect (host=host, port=port, user=user, database=database, password=password)
        print ('connection established')
        my_cursor = my_con.cursor() #
        sql_delete_query = """delete from `studentdetails` where id = 15"""
        my_cursor.execute()
    my_con.commit()
        sql_select_query = """select * from studentdetails"""
        my_cursor.execute()
        records = my_cursor.fetchall()
        print (records)
    except mysql.connector.connect as msg:
        print ('the cause of the exception is :', msg)
    finally:
        close_connection (my_con, my_cursor)
if __name__ == '__main__':
    delete_query()
```

Note:- * inorder to establish the connection from python to db we have to provide 5 details i.e., ipaddress, port, username, password, database name

* connect() :- this function is used to establish the connection from python to database.
* cursor() :- this function is used to create the cursor to interact with the table. to perform exicd operation wehav -e to use reference which is pointing to the table
* execute() :- this function is used to execute the query
* if the multiple datas has to be inserted then we have to be use executemany().
* commit() :- If the changes made with in the database has to be reflected in the table then we should use commit() .
* is_connected() :- this function is used to check whether the connection exist b/w python to database.
* fetchone() :- It is used to fetch the data present within single row.
* fetchall() :- It is used to fetch the data present within the table. (complete rows).

Dynamic operation → 'input taken from user'

Ex:- write a python prgm to establish the connection with the database and create a column without the table where id should be auto implemented for the remaining 3 columns user should provide the data dynamically through keyboard

```
import mysql.connector
host = 'localhost'
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def close_connection(my_con, my_cursor):
    my_cursor.close()
    my_con.close()
def insert_dynamic_record():
    try:
        my_con = mysql.connector.connect(port=port, host=host, user=user
                                         password=password, database=
                                         database)
        my_cursor = my_con.cursor()
        select_query = """insert into employee_student details (name, age, address)
                          values (%s, %s, %s)"""
        ~~my_cursor.execute(select_query)~~
        name = input ('enter the name of the employee')
        age = int(input ('enter the age of the employee'))
        address = input ('enter the address of the employee')
        record = (name, age, address)
        my_cursor.execute (select_query, record)
        my_con.commit()
    except mysql.connector.~~connect~~Error as msg:
        print (f'the exception is: {msg}')
    finally:
        close_connection (my_con, my_cursor)
if __name__ == '__main__':
    insert_dynamic_record()
```

Ex:- inserting multiple data.

```
import mysql.connector.~~connect~~
host = 'host'
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def close_connection(my_con, my_cursor):
    my_cursor.close()
    my_con.close()
def insert_dynamic_record():
    try:
        my_con = mysql.connector.connect(host=host, port=port, user=us
                                         -er, database=database, passwo
                                         =password)
        my_cursor = my_con.cursor()
        sql_insert_query = """insert into employee_student details (name, age addr
                             -ess) value (%s, %s, %s)"""
        num = int(input ('enter the no. of employees'))
        for i in range (num):
            name = input('enter the employee name')
            age = int (input('enter the employee age'))
            address = input ('enter the employee address')
            records = ~~my_cursor~~(name, age, address)
        my_~~execute~~cursor.execute ()
        my_con.commit()
    except mysql.connector.Error as msg:
        print (f'the cause of the exception: {msg}')
    finally:
        close_connection (my_con, my_cursor)
if __name__ == '__main__':
    insert_dynamic_record()
```

3) Wap to establish the connection with the database & fetch the complete data present within a row by taking the id from the user.

```
import mysql.connector
host = "localhost"
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def close_connection(my_con, my_cursor):
    my_cursor.close()
    my_con.close()
def select_record():
    try:
        my_con = mysql.connector.connect(host=host, port=port, user=user, password=password, database=database)
        my_cursor = my_con.cursor()
        sql_select_query = """ select * from `studentdetails` """
        my_cursor.execute(sql_select_query)
        records = my_cursor.fetchall()
        columnname = input("enter the columnname:")
        if columnname.lower() == 'id':
            for record in records:
                print(f'id:{record[0]}')
                print()
        elif column.lower() == 'name':
            for record in records:
                print(f'name:{record[1]}')
                print()
        elif column.lower() == 'address':
            for record in records:
                print(f'address:{record[2]}')
                print()
        else:
            print('such column name is not present')
    except mysql.connector.Error as msg:
        print(f'The cause of the exception is:{msg}')
    finally:
        close_connection(my_con, my_cursor)
if __name__ == '__main__':
    select_record()
```

4) Wap to establish the connection with the database & update the complete data present within a row by taking the id from user

```
import mysql.connector
host = 'localhost'
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def close_connection(my_con, my_cursor):
    my_cursor.close()
    my_con.close()
def update_record():
    try:
        my_con = mysql.connector.connect(host=host, port=port, user=user, Password=password, database=database)
        my_cursor = my_con.cursor()
        sql_update_query = """ update `studentdetails` set `name` = %s where `address` = %s """
        name = input('enter the name:')
        address = input('enter the address:')
        records = (name, address)
        my_cursor.execute(sql_update_query, records)
        my_con.commit()
        print('new data got updated')
        print()
        sql_select_query = """ select * from `studentdetails` where `id` = %s """
        id = int(input('enter the id:'))
        record = (id,)
        my_cursor.execute(sql_select_query, record)
        records = my_cursor.fetchone()
        print(f'the id:{record[0]}')
        print(f'name:{record[1]}')
        print(f'address:{record[2]}')
    except:
        print
    finally:
if __name__ == '__main__':
    update_record()
```

5) wapp to establish the connection with the database & delete the row present within a tbl by taking i/p from the user

```
import mysql.connector
host = 'localhost'
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def close_connection(my_con, my_cursor)
    my_cursor.closed()
    my_con.close()
def delete_record():
    try:
        my_con = mysql.connector.connect(---)
        my_cursor = my_con.cursor()
        sql_delete_query = """delete from `studentdetails` where `address`=%s"""
        address = input("Enter the address")
        record = (address,)
        my_cursor.execute(sql_delete_query, record)
        my_con.delete commit()
        sql_select_query = """select * from `studentdetails`""" where
        my_cursor.execute(sql_select_query)
        records = my_cursor.fetchall()
        print(records)
    except mysql.connector.Error as msg:
        print(---)
    finally:
        my close_connection(my_con, my_cursor)
if __name__ == '__main__':
    delete_record()
```

6) wap p to establish the connection with the database & perform all the CRUD operation.

Transaction

Note:- Transaction in the std. way of writing the code to achieve ACID property

* Atlast transaction code helper() will be used to apply changes within the database (commit()) as well as it is used to restrict the changes (roll back)

ACID Property
A - Atomicity
C - consistency
I - isolation
D - Durability

```
import mysql.connector
host = 'localhost'
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def helper_functions():
    data = input("enter ok to make the changes with in the database")
    if data.lower() == 'ok':
        return True
    else:
        return False
def show_details(my_con, department):
    try:
        my_cursor = my_con.cursor()
        print("sql_select_query = "select * from 'cricketer' where department = %s")
        record_data = (department,)
        my_cursor.execute(sql_select_query, record_data)
        records = my_cursor.fetchall()
        print(records)
    except mysql.connector.Error as msg:
        print("The cause of the exception is :", msg)
    finally:
        if my_con.is_connected():
            my_cursor.close()
def start_transaction():
    try:
        my_con = mysql.connector.connect(---)
        show_details(my_con, department = 'batting')
        show_details(my_con, department = 'bowling')
        sql_delete_query = """delete from 'cricketer' where 'department' = 'batting'"""
        sql_update_query = """update 'cricketer' set 'salary' = 100000 where 'department'
                              = 'bowling' """
        my_cursor = my_con.cursor()
        my_cursor.execute(sql_delete_query)
        my_cursor.execute(sql_updat_query)
        OK = helper_function
        if "OK":
            my_con.commit()
        else:
            my_con.rollback()
```

Except:-

```
finally:
    if my_con.isconnected():
        my_cursor.close()
    my_con.close()
```

showdetails(my_con, department = 'batting')
showdetails(my_con, department = 'bowling')

ACID property

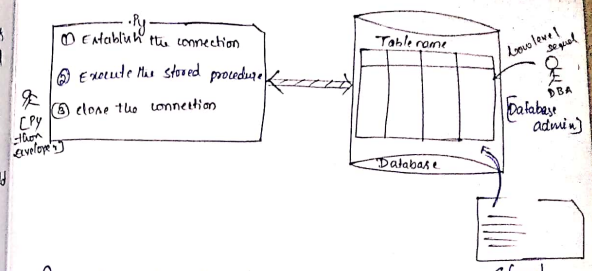
Automicity :- It is the property where the transaction should complete if it should not be completed. (100% or 0%) → commit / rollback

Consistency :- A transaction must start in consisting state and leave the s/m in the consistent state. (Go the same flow)

Isolation :- Intermediate result of the transaction should not be visible to the current transaction.

Durability :- once the transaction was committed then the effect should be constant even after a s/m failure.

Stored Procedure :-



General syntax for stored procedure

```
Create PROCEDURE 'stored procedure name' (IN_ , OUT_)
Begin
    } logic of stored procedure.
End
```

Ex:- Write a python program to establish the connection with the data-base count the no. of employees present within the respective dept. using the table given below.

code for stored procedure

```
Create procedure 'count_of_employee' (in The_department varchar(15),
                                      out The_count int)
Begin
    select count(*) into the_count from Employee table where 'department' = The_department
End
```

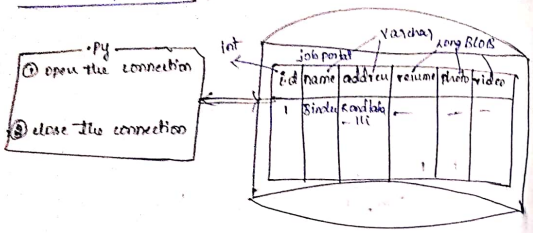
Ex:- write a python program to establish a connection with a database fetch the data present with in a table by providing department name as input stored procedure.

```
CREATE PROCEDURE 'display_info' ( in_dept_name varchar(20) )
BEGIN
    select * from 'employee' where 'department' = dept_name;
END
```

Note:- Stored procedure is the process by writing complex Sql queries by Database admin.

* after preparing stored procedure DBA will execute and later he will provide the name of stored procedure to the developer.
* callproc() in need to execute stored procedure present with in the database.
* if the data in been fetched from the database using stored procedure in order to retrieve the data we have to use stored_results()

BLOB(b) operation :-



Ex:- wap.p. to establish connection with a db and insert binary files into the database.

```
import mysql.connector
host = 'local host'
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def close_connection(my_con, my_cursor):
    my_con.close()
    my_cursor.close()
def convert_to_binary(file):
    with open(file=file, mode='rb') as my_file:
        binary = my_file.read()
    return binary
def insert_record():
    try:
        my_con = mysql.connector.connect(host=host, port=port, user=user,
                                         password=password, database=database)
        my_cursor=my_con.cursor()
        sql_insert_query = """insert into 'jobportal' ('name', 'address', 'resume',
                              'photo', 'video')
                              value (%s, %s, %s, %s, %s)"""
        name = input("enter the candidate name:")
        address = input("enter the address")
        path_resume = 'path'  path_photo='path', path_video='path'
        resume = convert_to_binary(path_resume)
        photo = convert_to_binary(path_photo)
        video = convert_to_binary(path_video)
        record_data = (name, address, resume, photo, video)
        my_cursor.execute(sql_insert_query, record_data)
        my_con.commit()
        print("data got inserted on to the database")
    except mysql.connector.Error as msg:
        print(f"the cause of the exception is :{msg}")
    finally:
        close_connection(my_con, my_cursor)
if __name__=='__main__':
    insert_record()
```

Ex:- wa python pgm to establish the connection b/w the database to fetch binary data (pdf, jpg, mp4) present with in the database

```
import mysql.connector
host = 'localhost'
port = 3306
user = 'root'
password = 'root123'
database = 'bindudatabase'
def close_connection(my_con, my_cursor):
    my_cursor.close()
    my_con.close()
def convert_from_binary(file, data):
    with open(file=file, mode='wb') as my_file:
        my_file.write(data)
def select_records():
    try:
        my_con = mysql.connector.connect(host=host, port=port, user=user,
                                         password=password, database=database)
        my_cursor=my_con.cursor()
        sql_select_query = """select * from `jobportal` where `id`=%s"""
        id=int(input("enter the id :"))
        record_data = (id,)
        my_cursor.execute(sql_select_query, record_data)
        record = my_cursor.fetchone()
        print(f"id:{record[0]}")
        print(f"address:{record[1]}")
        print(f"address:{record[2]}")
        photo_binary = record[3]
        resume_binary = record[4]
```

video_binary = Record[5]

new_file_resume = 'path' convertfrom_binary(new_file_resume, fin_resume_binary)

new_file_photo = 'path' convert_from_binary(new_file_photo, fresphoto_binary)

new_file_video = 'path' convert_from_binary(new_file.video, video_binary)

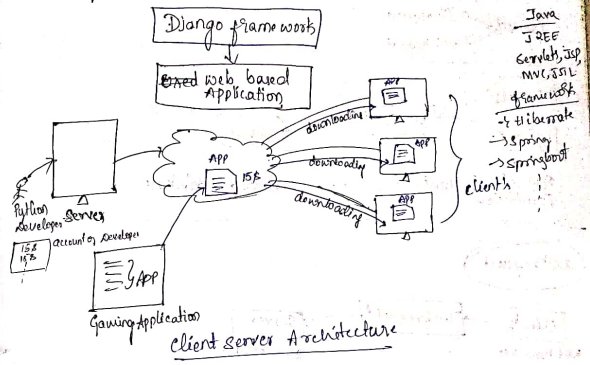
Exception:-

NOTE:- BLOB stands for binary large object

- while performing BLOB operation while inserting binary file into the database it should be converted into binary format (bytes)
- 'CLOB' stands for collecting the large object from the database
- while performing clock operation the data that is fetched will be available in the form of 'bytes'. so it should be converted into normal format.

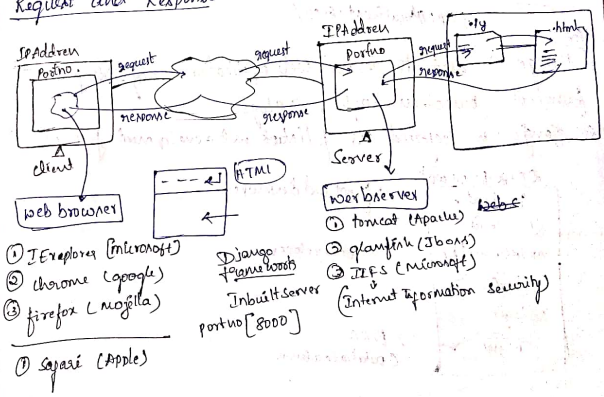
DJANGO

basically it is a framework used to develop web based applications



Client Server Architecture

Request and Response

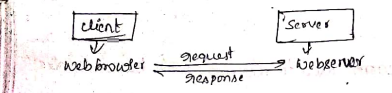


① IExplorer (Microsoft)
② chrome (google)
③ firefox (mozilla)

① Safari (Apple)

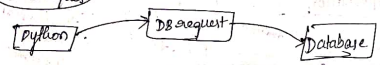
Django framework
Inbuilt server
portno [8000]

① tomcat (Apache)
② glassfish (Jboss)
③ IIFS (Microsoft)
(Internet Information Security)



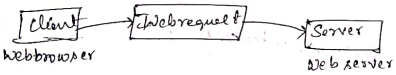
Request → DB request
→ web request

DB request



Protocols :- host, port, user, password, database
(credentials)

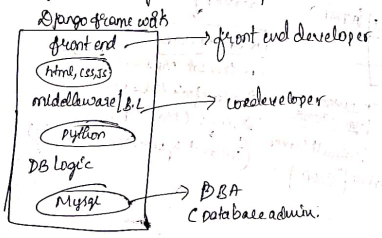
Web request



Protocols :- webservice name, [paddress, portno, pgname, name
http://localhost:8000/

Note :-

* Django is used to develop web application using this framework frontend, business logic, DB logic.
* So django is referred as full stack web development framework.



comments that are used in Django framework

1) Inorder to install Django we have to use following command

pip install Django == 3.0.2 (a)
pip3 install Django == 3.0.2 (b)
python -m pip install Django == 3.0.2

2) Inorder to check Django is installed or not we have to use following command

pip show Django

3) in order to create project following command is used

django-admin startproject first project

4) Command used to create the App

Python manage.py startapp Application name (first app)

5) Command used to run the server

Python manage.py runserver

dir
C:\\=>E:

→ CD Django project
(to folder)

C:\\=> E:

E:\$ cd programming language

E:\programming language > CD\ Django project

E:\programming language\ django project > django-admin startproject first project

E:\Programming language\ django project\ cd first project

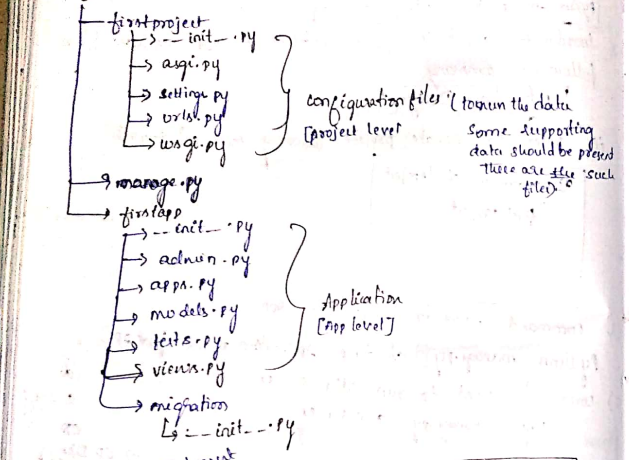
=> first project → edit
→ python manage.py startapp first app
→ python manage.py runserver

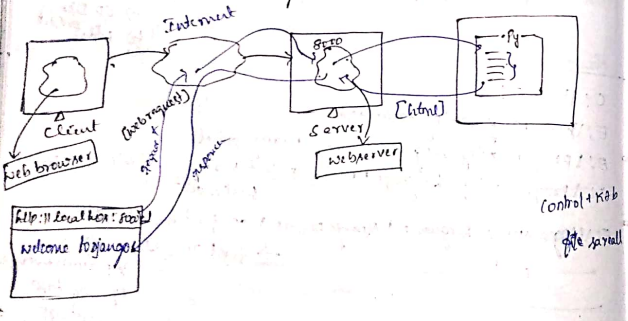
copy localhost:8000 → paste chrome

Sample project:-

1) using Django framework display a msg for the client (response) if there is any request from the client

firstproject (

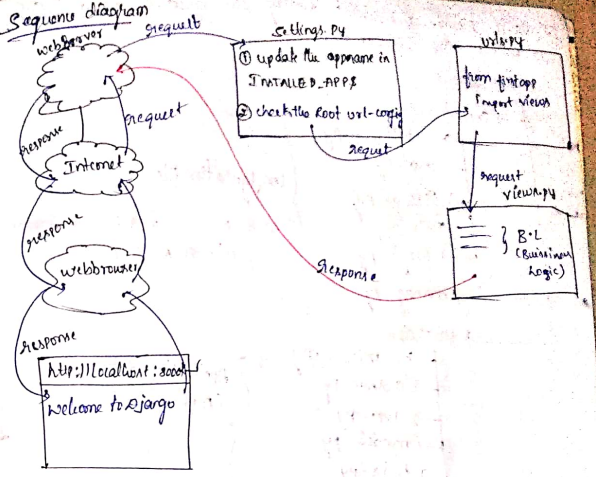




(control + Kab

file saved

Sequence diagram



settings.py

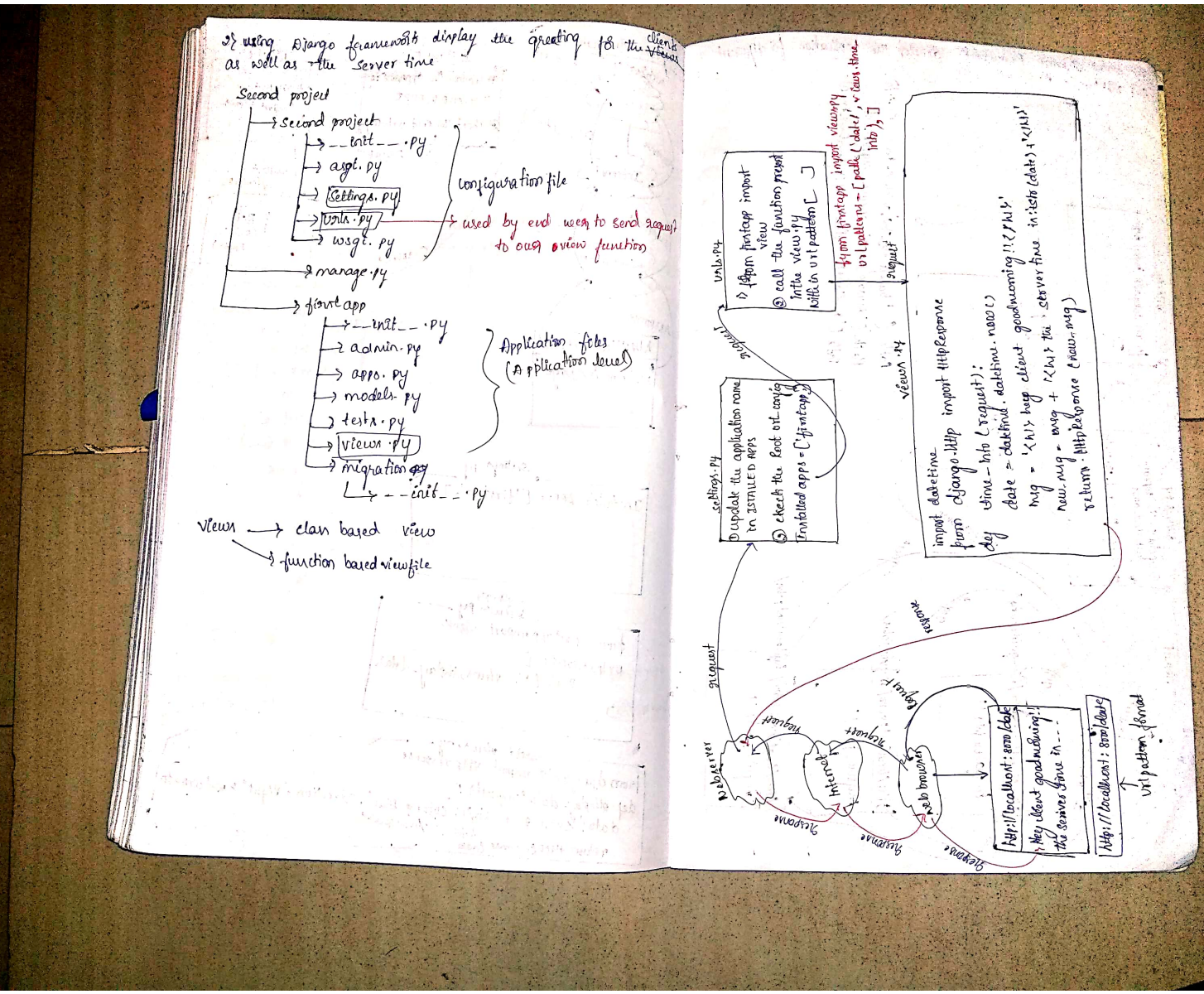
```
Installed APPs = ['firstapp',]
```

urls.py
~~views.py~~

```
from firstapp import views
urlpatterns = [
    Path ('', views.display_data),
]
```

Setti views.py

```
from django.http import Http response
def display_data(request):
    data:'<marquee style; color = Blue', Direction = "Right" > welcome to
                Django</marquee>'
    return HttpResponse(data
```



3) using Django Framework develop the application for different url pattern

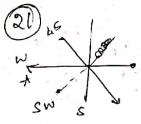
4) using Django framework display the greeting msg to users based upon the server time & display the server time for the client using the following details.

10AM - 12pm → Goodmorning
12pm - 4pm → Good afternoon
4pm - 8pm → good evening
8pm - 12am → good night

Note:- Inorder to fetch the hour we have to pass the format Specifier '%H' as an argument to 'strftime()'

Ex:- using Django framework create multiple application with in a single project

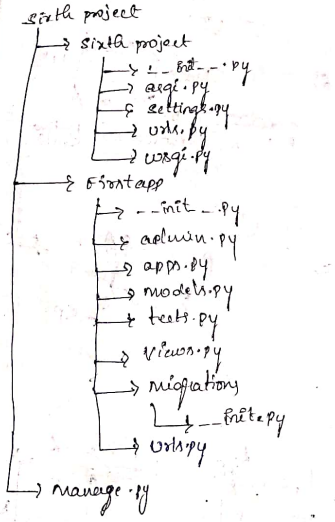
21

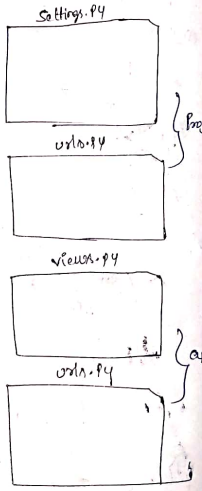


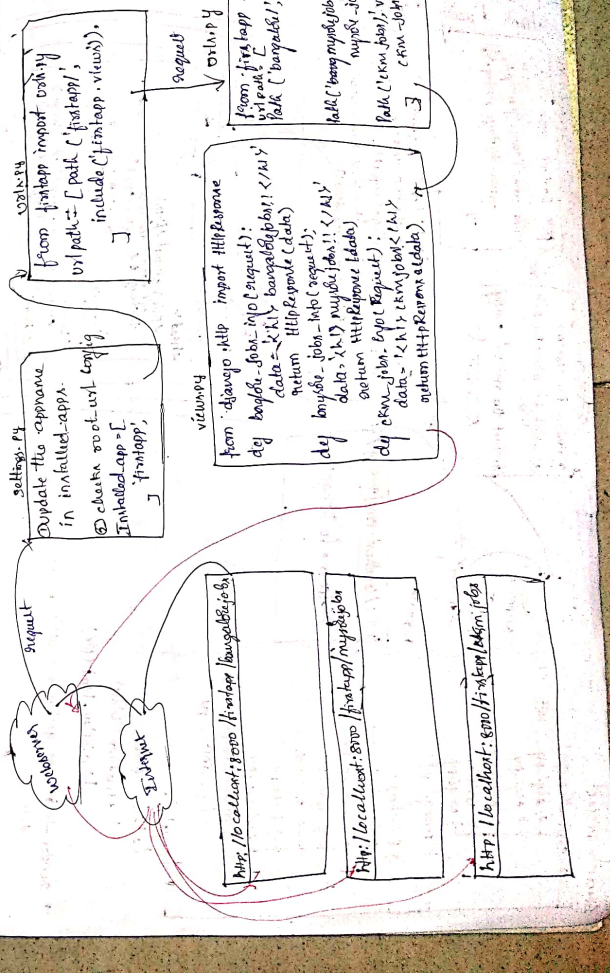
SW southwest

8) using Django framework define the url patterns at the application level

sixth project







Advantage of creating url patterns in application level

1) modularity can be achieved as well as code reusability will be there.
2) It promote the reuseability of code
3) provides the clarity about the url patterns.

Note:- If a seperate urls file is created at the app level then, with in the project level we have to make use of 'include()'

Template (we should html)

4) using Django projects display greetings message for the client using Templates

* render → means getting a response or O/P from the html file

Settings.py

```
INSTALLED_APPS = ['firstapp',]
BASE_DIR =
print (BASE_DIR)
TEMPLATE_DIR = os.path.join (BASE_DIR, 'template')
print (TEMPLATE_DIR)
```

8) using Django framework display the servertime for the client using template based application

eighth project

Note:

* from the python file whenever the data to be passed to html it should be passed in the form key: value pairs i.e., dictionary
* In order to collect the data inside the html file we have to use template variable (var)
* Inorder to get template variable file should be open in HTML(Django) mode.

g) using django framework display the servertime for the client using template based application along with static folder. css file

ninth project

MVT

In order to develop web based application using java the following architecture will be used.

MVC
- Models (Business logic)
- views (presentation logic)
- controller (to coordinate)

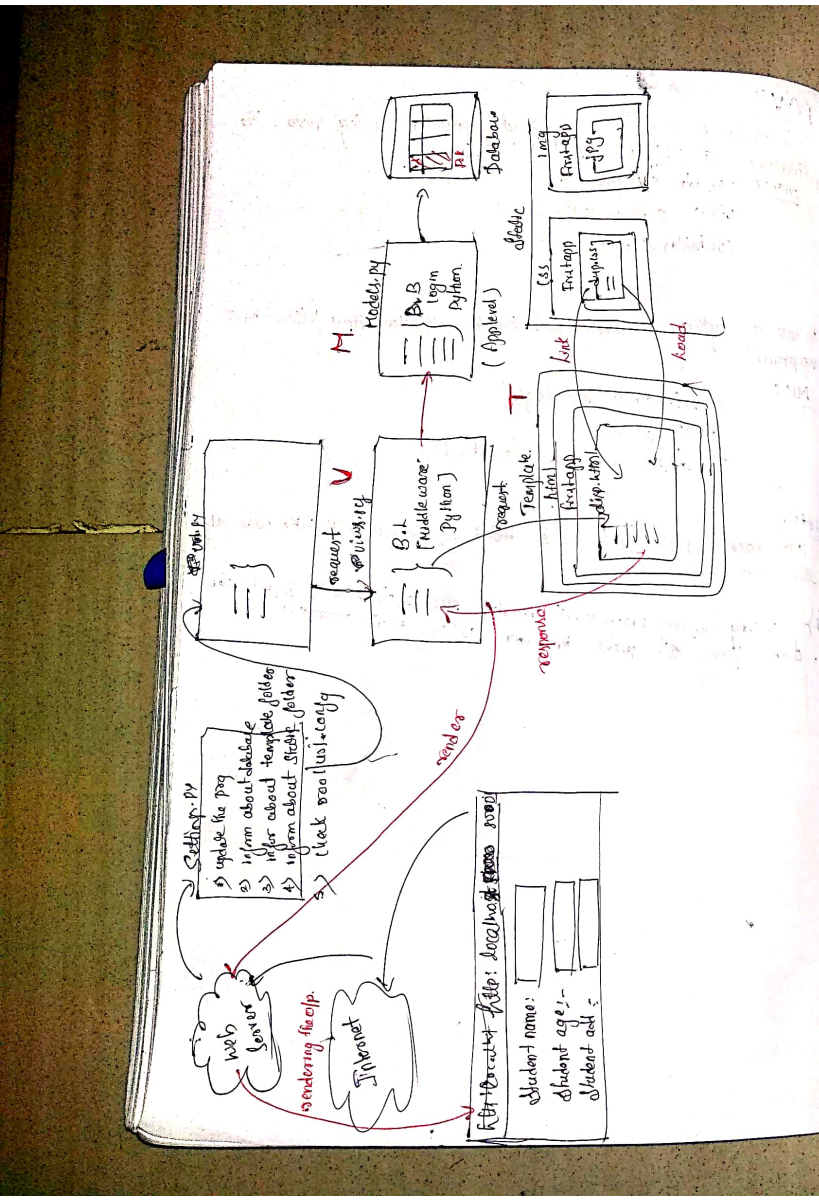
In case of python to develop web based application they follow MVT approach

MVT
- Models (Database logic)
- views (B.L)
- Templates (presentation logic)

Note:

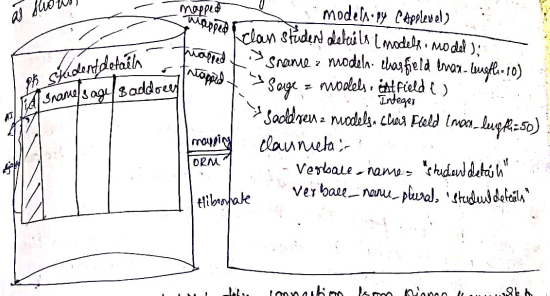
1) In case of python framework (i.e., django will take care all the coordination activity.

2) using django framework develop the application to fetch the data from the web browser using MVT approach.



Object Relational Mapping (ORM)

Internally Django framework will make use of hibernate to perform mapping of Python file (models.py) into database as shown in the below diagram



Note: In order to establish the connection from Django framework to database we have to provide below mentioned protocol

1) NAME:
2) USER
3) PASSWORD
4) HOST:
5) PORT:

- In order to establish the connection from Django framework to database we have to create a class within models.py & class name should be the table name which have to be created within the DB

ORM technique

- every model class would represent database information
- every model class would represent the information of table
- the fields in the model class represent the column name in the table.
- Django framework internally make use of ORM technique (Object Relational mapping) that is as shown in the above diagram
- Commands that are involved in the project:
- Inorder to create a table by referring to the model class we have to use following command

python manage.py makemigrations

make migrations command would not create a table rather it will create a query to create a table

* In order to see table information created in Django Fw (query) we use the following command

```
python manage.py sqlmigrate appname file no
```

* In order to execute prepare query we have to use following command.

```
python manage.py migrate
```

Note:- while creating table internally Django Fw would create default column that is id, which will be treated as primary key as well as auto implemented

* To enter the data from the browser as the admin
  - In order to create the superuser (admin) within the django fw - we have to use following command.

```
python manage.py createsuperuser
```

  - In order to Register the model to admin panel we have to use register function present within site class which available within admin model

```
admin.site.register(models.classname)
                    (studentdetails)
```

  - In order to make some changes with respect to the admin panel that is display the specific data we have to use manage method called as __str__ So we have to Return specific data as o/p

```
models.py   def __str__(self):
                return '%s' % (self.sname)
```

Note:- once magic method is been called by django fw then the specific field will be displayed the information we have to

* In order to display the information we have to override this magic method.

Procedure to install mysql client:-

pip install mysqlclient

(or)

pip3 install mysqlclient

(or)

Pip python -m pip install mysqlclient