* The allowed modes in python:- * For text - format * (1)

i) r → open an existing file for read operation. The file pointer is positioned at the beginning of the file. If the specified doesn't exist then we will get FNFError

ii) w) → open an existing file for write operation. If file already contains some data then it will be overridden. If the specified file is not already available then this mode will create that file.

iii) a → open an existing file for append operation. It won't ~~override~~ overridden existing data. If the specified file is not already available then this mode will create a new file.

iv) r+ → To read and write data into the file. The previous data in the file will not be deleted. The file pointer is placed at the beginning of the file.

v) w+ → To write and read data. It will overridden existing data.

vi) a+ → To append and read the data from file. It won't overridden the existing data.

vii) x → To open a file in exclusive creation mode for write operation. If already file exists then File exists errors are present.

## With statement :-

→ This statements can be used to do all files operations group together

→ The main advantage of 'with' statement is after Completion of all file operations, we need not close the file explicitly.

→ After executing with block statements file will be closed automatically

Ex: with open("zzz.txt", 'w') as f:
    f.write("welcom to durgasoft")
    f.write("hello hyderabad")
    Print("is file closed", f.closed) — False
Print("is file closed", fclosed). — True.

## tell() and seek() :-

→ "tell()" method is used to know the cursor Current position

→ 'seek()' method is used to move the cursor position at desired location

Ex-1: f = open("zzz.txt", 'r')
    Print(f.tell())
    Print(f.read(3))
    Print(f.tell())
    f.seek(9)
    Print(f.tell())

Example with file modifications :-

```
f = open("bbb.txt", 'w')  → data = "my name is durga"
f.write(data)
with open('bbb.txt', 'rt') as f:
    text = f.read()
    Print(text)
    Print("Current Cursor location", f.tell())
    f.seek(11)
    Print("Current Cursor location", f.tell())
    f.write("inshan")
    Print("Current Cursor location", f.tell())
    f.seek(0)
    text = f.read()
    Print("Data after modifications")
    Print(text).
```

* program to check whether the file is existing or not :-

→ To check whether file exist (or) not we use "os" library, then 'os' library is having information regarding files present in our disk location

→ "os" library is having sub module called "path"

→ path sub module is having a function called "isfile"

→ 'isfile' function is used to check, whether the file is exisist (or) not

Ex:-

```
import os, sys
fname = input("Enter file name to check")
if os.path.isfile(fname):
    print("File is exisist", fname)
    f = open("fname", 'r')
else:
    print("File doesn't exist", fname)
    sys.exit(0)
data = f.read()
print("The Content of the file")
print(data).
```

* program to Count no. of lines, no. of words, no. of characters present in file :-

same as above program upto sys.exit(0)

```
lcount = ccount = wcount = 0
for line in f:
    lcount = lcount + 1
    ccount = ccount + len(line)
    words = line.split(" ")
    wcount = wcount + len(words).
print("no of lines", lcount)
print("no of characters", ccount)
print("no of words", wcount)
```

## * zip and unziped files:-

⇒ zip files can be used to store all files into one folder that is zip-folder.

→ The main advantage of zip is increase the transportation time and reduce the memory space, we improve performance

→ zip-file extension is ".zip"

→ To create zip-file we use zip-file module

→ This module contains a class called "ZipFile" class

→ To create a zip-file we use a Constant "ZIP_DEFLATED"

Ex:
```
from zipfile import *
f= zipFile("files.zip", 'w', ZIP_DEFLATED)
f.write("abc.txt")
f.write("sample.txt")
f.write("zzz.txt")
f.write("xyz.txt")
print("zip file created")
f.close()
```

→ "unzip" means to extract files from zip-folder

→ For "unzip" operation we use a Constant called "ZIP_STORED"

→ After "unzip" all the file names are present in name.list method

Ex:

```
f = ZipFile("files.zip", 'r', ZIP_STORED)
names = f.namelist()
Print(names)
Print(type(names))
for name in names:
    Print("the file name is", name)
    Print("the Content of the file is")
    f = open(name, 'r')
    text = f.read()
    print(text).
```



→ Working with csv-file:

csv - stands for "Comma Separated Values"

→ To work with "csv" file we should import csv module

→ To csv-files to write the data we use "writer" method

→ To read the data we use "reader" method.

* program to write the data into csv-file:

```
import csv
with open("emp.csv", "w", newline = "") as f:
    w = csv.writer(f)
    w = writerow("ENO", "ENAME", "ESAL", "EADDR")
    n = int input("Enter no. of Employees")
```

```
for i in range(n):
    eno = input("Enter Employee no")
    enam = input("    "    name")
    esal = input("    "    salary")
    eaddr = input("    "    address")
    wo.writerow([eno, enam, esal, eaddr])
print("Total Employee's data written to csv file successfully")
```

* program to read data from csv-file :-

```
import csv
f = open("emp.csv", "r")
r = csv.reader(f)
data = list(r)
print(data)
for line in data:
    for word in line:
        print(word, "\t", end=" ")
    print().
```

## ** Pickle and unpickle :- **



→ pickling is the process of writing object into file

→ unpickling is the process of reading object data from file

→ To work with pickling and unpickling we use module called "pickle".

→ To pickle object used method. i.e dump()

→ To unpickle object used method i.e load().

Ex:-

```
import pickle
class Employee :
    def __init__(self, eid, ename, eaddress ):
        self.eid = eid
        self.ename = ename
        self.eaddress = eddaress
    def display(self):
        Print (self.eid, self.ename, self.eaddress)
with open("emp.dat", 'wb") as f:
    e = Employee (101, "sai", "hyd")
    Pickle.dump (e, f)
    Print("Employee data dumped into file")
with open("emp.dat", "rb") as f:
    obj = pickle.load (f)
```

pickling

unpickling

```
print("Employee data load from file")
obj.display()
```

### *** Packages in python :-

Packages → modules → classes → methods → Variables.

→ package is a folder (or) directory which consists of modules and sub-packages

→ we can access modules from package by using following syntax:

Syntax:

```
import sys
sys.path.append("path of the package")
```

→ Right click on python@3pm project; go to new, click on Python package give the name pack-1; Right click on Pack-1 go to new click on python file give the name module-1; Same like Create module-2;

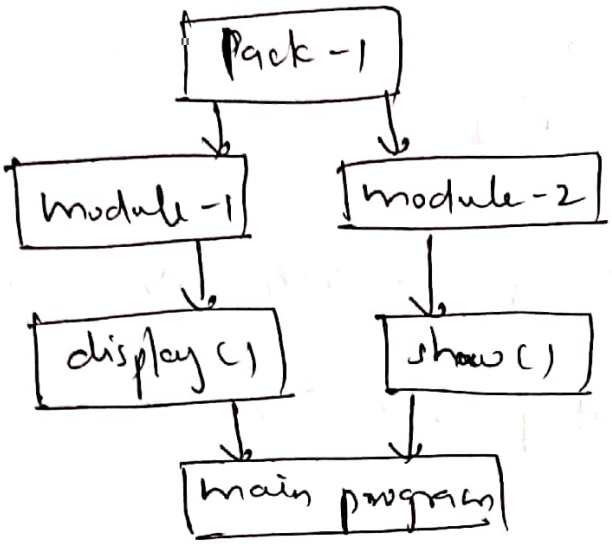
module-1.py:-

```
def display():
    print("Display function
          from module-1")
```

module-2.py:-

```
def show():
    print("show function
          from module 2")
```

## * Importing module from single package :-




→ Right click on python@3pm ; go to new and click on Python file give the name "mainprogram.py" ;

mainprogram.py :-

```
import sys
sys.path.append("C:/python@3pm/pack-1")
```

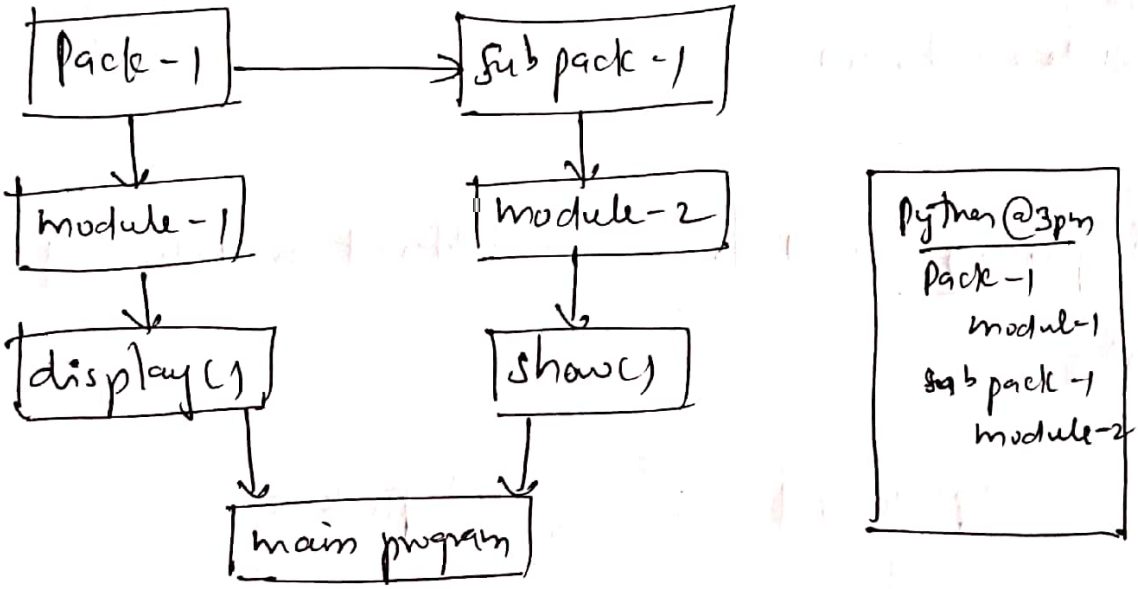
```
# option-1 :-
import module1
import module2
module1.display()
module2.display()
```

```
# option-2 :-
from module-1 import *
from module-2 import *
display().
show()
```

→ steps to give path of the package :-

Right click on pack-1, click on "Copy path", select "absolute path" place into sys.path.append(" ") and change the slash into "\" to "/".

* importing modules from different packages :-




module-1 :

```
def display():
    Print("display function from
        Pack-1 - module-1")
```

module-2 :

```
def show():
    Print("show function
        from pack-1 - Sub
        Pack-1 -
        module2
"Pack-1 → Sub pack-1 → module-2
```

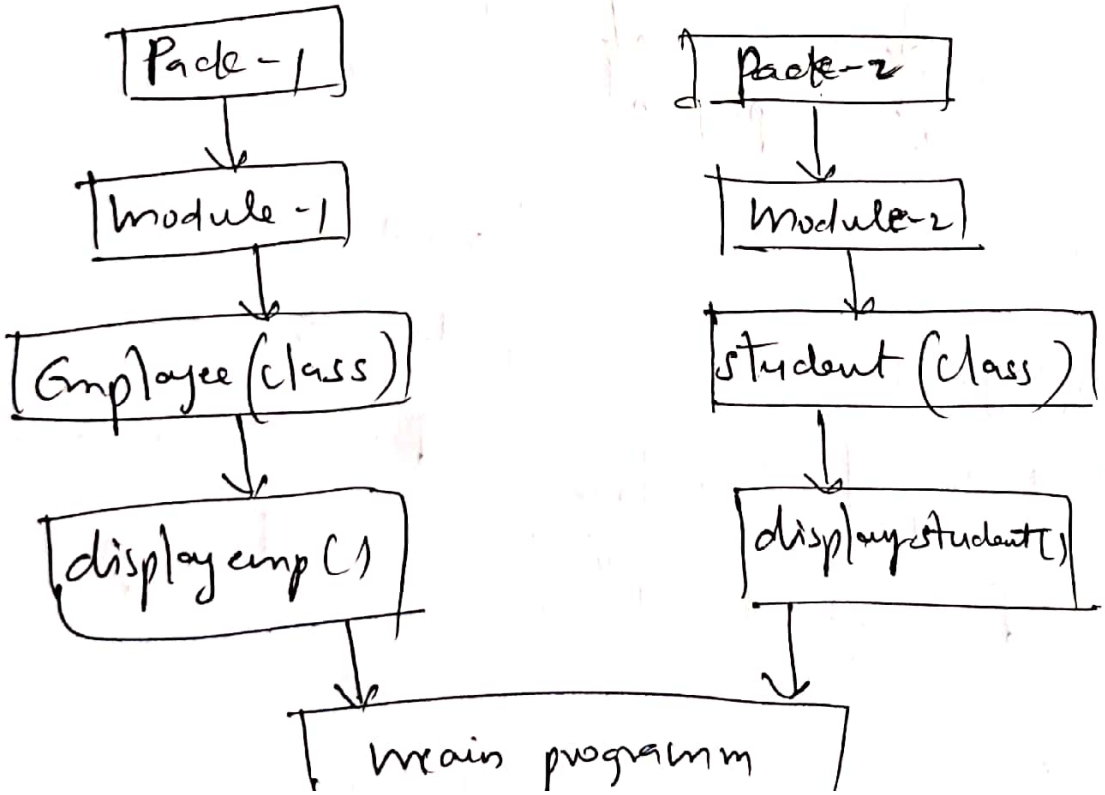
mainprogram:

```
import sys
sys.path.append("G:/python@3pm/pack-1")
import module1
module1.display()
sys.path.append("G:/python@3pm/pack-1/subpack-1")
import module2
module2.show()
```

pack-1

pack-1 / subpack-1

* Importing classes from two different modules and Packages :-



→ module1.py :

```
class Employee:
    def __init__(self, eid, ename, eaddress):
        self.eid = eid
        self.ename = ename
        self.eaddress = eaddress
    def displayemployee(self):
        print(self.eid, self.ename, self.eaddress)
```

→ module-2.py :

```
class Student:
    def __init__(self, sid, sname, saddress)
        self.sid = sid
        self.sname = sname
        self.saddress = saddress
    def displaystudent(self):
        print(self.sid, self.sname, self.saddress)
```

→ mainprogramm.py :

```
import sys
sys.path.append("G:/python@3pm/pack-1")
from module-1 import *
e = Employee(101, 'sai', 'hyd')
e.displayemployee()
```

pack-1 → module-1

```
sys.path.append("c:/python@3pm/pack-2")
from module2 import *
s = Student(111, 'Amar', 'Guntur')
s.displaystudent()
```

pack-2 → module-2

## Regular Expressions in python :-

→ It's a declarative mechanism to represent a group of strings according to particular pattern (or) format

→ We can use regular expressions to represent mobile numbers, mailids, passwords

→ Using regular expressions we can develop pattern matching applications, Validation framework and logic.

→ To work with regular expressions in python there is a built-in module called "re"

→ 're' module is having builtin functions

i) **Compile()**:- is used to Compile a pattern into regex objects

ii) **finditer()**:- is used to find iterations of match object

iii) **start()**:- on, match object we can apply the following methods.

i) **start**:- starting index

ii) **end**:- end+1 index

**group()**:- matched string

Ex:-1:-
```
import re
count = 0
pattern = re.compile('ab')
match = pattern.finditer("abcbcabcabcc")
for m in match:
    count += 1
    print(m.start(), "---", m.end(), "---", m.group())
print("no.of occurence:", count).
```

o/p:- start end group
0 → 3 → ab
5 → 7 → ab
8 → 10 → ab
no.of occurence = 3

→ character classes:-

i) [abc] = Either a (or) b (or) c
ii) [^abc] = Except a and b and c
iii) [a-z] = Any lower case alphabet symbol
iv) [A-Z] = Any upper " "
v) [a-z A-Z] = Any alphabet both lower and uppercase
vi) [0-9] = Any digit from 0-9
vii) [a-z A-Z 0-9] = Any alphanumeric characters all lower, upper, digits
viii) [^a-z A-Z 0-9] = Except Alphanumeric characters only (Special Symbols)

Ex: 2:-

Compile character classes (16)

```
import re
match = re.finditer("[a-zA-Z0-9]", "ab12Th'A#b@56")
for m in match:
    print(m.start(), "---", m.group())
```

→ predefined character classes:-

i) \s → space character

ii) \S → Any character except space

iii) \d → Any digit from 0-9

iv) \D → Any character except digit

v) \w → Any word character ([a-zA-Z0-9])

vi) \W → Any character except word character (special characters)

vii) . → Any characters including special characters also.

Ex: 3:-

```
import re
match = re.finditer("[\s]", "ab_12_ThA#---b@56")
```

(space)

```
for m in match:
    print(m.start(), "---", m.group())
```

→ Quantifiers:

a → Exactly one 'a'

a+ → Atleast one 'a'

a* → Any no of 'a' including zero number

a? → Atmost one 'a' either zero number (or) one number

a(m) → Exactly m no of 'a'

a(m,n) → min m no of 'a', and max n no of 'a's.

Ex:-

```
import re
match = re.finditer("a{1,3}", "aa5bc ab aaabbcc")
for m in match:
    print (m.start(), m.group()).
```

→ important functions in "re" module:-

match, fullmatch, search, findall, sub, subn, split, Compile, finditer.

i) match():- Is used to check, whether the provided pattern is starting at target string (or) not.

→ This method will returns match object if the provided pattern is starting of target string. otherwise it will returns.

Ex:-
```
import re
s = input("Enter pattern to check")
m = re.match(s, "durgasoft")
if m != None:
    Print("provided pattern matched at starting of
        target string")
else:
    Print("provided " not matched " " ")
```

ii) fullmatch():- This is used to check, whether provided pattern fully matched with target string (or) not

Ex:-
```
import re
s = input("Enter pattern to check")
m = re.fullmatch(s, "durgasoft")
if m != None:
    Print("fully matched")
else:
    Print("not matched")
```

I/P:- 1. durgasoft
Result → matched

2. —durgasoft
not matched
because space is present

iii) search():- Is used to check, whether provided pattern is available in target string (or) not. (19)

Ex:-
```
import re
s = input("Enter pattern to search")
m = re.search(s, "durgasoft")
if m != None:
    print("search is available at", m.start())
else:
    print("search is not available")
```

O/P:-
1. du
⇒ present in side by side
2. ra
⇒ not present because these are not in side by side

iv) findall:-
This method is used to find all occurances from the match

→ This method will return all matched things into list [].

Ex:-
```
import re
m = re.findall("[A-Za-z]", "Aa#12@x__sz3-*")
print(m)
```

v) sub():- means substitute (or replacement)

→ This function replace target string with provided replacement based on "regex".

vi) subn: This is same as sub, but only the difference is "subn" will return no. of replacements also (20)

→ subn function will return tuple with two elements

i) Result string
ii) No of replacements

Syntax of sub, subn:

m = re.sub("regex", "replacement", "target string")

Ex: import re
m = re.sub("[0-9]", "#", "Aa#12@xyz3*")
Print(m) o/p: # is replaced by numbers

} sub

Ex:
m = re.subn("[0-9]", "#", "Aa#12@xyz3*")
Print(m)
Print("Result string =", m[0])
Print("no. of replacement", m[1])

} subn

vii) split(): used to split target string based on the seperated. By default split function returns list.

Ex:

Ex: import re

```
l = re.split("\.", "www.durgasoft.com")
for i in l:
    print(i)
```

o/p: www
durgasoft
com.

i) ^-symbol: is used to check whether the provided thing is starting with target string (or) not

ii) $ → symbol: is used to check weather the provided thing is ending with target string (or) not.

Ex-1: import re

```
s = "hello world"
m = re.search("^hello", s)
if m != None:
    print("target string starts with hello")
else:
    print("  "  not starts  "  ")
```

Ex-2:

```
s = "hello world"
m = re.search("hello$", s)
if m != None:
    print("target strings ends with hello")
else:
    print("  "  not ends with  "  ").
```

## Program to represent 10 digit mobile number:

i) Every number should contain exactly 10 digits

ii) The first digit should be in [7, 8, 9]

→ [7-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]

→ [7-9][0-9]{9}

→ [7-9]\d{9}

→ \d{10} → to represent 10 digit mobile number.

} ways.

Eg:-
```
import re
n = input("Enter mobile number")
m = re.fullmatch("[7-9]\d{9}", n)
if m1 = None:
    print("Valid mobile number")
else:
    print("Invalid mobile number")
```

o/p: i/p 7954321067

valid

2. 5432167897

not valid.

* program to represent mail-Id: only gmail:

```
import re
s=input("Enter mail Id")
m= re.fullmatch("[a-zA-Z0-9._]+@gmail[.]com",s)
if m!=None:
    print("Invalid mail Id")
else:
    print("Valid mail Id")
```

→ for all mail-Id formats:

```
[a-zA-Z0-9._]+@[a-zA-Z]+[.][a-zA-Z]+
```

** Assertion: ***

→ Debugging python program using "assert" keyword:

→ In general debugging is the process of identify the 'bug' and fixing the bug

→ Any common way may to debug python program is by using print() statement.

→ But the problem with print() is, after completion

of debugging, if we forgot remove extra added print() statements that will make Console window distrubs, to avoid this we use "assert" statement.

→ The main advantage of "assert" statement over "print()" is after Completion of debugging we need not to remove extra added assert statements..

→ assert Condition_Expression msg → Syntax:

→ Condition Expression either returns true (or) false, if it is true, then program will continue with Execution (or) false then it shows assertion error and make alert to the programmer.

Ex: def square(n):
    return n*n

assert square(3) == 9, "square of 3 is 9"
assert square(4) == 16, "square of 4 is 16"
Print(square(3))
Print(square(4))

o/p: No error because 3*3 = 9
4*4 = 16

⇒ Exception is for handling run-time errors, whereas assertion is for handling development errors.



→ Exception = Error

Assertion = Issue.

## logging in python:

→ logging Concept is recommend to store complete programm execution flow into a file

→ logging is the Concept of creating "log" files

→ we can use log-files while performing "debugging"

→ In python there is a built-in module called "logging"

→ To perform logging, first we need to create file to store msgs and we have to specify, which level of msgs to store, we can do this by using "basicConfig" function

→ we can store warning and information and debugging msgs into log-file.

Ex:
```
import logging
logging.basicConfig(filename="mylog.txt", level=logging.INFO)
logging.info("A new request came")
try:
    x = int(input("Enter a number-1"))
```

```
y = int(input("Enter number-2"))
print(x/y)
```



```
except ZeroDivisionError as msg:
    print("Can't divide with zero")
    logging.exception(msg)
except ValueError as msg:
    print("Enter only int values")
    logging.exception(msg)
logging.info("Request processing completed")
```

## Multitasking :-

→ It is the process of to perform several tasks simultaneously.

→ Multitasking can achieve in two different ways :-

i) process based multi-tasking (multiprocessing)

ii) Thread based multitasking (multithreading)

→ In multiprocessing every process needs a seperate memory.

→ A process is a heavy weight.

→ Cost of communication b/w process to process is "high".

→ In multithreading every thread is not require a seperate memory, threads can share the memory

→ Thread is a light weight

→ Cost of Communication b/w Thread is low



→ In python by default, every program will run under a Thread i.e "main Thread"

→ we can get the thread name by using "getName" function

→ we can change the thread name by using "setName" function.

Ex:- → To work with multithreading pwgams in python we use the built-in module "Threading"

Ex:- from Threading import *

```
Print (Current_thread().getName())
Current_thread().setName("Pinga Thread")
Print (Current_thread().getName())
```

→ let us consider the program without multithreading:-

```
import time
def square(numbers):
    Print("square numbers")
    for n in numbers:
        time.sleep(0.2)
```

```
        print("square is", n*n)
def cube(numbers):
    print("Cube numbers")
    for n in numbers:
        time.sleep(0.2)
        print("cube is", n*n*n)
lst = [2,3,4,5,6]
t = time.time()
square(lst)
cube(lst).
print("done in", time.time() - t)
```

Hyphen

Op:- done in : 2.031171

→ Example with multithreading:-

```
import time
import threading
def square(numbers):
    print("square numbers")
    for n in numbers:
        time.sleep(0.2)
        print("square is", n*n)
def cube(numbers):
```

```
print("Cube numbers")
for n in numbers:
    time.sleep(0.2)
    print("Cube is", n*n*n)
lst = [2,3,4,5,6]
t = time.time()
t1 = threading.Thread(target = square, args=(lst,))
t2 = Threading.Thread(target = Cube, args=(lst,))
t1.start()
t2.start()
t1.join()
t2.join()
print("done in", time.time() - t)
```

o/p: done in 1.031127

Note: whether it is process based multitasking (or) Thread based multitasking only the target is to increase the responsive time and saving the time

→ In the above eg: t1, t2 are Thread objects

→ target is an attribute, which is used to execute a function by thread.

→ args is attribute, which is used to pass arguments to the target function.



→ To start Thread we use "start()" method

→ "join()" method can be used to stop the execution of current program until a thread is complete.

In python we can create threads in 3 ways:-

i) Creating thread without using any class.

ii) " " by extending from thread class.

iii) " " without " any thread class.

1) Creating thread without using any class:

```
from threading import *
def display():
    for i in range(1,4):
        print("child thread")
t = Thread(target = display)
t.start()
for i in range(1,4): # This code is executed by main thread
    print("main thread")
```

→ In above eg multiple threads executing parallely.

→ when multiple threads are execute, we don't ~~except~~ except proper o/p, it is different from every run to run.

31

## 2) Creating thread by extending from Thread class:-

→ First, we have to Create a class, which is extended from thread class.

→ Inside the class, we have to override "run()" method with required task.

→ when we call "start()" method, automatically "run()" method will execute and perform required task.

Eg:-

```
from threading import *
class MyThread(Thread):
    def run(self):
        for i in range(1,4):
            print("child class")
t = MyThread() → object for class
t.start()
for i in range(1,11): # This Code is executed by main thread #
    print("main class")
```

3) Creating Thread without extending any class :-



```
from threading import *
class MyThread():
    def display(self):
        for i in range(1,11):
            print("child thread")
obj = MyThread()
t = Thread(target = obj.display)
t.start().
```

* Thread identification number :-

→ Every thread internally a unique identification number is available, we can access, thread unique Id number by using the implicit variable "ident"

Ex:

```
from threading import *
def display():
    for i in range(1,11):
        print("child Thread")
t = Thread(target = display)
t.start()
print("main thread Id number", current_thread().ident())
print("child thread Id number", t.ident)
```

## join() :

→ If a Thread wants to wait untill the Completion of other Thread, Then we use "join()" method



Ex:
```
from threading import *
import time
def display():
    for i in range(10):
        print("child-thread")
        time.sleep(2)
t = Thread(target = display)
t.start()
t.join(5)
for i in range(10):
    print("main Thread")
```

→ After Completion of join(5) seconds then automatically the main-thread will start Executed, after the Completion of main Thread the child-thread will start Execution from where it had delayed.

## Thread Synchronization :-

→ If multiple Threads are executing simultaneously, we will get data inconsistency problems.

Ex: let us Consider the program :-

```
from threading import *
import time
def display(name):
    for i in range(10):
        print("hello", name)
        time.sleep(1)
        print(name)
t1 = Thread(target = display, args = ("mohan",))
t2 = Thread(target = display, args = ("sai",))
t1.start()
t2.start()
```

→ In the above example we are getting irregular o/p's i.e, data inconsistency, to avoid this we use Thread synchronization

→ Thread synchronization means at a time only one Thread.

→ Thread synchronization can be achieve in three different ways

i) Lock

ii) RLock

iii) Semaphore.



## i) Lock():-

→ lock is the fundamental synchronization mechanism, Provided by thread module

→ we can create (lock()) object like "l = Lock()"

→ The lock object can hold only one thread at a time, if any other thread required the same lock, Then it should wait until, thread release the lock.

→ To release the lock, Thread should be owner of that lock.

→ Thread can acquire the lock by using "acquire()" method

→ " release " " " " "release()" method

Ex: from threading import *

```
l = Lock()
# l.acquire() -> RuntimeError: released unrealsed lock.
l.release()
```

Ex: with lock() Concept:-

```
from threading import *
l = Lock()
def display(name):
    l.acquire() -> acquiring lock.
```

```
for i in range(10):
    Print("hello", name)
    Print(name)
    l.release()  ⟶ releasing lock
t1 = Thread(target = display, args = ("mohan",))
t2 = Thread(target = display, args = ("sai",))
t1.start()
t2.start()
```



→ The problem with simple "Lock()" is if thread already lock, then other threads acquire the same lock, threads will been blocked. To avoid this we use "RLock()"

→ "RLock()" means Reentrant Lock :-

→ In the "RLock" Concept one thread can acquire the lock again and again

Ex:- using RLock() Concept :-

```
from threading import *
l = RLock()
Print("main thread acquiring the lock")
l.acquire()
Print("main thread acquiring the lock again")
l.acquire().
```

iii) Semaphore() :-

→ In the case of lock and Rlock at a time only one Thread is allowed

→ Sometimes our requirement is to execute multiple Threads at a time, in that case "Semaphore()" Can been used.

→ we Can create Semaphore object like "s = Semaphore(counter)"

→ The default Counter part is 'one' = counter(1)

→ when the thread executes acquire method, then the Counter Part decrease 'one'

→ when the Thread executes relase method, then the Counter part increased by 'one'

Ex:
```
from threading import *
import time
s = Semaphore(3)
def display(name):
    s.acquire()
    for i in range(5):
        print("hello", name)
        time.sleep(2)
    print(name)
    s.release()
t1 = Thread(target = display, args = ("sai",))
t2 = Thread(target = display, args = ("mohan",))
```

```
t3 = Thread(target = display, args = ("vijay",))
t4 = Thread(target = display, args = ("durga",))
t5 = Thread(target = display, args = ("pavan",))
t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
```

(58)

## ** Decorator :- **

→ Decorator is a function, which can take a function as an argument and extend it's functionality and returns modified function with extended functionality.

→ The main objective of decorator is we can extent the functionality of existing functions without modifying that function.

Eg:
```
def wish(name):
    print("hello", name, "good morning")
wish(mohan)
wish(durga)
```

o/p:
```
hello mohan good morning
hello durga good morning
```

→ This functions can display always same o/p, but we want to modify the function to provide different msg if name is "raj"

→ We can do this, with out touching this function by using decorators.



Eg:1

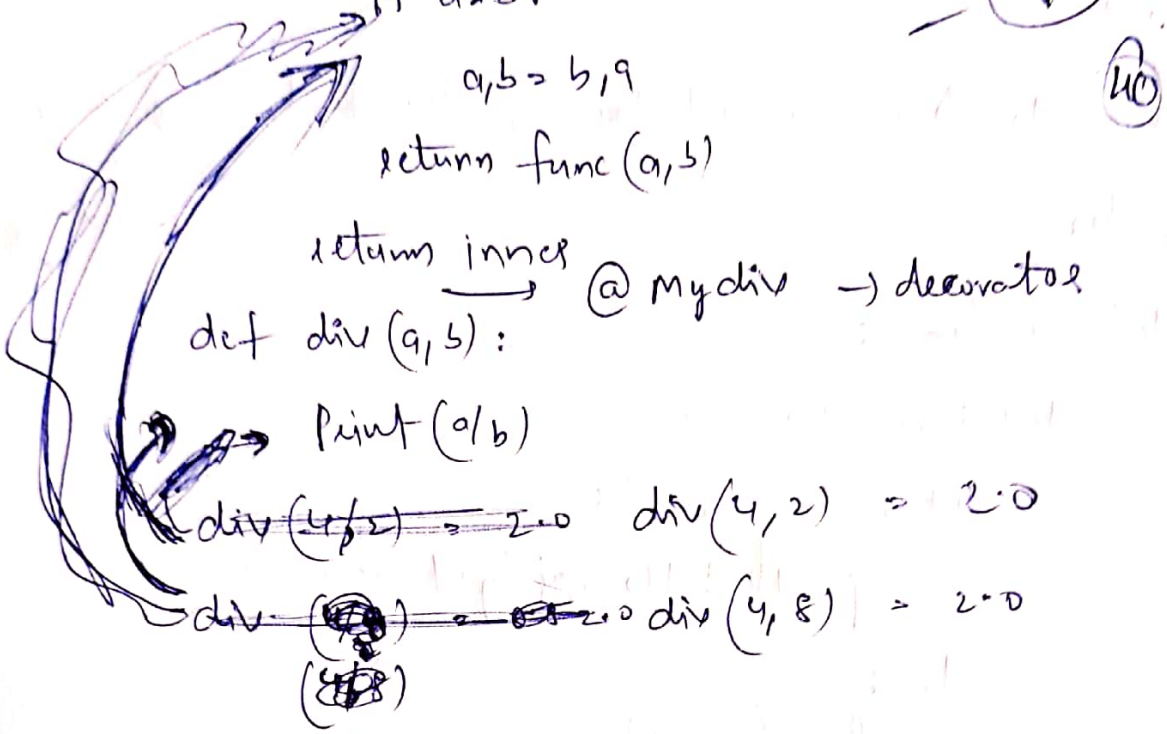
```
def fi(func):
    def inner(name):
        if name = 'raj':
            print("hello raj good evng")
        else:
            func(name)
    return inner
@fi  ⟶ decorator
def wish(name):
    print("hello", name, "good mrng")
wish(mohan)
wish(durga)
wish(raj)
```

o/p:

```
hello mohan good mrng
hello durga good mrng
hello raj good evng
```

Ex:- 2

```
def Mydiv(func):
    def inner(a,b):
        if a>b:
            a,b = b,a
        return func(a,b)
    return inner
@Mydiv
def div(a,b):
    print(a/b)
```



div(4,2) = 2.0

div(4,8) = 2.0

→ Always decorator name should starts with "@" symbol

## Generators :-

i) Generator is a function, which is used to generate the numbers (or values, with the help of "yield" keyword

→ Let us Consider the example :-

```
a = [i for i in range(0,999999999999)]
print(type(a))
for value in a:
    print(value)
```

→ In the above example a, is a list.

→ In the above example, we are to trying to generate large no of values, so that, their will be chance of memory error, because huge no of values first stored in memory, then after it should generates the values, to avoid the memory error with huge no of values we use generators. (41)

Ex: a = (i for i in range (0, 9999999999999))
print (type (a))
for value in a:
    print (value)

→ In the above example, 'a' is the generator, which is going to generate the values with out storing all the values in the memory

→ Generator-functions returns iterators, which can iterate one value at a time.

→ Generator-function is defined as like normal function, but whenever it needs to generate the value it will do with "yield" keyword rather than "returns"

→ If a body of def. Contains yield, the function

automatically becomes generator function.

Eg: with generator function



```
def genfunc():
    yield 'A'
    yield 1
    yield 'B'
    yield '2'
    yield 3
for i in genfunc():
    print(i)
```

*) generator-object :-

→ when we assign a generator function to variable, that variable is said to be generator-object

→ generator objects are used either by called "next()" method on generator-object (or) generator object in a for loop.

Eg:
```
def genfunc():
    yield 'A'
    yield 1
    yield 'B'
    yield 2
x = genfunc()
print(x.__next__())
print(x.__next__())
print(x.__next__())
```

→ for limited values we use next():

op: A
1
B

Ex: def genfunc():
    yield 'A'
    yield 1
    yield 'B'
    yield 2
x = genfunc()
for i in x:
    print(i)

"/b"

A
1
B
2

## python Database Connectivity :- (V.V.imp)

→ using python we can connect to any database like mysql, oracle, sql, etc

→ To connect any database, in python we use a library called "pyodbc" → python open database connectivity.

→ "pyodbc" will not come by default, we need to install explicitly

→ for this go to command prompt pip3 — install — pyodbc

Ex: To do CRUD operations using MS-sql server :-

→ for this we need to install sql-server 2016 software, after installation go to run type "ssms" (sql server management studio) click on 'ok'.

→ choose server type is "database engine" give the

Server name as (.), select authentication (windows), click on Connect, Go to object explorer right click on database folder, click on new database give the database name as python@3pm, then click on (ok). Expand python@3pm in object explorer, right click on tables go to (new) and click on table, give the required Column names and its datatypes click on save, give the table name (emp), go to object explorer right click on (emp) table click on (edit top 200 rows), insert any two records, click on execute (or press (F5.)

Ex:
```
import pyodbc
def read(Con):
    print("read")
    Cursor = Con.cursor()
    Cursor.execute("Select * From emp")
    for row in Cursor:
        print(row)
    print(====)
def create(Con):
    print("Create")
    Cursor = Con.Cursor()
```

read:

create:

```
        Cursor.execute("insert into emp id, name, address,
        salary) Values(?,?,?,?);", (3, 'raj', 'srnagar', 45000))
        Con.Commit()
        read(Con)
```



```
def update(Con):
    print("update")
    Cursor = Con.Cursor()
    Cursor.execute("update emp set name = ?, address = ?,
    salary = ?, where id = ?;", ("manoj", "srnagar", 23000, 1))
    Con.Commit()
    read(Con)
def delete(Con):
    print("delete")
    Cursor = Con.Cursor()
    Cursor.execute("delete from emp where id = 2")
    Con.commit()
    read(Con)
Con = pyodbc.Connect("Driver={SQL Server}; Server=.;
    database = python@3pm")
read(Con)
Create(Con)
Update(Con)
delete(Con)
Con.close().
```

# Python - GUI

## Python - GUI's

→ Using python we can develop desktop GUI, using Python GUI we can develop text-editor app's like Notepad, wordpad, editplus, python IDLE.



→ To develop GUI App's in python we use a module called as "tkinter"

→ tkinter module is available by default.

→ The Complete python IDLE was designed and develop using tkinter module

→ To Create a window (or) form, we need to create object for "TK()" class.

→ To display the form continously we use "mainloop()" method

→ "mainloop()" is infinite loop

Ex:

```
from tkinter import *
w = TK()
w.title("Python GUI")
w.geometry("600x500")  # width x height
w.maxsize(700, 600)
w.minsize(200, 200)
w.mainloop()
```

"Small x"

→ To set width and height of the window we use "geometry()" method

→ To provide restriction for maximum width and height we use "maxsize()" method



→ To provide restriction for minimum width and height we use "minsize()" method

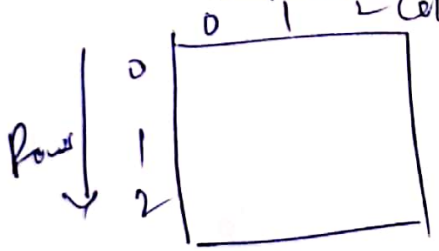
→ A window is also called as Container, which Contain some Controls (or widgets with in it.

→ After Creating Controls, to add Controls to our window, we use the following methods :-

i) Pack()
ii) grid()
iii) place()

i) pack() :- By default pack() will place the Controls in Center of window

ii) "grid()" :- Can use to place the Controls in table structure



→ place():- is used to place the Control in a particular position

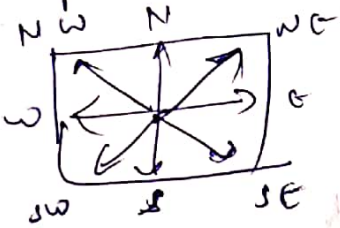
Ex:
```
from tkinter import *
w = Tk()
w.title("python GUI")
w.geometry("500x400")
```
} pack() method

(49)

```
lb1 = label(text = "welcome to durgasoft", font=("algerian", 20),
      fg = "red", bg = "yellow")
lb1.pack()
w.mainloop()
```

welcome to durgasoft

```
# lb1.pack(anchor = "sw", side = BOTTOM)
```




Ex: with label-image:-

```
from tkinter import *
w = Tk()
w.title("My First GUI")
w.geometry("500x400")
img = photoImage(file = C:/Users/Anvar/Desktop/myimage.png)
lb1 = label(image = img)
lb1 = pack()
w.mainloop()
```

Ex: with button event:-

```
from tkinter import *
w = Tk()
w.title("python GUI")
w.geometry("600x500")
def f1():
    lb1.config(text="button was clicked")
lb1 = Label(text="Hello", font=("arial", 15, "bold"))
lb1.pack()
btn = Button(text="submit", font=("arial", 15, "bold"))
btn.pack()
w.mainloop()
```



Ex: with message box:-

```
~~def f1():~~ from tkinter import *
    from tkinter import messagebox
    w = Tk()
    w.geometry("600x500")
    w.title("python GUI")
    def f1():
        messagebox.showinfo("message", "Button was clicked")
    btn = Button(text="submit", font=("arial", 15, "bold"), command=f1)
    btn.pack()
    w.mainloop()
```

Eg: with scrolledtext box:-

```
from tkinter import *
from tkinter import scrolledtext
w = Tk()
w.title("python GUI")
w.geometry("600x300")
txt = scrolledtext.ScrolledText(width=30, height=10)
txt.insert(INSERT, "you text goes here")
txt.pack()
w.mainloop().
```

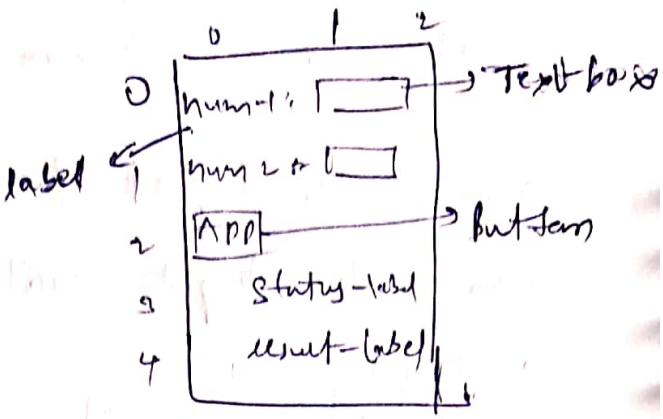


Eg: with text-box and button event:-

```
lbl = Label(text="hello", font="arial, 15 .bold")
lbl.pack()
txt = Entry(width=20, font="arial, 15 bold")
txt.focus()
txt.pack()
def fi():
    result = ("welcome to " + " " + txt.get())
    lbl.config(text=result)
btn = Button(text="click Here", font="arial 15 bold",
             command=fi)
btn.pack()
```

Eg: To add two numbers :

* using grid() *

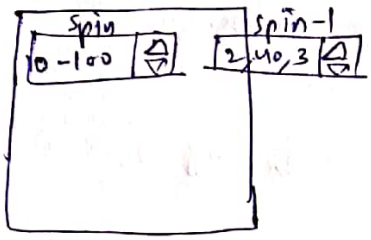



```
lbl1 = Label(text = "Num:-1", font = "arial, 15 bold")
lbl1.grid(row=0, Column = 0)
lbl2 = Label(text = "Num: 2", font = "arial. 15 bold")
lbl2. grid(row =1, Column= 0)
txt1 = Entry(font="arial, 15 bold,") → width = 15
txt1. grid(row = 0, Column = 1) → txt1. focus()
txt2 = Entry(font = "arial 15 bold,") → width =15
txt2. grid(row = 1, Column = 1) → txt2. focus()
def add():
    if(txt1.get() and txt2.get() != ""):
        try:
            num1 = float(txt1.get())
            num2 = float(txt2.get())
            result = num1 + num2
            result-label.Config(text = result)
            status_label.Config(text = "Successfully Computed")
        except:
```

```
        status_label.Config(text="invalid i/p, check the
                                  i/p type")
    else:
        status_label.Configure(text="fill all the required fields")
btn = Button(text="Add", font="arial 15 bold", width=5,
                                  command=add)
status_label = Label(font="arial 15 bold")
status_label.grid(row=4, Column=0)
result_label = Label(font="arial 15 bold")
result_label.grid(row=5, Column=0)
w.mainloop()
```

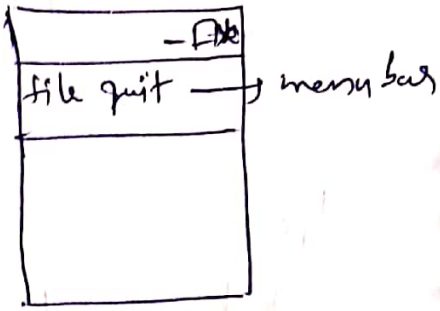
(53)

Ex: with spinbox() :-




```
spin = Spinbox(from_=0, to=100, width=15, font="arial 15 bold")
spin.grid(row=0, Column=0)
spin1 = Spinbox(values=(list(range(2, 40, 3))), width=15, font=
                                  "arial 15 bold")
spin1.grid(row=0, Column=3)
w.mainloop()
```

Ex: with menu:-




```
→ from tkinter import msgbox
def f1():
    messagebox.showinfo("Hello", "This is my first menu")
m1 = Menu()
m1.add_command(label = "File", command = f1)
m1.add_command(label = "Exit", command = quit)
w.config(menu = m1)
w.mainloop()
```



## ~~Multiple~~ Inheritance

i) Isinstance():-

ii) By subclass():-

→ "isinstance()" is used to check whether an object is instance of particular class (or) not.

→ "issubclass()" is used to check whether a class is subclass of particular main class (or) not.

Ex:
```
class Vehicle():
    def f1(self):
        print("Vehicle")
class motorcycle(Vehicle):
    def f2(self):
        print("motorcycle")
class Car(Vehicle):
    def f3(self):
        print("Car")
c = Car()
m = Motorcycle()
```

```
class(A):
    class B(A):
    class C(A):
    class D(A):
```
Single inheritance

Single inheritance



print(isinstance(c, motorcycle)) = False

print(isinstance(m, motorcycle)) = True

print isinstance(m, Car) = False, (m, motorcycle) = True

print(issubclass(Car, Vehicle)) = True

print(issubclass(motorcycle, Vehicle)) = True

print(issubclass(Car, motorcycle)) = False

print(issubclass(motorcycle, Car)) = False

## Inheritance: (practise purpose)



```
class (A):
class B(A):
class C(A):
class D(A):
    }
class n(A):
```

Single Inheritance

```
class (A):
class B(A):
class C(B):
class D(C):
    |
class (n(n-1)):
```

multi-level Inheritance

```
class (A):
class (B):
class (C):
class D(A, B, C):
```

multiple Inheritance.

```
class ∈ (A, B, C, D): (or) class ∈(D):
```

## Webscraping:

→ It is the process to extract large amount of data from website.

→ The data which will come from different website is un-structured.



→ And we need to store in a structured form

→ There are different ways to scrap the data from websites like online services, API's (or) with our code

→ webscraping is a legal for some website and it is illegal for some other website.

→ To known whether it is legal (or) illegal

"www.flipkart.com/robots.txt"

Ex:
```
import re, urllib
import urllib.request
u = urllib.request.urlopen('http://www.dhenzi.com/Contact.asp')
text = u.read()
numbers = re.findall('[0-9]{3}[0-9]+', str(text), re.I)
for n in numbers:
    print(n)
```

(re.I = Ignore case #.)

* AnaConda *
* pandas *
* matplotlib *

* pandas *

# pandas and matplotlib

→ pandas → python data analysis

→ pandas is used to create, series and dataframes

→ pandas is an opensource library, that allows you to perform data manipulation in python. (59)

→ pandas library is built on top of "numpy" i.e, pandas needs "numpy" to operate.

→ pandas provide an easy way to create and manipulate the data.

→ In pandas series is a one dimensional datastructure and data frame is two-dimensional array structure.

→ To Create series we need data and index

→ Data can be a list (or) dictionary (or) numpy arrays

→ To work with pandas and matplotlib we use the "Anaconda 3" distribution

→ "Anaconda" is standard platform for python data science.

## Install of Anaconda:

↳ go to google type Anaconda.com; click on link www.Anaconda.com; click on the download; click Python 3.7 Version download. and install. (60)

→ go to search and type Anaconda, click on Anaconda-navigator; click on "Launch Jupyter" notebook.

→ click on "new" from R.H.S; select and click on folder select untitled-folder, click on rename option give the name python@3pm, select python@3pm and click on it.

Ex:-1

```
import pandas
s=pandas.Series()
print(s)
```

Ex:-2:-

```
import pandas as pd
data=[10,20,30,40,50]
s=pd.Series(data)
print(s)
```

o/p:

```
01 = 10
02 = 20
03 = 30
04 = 40
05 = 50
```

Ex:3 Series with Index:

```
import pandas as pd
data = [10, 20, 30, 40, 50]
s = pd.Series(data, index = [101, 102, 103, 104, 105])
Print(s)
```

o/p: 101 = 10 104 = 40
102 = 20 105 = 50
103 = 30



Ex:4

```
import pandas as pd
s = pd.Series(['mohan', 'saj', 'durga'], ['name-1', 'name-2', 'name-3'])
Print(s)
```

o/p: name-1 = mohan
name-2 = saj
name-3 = durga

Ex:5:

```
import pandas as pd
s = pd.Series({'a': 10, 'b': 20, 'c': 30}, index = ['b', 'c', 'a', 'd'])
Print(s)
```

o/p:

b = 20
c = 30
a = 10
d = None.

Ex: 6:

```
import panday as pd
import numpy as np
data = np.array([10, 20, 30, 40])
s = pd.Series(data, index = [1,2,3,4])
Print(s)
```



Ex: 7 Using dataframe:

```
import panday as pd
data = {"names": ["mohan", "saj", "manoj"], "age": [15, 20, 25],
        "salary": [10,00, 20000, 30000]}
df = pd.DataFrame(data)
Print(df)
```

O/p:

| | names | age | salary |
|---|---|---|---|
| 0 | mohan | 15 | 10000 |
| 1 | saj | 20 | 20000 |
| 2 | manoj | 25 | 30000 |

Ex: 8:

```
import pandas as pd
data = {"Name": ['Anna', 'Sai', "balyan"], "Age": [20, 17, 25]}
df = pd.DataFrame(data)
print(df)          → head = from top 5 rows
df.tail(2)         → tail = from bottom 2 rows
df.head(1)
```



Ex: 9: Two dataframes:

```
import pandas as pd
data = [{'a': 10, 'b': 20}, {'a': 5, 'b': 30, 'c': 40}]
df1 = pd.DataFrame(data, index=['first', 'scand'],
                   Columns=['a', 'b'])
df2 = pd.DataFrame(data, index['First', 'scand'],
                   column=['a', 'b,'])
print(df1)
print(df2)
```

Sri:-10 Concatenate the dataframes:

```
Import pandas as pd
df1 = pd.DataFrame({'name':['Arnav', 'Sai', 'Edyan'],
                    'age': [20, 15, 25]}
                    index = [0, 1, 2]}
df2 = pd.DataFrame({'name': ['Dinesh', 'Akshara'],
                    'age': [15, 25]},
                    index = [3, 4]}
df3 = pd.Concat([df1, df2])
Print(df3)
```

(64)

* Matplot.lib *

→ matplotlib is a data science library for visualization.

→ matplotlib built on numpy arrays

→ matplotlib consists several plots like line, bar, scatter, histogram, etc, pie charts.



Ex:1 with linechart:

```
from matplotlib import pyplot as plt
x = [1, 2, 3]
y = [5, 4, 3]
x2 = [6, 9, 11]
y2 = [9, 6, 11]
plt.plot(x, y, 'g', label = 'line one', linewidth = 3)   # 'g' → colour code
plt.plot(x2, y2, 'c', label = 'line two', linewidth = 6)
plt.title("Amos")
plt.ylabel("Y axis")
plt.xlabel("X axis")
plt.legend()
plt.grid(True, color = 'k')
plt.show()
```

Ex:2 with bar-plot:

```
from matplotlib import pyplot as plt
plt.bar([1,3,5,7],[5,2,7,9], label = "Example one",
                                color = 'r')
plt.bar([1,2,3,4],[4,1,6,7], label = "Example two",
                                color = 'c')
plt.legend()
plt.xlabel("bar number")
plt.ylabel("bar height")
plt.title("Bar graph")
plt.show()
```

Ex:3 with scatter graph:

```
from matplotlib import pyplot as plt
plt.scatter x = [1,2,3,4,5]
             y = [4,5,6,7,8]
plt.scatter(x,y, label = "skitscat", color = 'r', s=150,
                                marker = "*")
plt.xlabel('x')
plt.ylabel('y')
plt.title("scatterplot")
plt.show()
```

Ex: 4 with stalle-plot:

```
from matplotlib import pyplot as plt
days = [1, 2, 3, 4, 5]
sleeping = [7, 8, 6, 11, 7]
eating = [2, 3, 4, 3, 2]
working = [7, 8, 7, 2, 2]
playing = [8, 5, 7, 8, 13]
plt.plot([], [], color = 'm', label = "sleeping", linewidth = 5)
plt.plot([], [], color = 'c', label = "eating", linewidth = 5)
plt.plot([], [], color = 'r', label = "working", linewidth = 5)
plt.plot([], [], color = 'k', label = 'playing', linewidth = 5)
plt.stackplot(days, sleeping, eating, working, playing,
              colors = ['m', 'c', 'r', 'k'])
plt.xlabel('x')
plt.ylabel('y')
plt.title('stack plot')
plt.legend()
plt.show()
```

Ex: 5: with pie-chart

```
from matplotlib import pyplot as plt
values = [7, 2, 2, 13]
activities = ['sleeping', 'eating', 'working', 'playing']
cols = ['c', 'm', 'r', 'b']
plt.pie(values, labels = activities, color = cols,
        startangle = 90, shadow = True,
        explode = (0, 0.3, 0, 0),
        autopct = '%1.1f%%')
plt.title('pie plot')
plt.show()
```