Unit - 1 Introduction to Python

Facts about Python:

* Python is a Popular Programming language it was created in 1991 by Guido Van Rossum.

* It has a simple syntax similar to English language.

* Its syntax allows developers to write Programs with fewer lines.

* It can be treated in a procedural way, an object oriented way or a function way.

* Reason behind its name, The author or inventor (Guido Van Rossum) of Python is a big fan of British comedy movie "Monty Python Flying Circus".

* Python is an object oriented Programming Language: it uses object created from classes.

(A class defines contain similar objects of a particular type)

Versions of Python:

1) Python 1.0 - January 1994
2) Python 2.0 - October 2000
3) Python 3.0 - December 2008
4) Python 3.10.6 - August 2022 (latest version)

(NOTE: Both are two versions, is not compatible with other)
But Python 3 Provides a tool that can convert Python 2 code syntax of the Python 3 and that can be used.

Important Question:

Ques. What are various versions of Python, who invented Python and How was it name given

Ques. On which language Python is based on
Ans. It is based on C Program

Ques. Basic Benefits of Python?
Ans 1) used for statistical mean, median mode of large statistical data
2) helps in graphic representation (Pie chart, graph, histogram)
3) Matrix Multiplication like complex codes are also used i.e [c = a.b] & no need for loops

Ques. Who designed Python logo and from which concept he was inspired?

Ans. Python logo was designed by "Jim Parkin"
it was based on ancient Mayan drawing where a snake has a big head and shortened tail and structure of snake is for meeting of the snake representing

Ques. Explain Features of Python?

Ans. Various features of Python:
1) Software Quality:
At start, Python code is clean, easy to read and to
These factors helps in code reusability and also "Maintainability"
And Python has strong re-use Mechanisms such as OOPs

4) Developer Productivity:
Python enhances developer productivity as compared to other programming language.
As its code is one-third to one-fifth size of equivalent code written in any another language.
Hence less To Type, debug, maintain.
Moreover, python programs run immediately without compile & link overhead.

5) Program Portability:
Python Programs run across the platforms. ie if we have to run script code, written on Linux platform, we can run same code on windows platform we just need to copy & Paste.

6) Support libraries:
it comes with vast collection of pre-built functionality called standard library.
These libraries supports various functions & Tools like for website construction, network programming, serial port access, game development and much more.

5) Component Integration:
There are various integration mechanisms present in python which lets Python scripts to communicate with other parts of an application.
for instance, from python code we can invoke C and C++ libraries. etc

P.T.O

6) Interpreted language:

Python is interpreted programming language which means program is being executed by reading it instruction by instruction.

7) Dynamic and strong Typed nature:

Python is both a strongly Typed and a dynamically Typed language

Strong Typing means that variables do have a Type (integer, float and string etc) And that Type matters when performing operations on a variable.

Dynamic Typing means that the Type of variables is determined only during run time.

Coding standard in Python:

* All the letters in a variable name should be in lowercase.
* When there are more than two words in variable name, underscore can be used.
* Limit all line to a maximum of 79 characters.
* Always surround binary operator with space on either side. Eg: a = a + 1.

operator that is operated on two operands

* A function and class should be separated by 2 blank lines

And Methods within classes should be separated by single blank line

* To render a comment, start sentence or line of code with #

* To render a Multiline comment (or Doc strings) we use """ Text to be commented """ it can be Multiline.

* Python indentation: if a line of code is under same condition it is written with same indentation

eg: if 5 > 2 :
    print ("Five is greater than two!")

if we don't add colon (:) most with condition statement it would throw indentation error.

Indentation space.

Python data Types:

1) Numeric Types: int, float, complex
2) Sequence Types: list, tuple, range
3) Mapping Type: Dict
4) Set Types: set, frozenset
5) Boolean Type: Bool
6) Binary Type: Bytes, Bytearray, memoryview
7) x = "Hello world" : str
8) x = 20 : int
9) x = 20.5 : float
10) x = 1j : complex
11) x = ["apple", "banana", "cherry"] : list
12) x = ("apple", "banana", "cherry") : tuple
13) x = range (6) : range
14) x = {"name": "John", "age": 36} : dict
15) x = {"apple", "banana", "cherry"} : set
16) x = frozenset ({"apple", "banana", "cherry"}) : frozenset
17) x = True : bool
18) x = b"Hello" : bytes
19) x = bytearray (5) : bytearray
20) x = memoryview (bytes (5)) : memoryview

Python Basics:

Operation	output	
2+3	5	(Addition)
7-5	1	(Subtraction)
4*6	24	(Multiplication)
8/4	2.0	(/ → it represents float division)
5/2	2.5	
5//2	2	(// → it represents integer division)
3+7-10	7	(Multiple operations)
8+2*3	14	(it follows BODMAS)
(8+2)*3	30	
2**3	8	(** - represents Power function)
10%3	1	(remainder of Modulus)

Various Types of operators:

1) **Arithmetic operator:** + → addition
- → Subtraction
* → Multiplication
/ → float division
% → Modulus
** → Power
// → Integer division

2) **Assignment operator:** =, +=, -=, *=, /=, %=, **=, //=

(NOTE: Python does not have ++ and -- operator.)

3) **Relational operator:** it gives output in boolean i.e. True / false

>, >=, <, <=, ==, !=

4) **Logical operators:** (or Boolean operators)
And, or, not.
it also gives output in True & False
But it is basically used as Condition statement.

5. **Bitwise operator:**
~ - (Bitwise Not) if a = 10 (0000 1010) output: -11 (1111 0101)
& - (Bitwise And)
| - (Bitwise OR)
^ - (Bitwise xOR)
<< - (Bitwise left shift)
>> - (Bitwise right shift)

6) **Membership operator:**
in - Returns True, if an item is found in the specified sequence
not in - Returns True, if an item is not found in the specified sequence

7) **Identity operator:**
it helps to get Memory location ID.
it is used to compare the memory location of two objects - [id() is used as syntax].

Programs:

WAP To print message:

This symbol is used to represent comments

To Print a message Just use double quotes

```
print ("Python Programming")
```

output: Python Programming

If we use single quote:

```
print ('Python Programming')
```

output: Python Programming

If we using Triple quote:

(As otherwise it would become comment)

```
print ('''"abc"''')
```

output: "abc"

WAP To find Sum of Two no's

```
a = 10
b = 20
c = a + b
print ("value of c is", c)
```

If we put some extra spaces then it would give indentation error. As it would mean that print statement is part of above c statement which is not true.

Here there is no need for declaration of variable or termination using semi colon etc.

No data type required during default value is Program. (i.e value entered directly in itself.)

P.T.O..

WAP To find sum of two numbers by entering values runtime:

```
a = input ("Enter the value of a")
b = input ("Enter the value of b")
c = a + b
Print ("The value of c is", c)
```

output: Enter the value of a 10
" " " " b 10
The value of c is 10 10

Input function is used to enter values during run time.

This would perform Concatination.

★

```
a = int (input ("Enter value of A = "))
b = int (input (" Enter value of B = "))
c = a + b
print ("The value of c is", c)
```

output: Enter value of A = 10
" " " B = 10
The value of c is 20

Eval function: we can use this function instead of int, float. This is an expression argument and it converts datatype according to users input whether it is int, float, char.

WAP To find Area of circle.

```
r = eval (input (" Enter the radius - "))
Area = r * r * 3.14
print ("Area of circle =", Area)
```

NOTE: here in Python there are No keywords like Pi.

Working with various commands:

1) Sizeof Command: It tells about size of data

* WAP to show Execution of Sizeof Command:

```
Import sys
a = 20
print (sys.getsizeof(a))
```

Output: 28

with the help of sys function we can use various libraries of python

Syntax: Import sys

* WAP to find out size of string.

```
Import sys
a = "20"
print (sys.getsizeof(a))
```

Output: 51

Note: The size of a data type is directly proportional to the entered data and is calculated by computer itself.

* WAP to find out size of float variable:

```
Import sys
a = 20.7
print(sys.getsizeof(a))
```

Output: 24

* PTO..

b) Type function: This function is used to check data type.

* WAP to find Data Type of various data elements:

a)
```
a = 20
print (type (a))
```
output : <class 'int'>

b)
```
b = 20.7
print (type (b))
```
output : <class 'float'>

c)
```
b = "20.7"
print (type (b))
```
output : <class 'string'>

Triple quote is used for printing a paragraph or Multi-Line string. i.e """ """.

d)
```
b = 20
print (type (c > 2))
```
output : <class 'bool'>

e) Id function: it shows the address of variable where it is stored.

* WAP to show execution of ID function..

*
```
a = 20
print (id (a))
```
output: 2209058905

*
```
b = 30
print (id (b))
```
output: 1804735311056

*
```
c = 30
print (id (c))
```
output: 1804735311056

∴ NOTE: In python if 2 variable store same value then their address is

These both shows same address as constant 30 is stored at only 1 location & its value is assigned to or is pointing at same value.

NOTE: we don't use curly braces in python. Therefore to write multiple statements we use indentation.

i.e to write exactly under it eg Python welcome

And if not then it would show indentation error

eg
```
print ("python Programming")
print ("welcome")
```

output : python Programming
welcome

NOTE: ★ we use space or Tab for loops or condition statement.

As there print or other statement are part of those decision.

★ In python each control statement is ended with colon symbol

eg: To show an Extension where error does not occur

```
a = 30
if (a > 3):
    print ("welcome")
```

output : welcome.

M T W T F S

Page No.

Date

YOUVA

* Division variation :

a) $z = \frac{10}{6}$ or 10/6 (Single slash shows floating value)

print (z)

output: (welcome) 1.666667

b) z = 10 // 6 (double slash shows only integer value)

print (z)

output : 1

Modulus operation :

z = 10 % 6

print (z)

output : 4

* Power function or Exponential function

z = 2 ** 3 # 2 astric symbol represents power

print (z)

output : 8

* Precedence (or priority)

z = 5 + 6 * 7 + 6/3

print (z)

output : 490

Flow control statement:

These statements basically controlling flow of program

* Types: 1) Conditional statement: if, if.. else, Nested if, elif
2) Looping statement: For, while, Nested loop.
3) Jumping statement: Pass, break, Continue.

1) Conditional statement: A program can decide which part to be executed. And this conditional statement helps in decision making.

* If statement: it is used for decision making when we have single Alternative.

* WAP To show execution of If statement...

```
a= eval (input (" Enter any Number "))
if (a >0):
    print (" Entered No. is +ve ")
```

Output: Enter any Number → 93
Entered No. is +ve.

Control statement are always terminated using colon.
if we don't use colon then it would show indentation error

* If-else statement: It is used when we have to choose between Two Alternatives.

* WAP To if-show execution of if else:

```
a = eval (input (" Enter any Number "))
if (a > 0):
    print (" Entered No. is +ve)
else:
    print (" Entered No. is -ve)
```

* WAP to check whether a no. is even or odd

```
a = eval (input ("Enter Any No."))
if (a % 2 == 0):
    print ("Entered No. is Even")
else:
    print ("Entered No. is odd")
```

Nested - if statement : it is also known as Multi-way or if-elif-else.

when one "if statement" can be placed inside another if statement to form nested if statement.

i.e if within if.

* WAT to find greater b/w Three No.'s:

```
a = eval (input ("Enter first No."))
b = "  (  "  (  "  2nd  "  )
c = "  (  "  (  "  3rd  "  )
if (a > b):
    if (a > c):
        print ("a is greatest among Three")
    else:
        print ("c is greatest among Three")
else:
    if (b > c):
        print ("b is greatest among Three")
    else:
        print ("c is greatest among Three")
```

* WAP to swap two no.

```
a = eval (input (" enter 1st No. "))
b = eval (input (" enter 2nd No. "))
temp = a
a = b
b = temp
print (" value after swap ")
print (" value of a becomes =>", a)
print (" value of b becomes =>", b)
```

* WAP to find square root of number:

```
a = eval (input (" enter no. whoose square ro
a ** 1/2
```

+ By applying pow function

```
print (" value of a is ", a)
```

Or

```
import math
a = eval (input (" enter a. "))
sq = math.sqrt (a)
print (" The square root ", sq)
```

Or

```
a = eval (input (" enter a. "))
sq = pow (a, 0.5)
print (" The square root :", sq)
```

P.T.O

Single line IF / Ternary operator :→

<var> = <val 1> if <condition> else <vale>

```
food = input (" food ; ")
eat = "yes" if food = "cake" else "no"
```

food = input ("food: ") (or)
print ("sweet") if food == "cake" or food == "sweet"

age = int (input ("age: "))
vote = ("yes", "no") [age >= 18]

if... elif.. else statement: This conditional statement is used when we have multiple alternatives and we have to choose 1.

WAP To check if entered alphabet is vowel or not:

```
ab = input ("enter the alphabet")

if (ab == 'a' OR ab == 'A'):
    print ("entered Alphabet is a vowel")
elif (ab == 'e' OR ab == 'E'):
    print ("entered Alphabet is a vowel")
elif (ab == 'i' OR ab == 'I'):
    print ("entered Alphabet is a vowel")
elif (ab == 'o' OR ab == 'O'):
    print ("entered Alphabet is a vowel")
elif (ab == 'u' OR ab == 'U'):
    print ("entered Alphabet is a vowel")
else:
    print ("entered alphabet is a consonant")
```

WAP To print Day of week using if - elif - else

```
Day = int (input ("enter the day No. of the week"))
    if (Day == 0):
        print ("Sunday")
    elif (Day == 1):
        print ("Monday")
        :
        :
    else:
        print ("Entered day no. is not valid")
```

Random Numbers:

We can use random No's using "Import library". i.e To generate random no. and by default it shows data in float data type.

Program To create random No:

```
Import random
print (random.random())
```

output : 0.610619865144

WAP To print Memory location of any random Number:

we check memory location with help of id function

```
import random
a = random.random()
print (id(a))
```

output : 2051847171806

WAP to print Random no. as integer:

```
import random
print (random.randint(1,10)   # here (1,10) stands for starting & end value of Integer. i.e random no. would be choosed in b/w them.
```

Output : 7

WAP To print Random no. as float:

```
print (random.uniform(10.6, 30.7))
```

output: 15.998477 6979

WAP To print Seed Value:
(Permanent value)

```
Import random
Random.seed(10)
print(random.random())
print(random.random())
```

Output: 0.5714059468899135

As time in our eg. even seed(10) value is 0.5714059468899135. And it would be same for everyone in every system

But if we use same command again then seed value will come different. i.e 2nd seed value or i.e fixed 2nd seed value of (10). And 1st seed value, 2nd seed value & so on are fixed for each specific no.

WAP To Compute & display distance b/w Two points:

A(x_1, y_1)
B(x_2, y_2)
$x_1 = 20$
$x_2 = 10$
$y_1 = 15$
$y_2 = 60$

$$d = [(x_2 - x_1) ** 2 + (y_2 - y_1) ** 2] ** \frac{1}{2}$$

```
print("distance between Two Points is :", d)
```

Output: 46.0977223

2) Looping Statements:

A loop can be used To Tell a program To execute a statement multiple number of Times.

WAP To print "python Programming" using while loop:

```
n = eval(input("Enter the value of n:"))
count = 0
while (count < n):
    print("Python Programming")
    count = count + 1
```

WAP to print a series 1 to 10 using while loop.

```
count = 1
while (count <= 10):
    print (count)
    count += 1
```

WAP to find sum of the series 1 to 10 using while loop.

```
count = 1
sum = 0
while (count <= 10)
    sum += count
    count = count + 1
print ("The value of sum", sum)
```

Output: The value of sum 55

WAP to check no. is armstrong or not:

```
n = int (input ("Enter a number"))
arm = 0
temp = n
while (temp > 0):
    d = temp % 10
    temp = temp // 10
    arm = arm + (d ** 3)
if (arm == n):
    print ("entered number is Armstrong")
else:
    print ("entered number is not Armstrong")
```

Output: Enter a number 371
Entered number is Armstrong

Ques:

WAP To Print Armstrong Numbers in b/w 1 to 1000:

```
i = 100
while (i <= 1000):
    arm = 0
    temp = i
    while (temp > 0):
        d = temp % 10
        temp = temp // 10
        arm = arm + (d**3)
    if (arm == i):
        print (arm)
    i = i + 1
```

Ques:

WAP To Print Palindrome No.

```
a = eval (input ("enter a number to check whether entered number is
                 Palindrome or not =>"))
temp = a          # Creating a temporary variable to store value of a
i = -1            # here i is used to represent the powers, where we want to
                  # create reverse of number. ∴ if 327 here digit = 3
                  # for 100 we find 10's power = 2
                  # here we starting on but i = -1
                  # so that we get 1 less digit
new_num = 0
while (a > 0):
    a = a // 10
    i += 1
a = temp          # As value of a changed in previous transaction, ∴ we
                  # re-initialize original value
while (a > 0):
    m = a % 10
    new_num = new_num + m * (10 ** i)   # Creating reverse of a number
    i -= 1                              # reducing the value as digits of
    a //= 10                            # number's no. is reduced.
                                        # As hundreds → tens → ones.
print ("The reverse of given is ", new_num)
if (temp == new_num):
    print ("entered number is palindrome")
else:
    print ("entered number is not palindrome")
```

11 WAP to print a number of digits in a number

```
a = eval(input("enter any number"))
no. of digits = 0
while (a>0):
    a = a//10
    no_of_digit += 1
print("Number of digits:", no_of_digits)
```

WAP to print a number in reverse order

```
a = eval(input("enter a number"))
while (a>0):
    n = a % 10
    print(n, end=' '),      # (comma is used after print statement if we want to print values in same line)
    a = a//10
```

use any end=' ' to print elements in same line

Output: Enter a number : 987
9 8 7

11 WAP to check entered no. is prime or not.

```
a = int(input("Enter a number:"))
fact = 0
for i in range(1, a+1):
    if (a % i == 0):
        fact += 1
if (fact == 2):
    print("Entered number is prime")
else:
    print("entered number is not prime")
```

WAP To Print First 1000 Prime Numbers

```
K = 1
while (K <= 100):
    fact = 0
    for i in range (1, K+1):
        if (K % i == 0):
            fact += 1
    if (fact == 2):
        print (K, end = ' ')
    K += 1
```

Various Looping statements are:

a) While loop with else:

```
Count = 0
while (Count < 3):
    Count = Count + 1
    Print ("Hello world")
else:
    Print ("Welcome")
```

OUTPUT:

Hello world
Hello world
Hello world
welcome

WAP To print series 1 to 10 & when message goes fail Print Python Programm:

```
Count = 1
while (count <= 10):
    print (count)
    count += 1
else:
    print ("Python Programming")
```

i) For loop statement:

Starting value, Ending value, Step value

Syntax 1: for <Var> in range (Val1, Val2, Val3):

eg: for i in range (1, 10, 2)

Syntax 2: for <Var> in <sequence>:

eg: for i in list1:

WAP to print a series using for loop:

```
for i in range (10):
    print (i)
```

output: 0 to 9. i.e 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

* Extension: `for i in range (2, 10):` # By default there is increment by 1

```
    print (i)
```

output: 2, 3, 4, 5, 6, 7, 8, 9

* Extension: `for i in range (2, 10, 2):` (start val, end val, increment step)

```
    print (i)
```

output: 2, 4, 6, 8

* Extension: `for i in range (10, 1, -2):`

```
    print (i)
```

output: 10, 8, 6, 4, 2

WAP to find the sum of the series 1 to 10 using for loop:

```
sum = 0
for i range (1, 11):
    sum = sum + i
print (sum)
```

M T W T F S S

Page No:

Date

YOUVA

WAP To execute the format $1^2 + \frac{2^2}{2} + \frac{3^2}{3} + \ldots + \frac{n^2}{n}$

```
n = eval(input("Enter the value till which you want to execute"))
i = 1
sum = 0
while (i <= n):
    sum = sum + (i * i) / i
print(sum)
```

Output: n = 3

6

Nested for loop:

```
for i in range(5):
    for j in range(5):
        print((i*j), end=' ')
    print(' ')
```

output:

0 0 0 0 0

0 1 2 3 4

0 2 4 6 8

0 3 6 9 12

0 4 8 12 16

c) Jumping statement:

a) break b) Continue c) Pass

Break statement: it breaks the flow of control.

```
for i in range(10):
    print(i)
    if (i == 5):
        print("Break")
        break
```

Output: 0 1 2 3 4 5

break

Continue statement: it skips condition statement.

eg:
```
for i in range (6):
    if (i == 5)
        Continue
        print (i)
```

Output: 0
1
2
3
4
6
7
8
9

eg: ~~Diff to continue from~~
Diff b/w Pass & Continue

* Pass: it just checks the statement. it works as an empty statement but

* Continue: it skips the conditional part.

Pass statement: it is used to "Pass" an empty condition statement without any output. there would be no output

eg
```
a = 10
b = 20
if (b > a):
    Pass
```

eg
```
For i in range 'Programming'
    if (i == 'r'):
        pass
    else:
        print (i)
```

Output:
P
o
g
a
m
m
i
n
g

Page No. YOUVA
Date

Q for i in 'Programming': output: P r o g r a m m i n g
    if (i == 'i'):
        continue
    print(i)

Q WAP to print 'hello' using Pass statement:

```
str1 = "hello"
for i in str1:
    if (i == 'h'):
        pass
        print(i)
    elseif (i == 'e'):
        pass
        print(i)
    elseif (i == 'l'):
        pass
        print(i)
    elseif (i == 'o')
        pass
        print(i)
```

output: h
e
l
l
o

String Operations:

① String: Indexing

- A string can be traversed both side (with help of index):

0	1	2	3	4	5	6
W	e	l	c	o	m	e

Eg: if t = "welcome"

t[3] = will give 'c'
t[2] = " " 'l'
t[0] = " " "W"
t[-1] = " " "e"

Basic string operators:

a) String concatenation Operator (+):
The '+' operator creates a new string by joining the operand strings.

eg: "Hello" + "students" # output "Hellostudents"

But "a" + 5 # output: Error as both operands should be string.

b) String replication Operator (*):
The '*' operator when used with string, then it creates a new string that is member of replication of the input string.

eg: "Xy" * 3 will give # output: "XyXyXy"
"3" * 4 will give # output: "3333"

c) Membership operators (in & not in):

a) in: returns True if a character or a substring exists in the given string, "False" otherwise.

eg: "D" in "Rajat" output: True
"Raj" in "Rajat" output: True
"67" in "1234" output: False

b) not in: returns "True" if a character or a sub-string does not exists in the given string, "False" otherwise.

Eg: "a" not in "Rajat" Output: False
"Raj" not in "Rajat" output: False
"67" not in "1234" output: True.

a) ord() function: it gives ASCII value of a single character

eg: ord ("a") Output 97
ord ("z") output: 90
ord ("0") output: 48

b) chr() function: it is opposite of ord() function

eg: chr (97) Output: "a"
chr (90) Output: "Z"
chr (48) output: "0"

c) String Slice:

This function helps to fetch a specific part of string.

i) Let a part of a string containing some contiguous character from the string.

Let word = "WELCOME"

∴ Index =

	W	E	L	C	O	M	E
	0	1	2	3	4	5	6
	-7	-6	-5	-4	-3	-2	-1

	OUTPUT:
word [0:7]	WELCOME
word [0:5]	WEL
word [2:5]	LCO
word [-6:-3]	ELC
word [:5]	WELCO
word [3:]	COME

NOTE: The values are stored and print till index-1. i.e. if word [0:7]. Then we will go till (7-1=6) 6th index.

a) **string.capitalize():**

it gives the copy of string with its first character capitalized

eg. t = "sumit rana" output: "SUMIT RANA"

R = t.capitalize()

Print(R)

b) **string.find(substring [, start [, end]]):**

it returns the lowest index in the string where the substring is found with in the slice range of start to end. returns -1

i.e. if substring not found

eg t = "We are learning strings. strings are used in Python"

x = "strings"

t.find(x)

t.find(x, 20)

t.find(x, 20, 30) # This will search from index 20 to 30

t.find("Hello")

OUTPUT:

14 (index no)

35 (")

-1

-1 (as string not found)

P.T.O

1) String.isalnum():

It gives True, if the characters in the string are alpha numeric (alphabet or numbers) and there is atleast one character, False otherwise.

```
eg: t = "Hello_123"          # Output: True
    t.isalnum()

    t = "Hello 123"          # output: False
    t.isalnum()
```

2) String.isalpha():

It gives True if the characters in the string are alphabet only and there is atleast one character, False otherwise.

```
eg: t = "Hello123"           # OUTPUT: False
    t.isalpha()
    t = "Hello"              # OUTPUT: True
    t.isalpha()
```

3) String.isdigit():

It gives True if the characters in the string are Digits only and there should be atleast one character, False otherwise.

```
eg: t = "Hello123"
    t.isdigit()              # OUTPUT: False
    t = "123"
    t.isdigit()              # OUTPUT: True
```

P.T.O

1) String.islower():
This function gives True, if all letters in given string are lowercase, otherwise it gives false.

eg: if t="SUNDAY"
t.islower() # output False
t="sunday"
t.islower() # output True

2) String.isupper():
This function is opposite of islower()

3) String.isspace():
it gives True if there are only white space characters in the string, false otherwise.

eg: t=" "
t.isspace() # output: True
t="s"
t.isspace() # output: False

4) String.upper():
This function gives a copy of the string converted to uppercase.

eg: t="SUNday12"
x=t.upper() # output SUNDAY12
Print(x)

5) string.lower(): it is just opposite to upper()

6) len(string): This function returns length of string.
eg: Len("TOT BRAND")

2) for loop with string:

eg: for ch in "Hello":
    print (ch)

OUTPUT
H
e
l
l
o

eg: T = "Hello"
for ch in T:
    print (ch)

OUTPUT:
H
e
l
l
o

eg: T = "Hello"
Len = len (T)
for i in range (Len):
    Print (T[i])

OUTPUT:
H
e
l
l
o

eg: T = "Hello"
Len = len (T)
for i range (0, Len):
    Print (T[i])

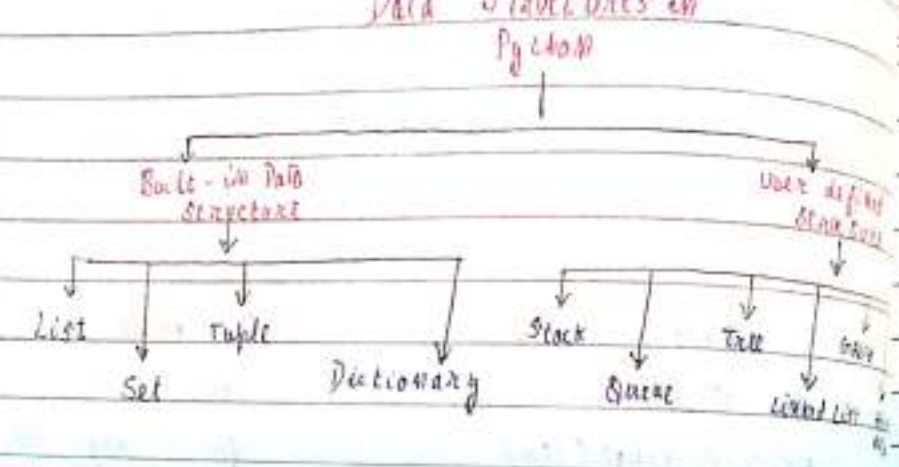
OUTPUT:
H
e
l
l
o

T

Python Data structures

Data structures:

These are coding structures for storing and organizing data. That makes it easier to modify, navigate and access information



1) List:

List is an Ordered collection of items enclosed within "[ ]"

a) A List can store a collection of data of any type
b) It is a sequence defined by the list class.
c) There are addresses assigned to every element of list
d) Elements in List can be accessed through an index
e) Lists are Mutable [i.e we can add elements in it
f) The elements in a list are separated by comma
g) List can contain elements of various data types

* Index Operator [ ]: An element in a list can be accessed through the index

SYNTAX: My list [index]

The list indexes are 0 based, i.e they range from 0 to len(my list) - 1.

eg: my list = [5, 6, 45, 3, ...]

* Functions for Lists:

1) len → if we want to get length of list.
2) Max → if we want to find maximum elements in List
3) Min → if we want to find minimum element in List
4) Sum → if we want to find sum
5) Shuffle → if we want to shuffle the elements in list

WAP To show execution of function for List:

```
list 1 = [2, 3, 4, 1, 32]
print (len (list 1))
print (max (list 1))
print (min (list 1))
print (sum (list 1))

import random
random.shuffle (list 1)      # Shuffle the elements in list
list 1
```

Output: 5
32
1
42
[4, 3, 1, 32, 2]

6) Index operator [ ]:

An element in a list can be accessed through the index operator.

Syntax: myList [index]

List indexes are 0 based; that is they range from 0 to len (myList) - 1

F T D

* Python also allows the use of negative numbers as indexes to refer positions relative to the end of the list. eg: list1 = [2, 3, 5, 2, 33, 21]
list1[-1]
OUTPUT: 21

* The actual position is obtained by adding the length of the list with the negative index

eg: list1 = [2, 3, 5, 2, 33, 21]
list1[-3]

OUTPUT: 2

7) List Slicing [start : end] =>
The index operator allows you to select an element at the specified index.
The slicing o/p returns a slice of the list using the syntax list [start : end].
The slice is a sublist from index start to index end - 1.

eg: list1 = [2, 3, 5, 7, 9, 1]
list1[2:4]
OUTPUT: [5, 7]

eg: list1 = [2, 3, 5, 7, 9, 1]
list1 = [-4 : -1]
OUTPUT: [5, 7, 9]

eg: list1 = [2, 3, 5, 2, 33, 21]
list1[:2]
OUTPUT: [2, 3]

eg: list1 = [2, 3, 5, 2, 33, 21]
list1[3:]
OUTPUT: [2, 33, 21]

eg: list1 = [2, 3, 5, 2, 33, 21]
list1[1:-3]
OUTPUT: [3, 5]

eg: list1 = [2, 3, 5, 2, 33, 21]
list1[-4:-2]
OUTPUT: [5, 2]

5) "+", "*", and "in/not in" O/P:

a) + → list 1 = [1, 4, 6, 7.8, 9]
list 2 = [5, 3, 5, 72, 9]

list 3 = list 1 + list 2
list 3

OUTPUT: [1, 4, 6, 7.8, 9, 5, 3, 5, 72, 9]

Note: Before adding element, the count of both list should be same and only then it concatenates.

b) eg: * → list 4 = 3* list 1
list 4

OUTPUT: [1, 4, 6, 7.8, 9, 1, 4, 6, 7.8, 9, 1, 4, 6, 7.8, 9]

Before adding element the count of both list should be same & only then it concatenates.

c) in → eg: list 1 = [1, 4, 6, 7.8, 9]
4 in list 1

output: True

it gives output either in True or false depending on whether the element is present in list or not.

d) not in → eg: list 1 = [2, 3.5, 2, 33, 21]
2 not in list 1

output: False

it is same like "in O/P". But checks whether mentioned element shouldn't in in list.

1) Comparing Lists: we compare list using the comparison O/P (>, >=, <, <=, = = and !=) and it gives OUTPUT in Boolean.

eg: list 1 = [30, 1, 2, 1, 0]
list 2 = [1, 21, 13]
list 1 != list 2

OUTPUT: True

10) Traversing elements in a for loop:

eg:
```
mylist = [1, 2, 3, 4, 5, 6, 7]
for i in mylist:
    print (i)
```

Output:
```
1
2
3
4
5
6
7
```

11) List Comprehensions:

It provides a concise way to create a sequential list of elements. A list comprehensions consists of brackets containing an expression followed by a for clause, the zero or more for or if clauses.

The list comprehension produces a list with the results from evaluating the expression

eg:
```
List 1 = [x for x in range (5)]
list 1                          OUTPUT: # [0, 1, 2, 3, 4]
List 2 = [0.5 * x for x in list 1]
list 2                          OUTPUT: # [0.0, 0.5, 1.0, 1.5, 2.0]
List 3 = [x for x in list2 if x < 1.5]
list 3                          OUTPUT: # [0.0, 0.5, 1.0]
```

1) List Methods :

a) Append :→ new element is added at end, without affecting other elements.

Eg: List 1 = [1, 4, 6, 8, 1]
List 1. append (20)
List 1

OUTPUT: [1, 4, 6, 8, 9, 20]

b) Count :→ it counts the no. of that element in the list.

Eg: List 1 = [2, 3, 4, 1, 32, 4]
List 1. coount (4)

OutPut: 2 (As '4' element is 2 times in list)

c) Extend :→ it kind of extends the list by concatenating.

Eg: List 1 = [2, 3, 4, 1, 32, 4]
List 2 = [99, 54]
list1. extend (list 2)
list 1
OutPut: [2, 3, 4, 1, 32, 4, 99, 54]

d) index :→ it returns the index of that particular element. And index starts from zero.

And if there Two same elements then By default it would return the index of element at First Place.

eg list 1 = [2, 3, 4, 1, 32, 8]
list 1 . index (4)

output: 2

c) insert :- here we can insert a element at a specific index

eg: list 1 = [2, 3, 4, 1, 32, 8]
list 1 . insert (1, 25)
list 1

output: [2, 25, 3, 4, 1, 32, 8]

d) pop ():- if we haven't mentioned any index then by default it deletes the last element in list.

eg: list 1 = [2, 25, 3, 4, 1, 32, 4, 19, 99, 54]
list 1 . pop()
output: 54

list 1 = [2, 25, 3, 4, 1, 32, 4, 19, 99, 54]
list 1 . pop(2)
output: 3

e) remove :- it is used to remove a specific element in list.

eg list 1 = [2, 25, 3, 4, 1, 32, 4, 19, 99, 54]
list 1 . remove (32)
list 1
output: [2, 25, 3, 4, 1, 4, 19, 99, 54]

NOTE: Difference b/w pop & remove

pop refers to index while removing whereas 'remove' removes directly the element

2) REVERSE :- helps to find reverse of list.

eg list1 = [2, 25, 3, 4, 1, 32, 4, 19, 99, 54]

Ans list1 = [2, 25, 3, 4, 1, 32, 4, 19, 99, 54]
list1.reverse()
list1

Output: [54, 99, 19, 4, 32, 1, 4, 3, 25, 2]

3) sort :- All the elements in list would be sorted in ascending order

eg list1 = [2, 25, 3, 4, 1, 32, 4, 19, 99, 54]
list1.sort()
list1

Output: [1, 2, 3, 4, 4, 19, 25, 32, 54, 99]

To sort in descending order :

eg list1 = [2, 25, 3, 4, 1, 32, 4, 19, 99, 54]
list1.sort(reverse = true)
list1

Output : [99, 54, 32, 25, 19, 4, 4, 3, 2, 1]

i) Product of Two lists:

This method is used to Multiply index element of one list with same index of other list.

eg:
```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
New_list = [ ]

for num1, num2 in zip (list1, list2):
    New_list.append (num1 * num2)
print (Products)
```

Output: [4, 10, 18]

if we want to Multiply whole list with particular No.

eg:
```
my_list = [1, 2, 3, 4, 5]
my_new_list = [i * 5 for i in my_list]
print (my_new_list)
```

Output: [5, 10, 15, 20, 25]

ii) Inputting in list during run time:

eg:
```
l1 = [ ]        # Create a list.
Print ("Enter 5 Numbers")
for i in range (5):
    l1.append (eval (input ("Enter The Numbers")))
print (l1)
```

OUtPut: Enter 5 Numbers:
Enter The Numbers 1
" " " 2
" " " 3
" " " 4
" " " 5
[1, 2, 3, 4, 5]

Traverse The elements with for loop:

```
list 1 = [1, 4, 6, 8, 9]
for i in list 1:
    print(i)
```

OutPut: 1 4 6 8 9

NOTE: Here As we have list given. ∴ we don't have To use range function. As we know its start & Ending.

b) Shuffle:

For Shuffling The list & printing it randomly. we'll Firstly import random function (library). Then we'll call Shuffle Function From it.

Eg:
```
list 1 = [1, 4, 6, 8, 9]
list 1
import random
random.shuffle(list 1)
print(list 1)
```

OUtPut: [1, 4, 6, 8, 9]
[4, 8, 6, 1, 9]

m) **Comparing of two lists:**

it is done with the help of relational operator

eg: list 1 = [2, 4, 6, 7, 8]
list 2 = [3, 4, 6, 1]

OUTPUT:

list1 <= list 2 — True
list 1 == list 2 — False
list 2 >= list 1 — True

n) **To replace a specific list element:**

eg: list 1 = [3, 4, 5, 20, 2]
list 1 [2] = 100 # This would replace the element at 2 with 100.
list 1

OUTPUT: [3, 4, 100, 20, 2]

o) **Copying list:**

eg: list 1 = [67, 89]
Print (id(list 1)) # To Print address of list 1
list 2 = [34, 90]
Print (id(list 2)) # To Print address of list 2

list 1 = list 2 # To copy elements of list 2 in list 1
Print (list 1)
Print (id(list 1)) # After copying its old address has been replaced by address of list 2.

Output: 19 270 58 320
22 66 49 14 31
[34, 90]
22 66 49 14 31

Small Extensions:

if we want to create a copy of list elements.

if we do 2 x list2

or

list2 x 2

These both would create copy of list.

2) Tuples:

Tuples are like lists, but their elements are Fixed.
i.e Once a Tuple is created, we cannot add new elements, delete elements, replace elements, or ~~even~~ we can't even reorder the elements in the Tuple.

a) Tuples is immutable.
b) it is denoted by '( )'.

a) Print elements in Tuple:

eg: Tuple 1 = (1, 3, 5, 6)
Tuple 1
Output: (1, 3, 5, 6)

b) To check nature of Tuples

Tuple 1 = (1, 3, 5, 6)
eg: print (type (tuple 1))

c) How to store Tuple in Square bracket without changing its Primary Nature? (i.e creating Tuple from a list)

tuple 2 = tuple [2, 4, 6, 7, 87]
Eg: print (tuple 2)

d) Count Function in Tuple :

eg: Tuple 1 = (6, 8, 4, 5, 8)
print (Tuple 1. count (8))

Output: 2

e) Rest All operations are same as we did in list :

eg: Tuple 1 = ("green", "red", "blue")
print (tuple 1)
Tuple 2 = tuple ([7, 1, 2, 23, 4, 5])
print (tuple 2)

max function Print ("max is", max (tuple 2))
min function Print ("min is", min (tuple 2))
sum function Print ("sum is", sum (tuple 2))
index operator Print ("The first element is", tuple 2[0])
To print element of Tuple at 0 index

tuple 3 = tuple 1 + tuple 2 # Combining two Tuple
Print (tuple 3)

Duplicate a Tuple tuple 3 = 2 * tuple 1
print (tuple 3)
Slicing operator print (tuple 2 [2:4])
print (tuple 1 [-1])
in operator print (2 in tuple 2)

for v in tuple 1:
print (v, end=' ')
print()

```
List1 = list (tuple2)          # obtain list from a Tuple
List1.sort()                   # checking whether it is extracted
                                 as list or not

tuple4 = tuple (list1)
tuple5 = tuple (list2)
Print (tuple4)

Print (tuple4 == tuple5)       # Comparing Two Tuples
```

Outputs:

```
('green', 'red', 'blue')
(7, 1, 2, 23, 4, 5)
max is 23
min is 1
sum is 42

The First element is 7
('green', 'red', 'blue', 7, 1, 2, 23, 4, 5)
('green', 'red', 'blue', 'green', 'red', 'blue')
(2, 3)
blue
True
green red blue
(1, 2, 4, 5, 7, 23)
True
```

P.T.O of some Examples where we can modify Tuple.

eg: Tuple1 = (6, 8, 4, 5, 8)
tuple1[2] = 50
Output: Error.

(It is trying to change element at index 2.)

eg: Tuple1 = ("green", "red", "blue")
Tuple1.sort()
Output: Error.

3.) Sets:

* Sets are like lists storing a collection of elements.

Unlike lists, however the elements in a set are "non-duplicates" and are not placed in any particular order

* it is denoted by:- '{ }'

* it can contain elements of same type or mixed types. (i.e in one set we can store Multiple data types).

Eg: S = { 1, 2, 3, "one", "two", "three" }

here it is a set that contains members & strings.

* Sets items are unchangeable, But we can remove and add new items.

* Sets cannot have two items with same value.

a.) Creating a Set:

eg: Set 1 = { 1, 5, 7, 9 }

Print (set1)

Output: { 1, 5, 7, 9 }

eg: Set 2 = { 1, 4, 9, 8 }

Print (set2)

Out Put: { 1, 4, 8 } # Duplicate value removed.

P.T.O...

eg: Set3 = {8, 1, 3, 9, 1, 0}
Print (set 3)

OutPut: {0, 1, 3, 8, 9} # here by default the element are stored in ascending order

eg: Set1 = ("green", "blue", "black", "orange"}
Set1

OutPut: { "black", "blue", "green", "orange"} # arranged in alphabetical order

b) To insert element in list:

eg: S_1 = {1, 4, 6}
S_1 add (7)
S_1

OutPut: {1, 4, 6, 7}

if we want to insert element at starting, we can't do it as output would be in sorted manner. so we can't deliberately decide position to enter

c) To remove element, Find length, Max, Min, Sum:

eg: S_1 = {1, 4, 6, 7}
len (S_1)
Max (S_1)
Min (S_1)
Sum (S_1)
S_1.remove (4)

OutPut: 4
7
1
18
{1, 6, 7}

d) To add strings:

eg: S1 = {1, 4, 6}

S1 add ("Python")

S1

Output: {1, 4, 6, 'Python'}

if we want to add two strings:

eg S1 = {1, 4, 6}

S1 add ("Python", "Programming")

S1

Output: Error

∴ To add more than 1 element we add it as list

e.g S1 = {1, 4, 6}

S1 update (["Blue", "Python"])

Output: {1, 4, 6, 'Python', 'Blue'}

e) "in" function: returns True if element is in S1 otherwise it returns False.

eg: S1 = {1, 2, 4}

3 in S1

Output: False

f) Equality Test: This test is used to == or != operator to test if two sets contain the same elements.

eg: S1 = {1, 2, 4}

S2 = {1, 4, 2}

S1 == S2

S1 != S2

Output: True

False

a) Union of two sets:

eg: $S_1 = \{1, 2, 4\}$

$S_2 = \{1, 3, 5\}$

S_1 union (S_2) # or $S_1 \mid S_2$ — using operator

↳ Pipeline operator

Output: $\{1, 2, 3, 4, 5\}$

b) intersection of two sets:

eg: $S_1 = \{1, 2, 4\}$

$S_2 = \{1, 3, 5\}$

S_1 intersection (S_2) # or $S_1 \& S_2$ — using operator

↳ Ampersand operator

Output: $\{1\}$

c) Difference of two sets: it is used like Set1 & Set2 Then it would return a set that contain elements in set1 but not in Set2

eg: $S_1 = \{1, 2, 4\}$

$S_2 = \{1, 3, 5\}$

S_1. difference (S_2) # or $S_1 - S_2$ — using operator

Output: $\{2, 4\}$

d) Symmetric Difference (^): it is used to display un-common elements. it is also known as "exclusive or".

eg: $S_1 = \{1, 2, 4\}$

$S_2 = \{1, 3, 5\}$

S_1. symmetric_difference (S_2) # or $S_1 \wedge S_2$ — using operator

Output: $\{2, 3, 4, 5\}$

k) Obtaining List from a set:

```
Set2 = {7, 1, 2, 23}
List1 = list(set2)
Print list1
```

Output: [7, 1, 2, 23]

Some basic Programs based on these Data Structures

WAP To find sum of cube of First n Natural No:

```
n = eval(input("Enter the value of n: "))
Sum = 0
For i in range(0, n):
    Sum = i ** 3
Print("The sum is :-", sum)
```

Or

```
import math
sum = 0
n = eval(input("Enter the value of n :- "))
    for i in range(1, n+1):
        sum = sum + math.Pow(i, 3)
    print("The sum is :-", sum)
```

Output: Enter the value of n:- 3
The sum is :- 36.0

P.T.O

In a list elements are list1 = [4, 7, 1, 10, 16, 2, 1, 5], print the even numbers and in different list and odd elements in another list

```
list1 = [4, 7, 1, 10, 16, 2, 1, 5]
list2 = [ ]
list3 = [ ]

for i in list1:
    if i / 2 = 0
        list2.append(i)
    else
        list3.append(i)

print("even no's in new list[2] are", list2)
print("odd no's in new list[23] are", list3)
```

Output: [4, 10, 16, 2]
[7, 1, 1, 5]

Write to remove each element from the list, which are divisible by 2.

```
list1 = [4, 7, 1, 10, 16, 1, 2, 5]
list2 = [ ]
for i in list1:
    if i / 2 = = 0
        list2.append(i)
for i in list2:
    if i / 2 = = 0
        list1.remove(i)
list1[ ]
```

Output:
[7, 1, 1, 5]

WAP to print -ve no. in the given list:

[4, -7, 9, -10, 16, -1, 25, -64]

```
list 1 = [4, -7, 9, -10, 16, -1, 25, -64]
list 2 = [ ]

for i in list 1:
    if i < 0:
        list 2 append (i)
print (list 2)
```

Output:
-7, -10, -1, -64

WAP to find size of a tuple:

```
t = (1, 2, 3, 4, 5, 6, 7)
s = len (t)
print (s)
```

Output: 7

To Take Multiple inputs in single line:

```
a, b, c, d = eval (input ("Enter the value"))
```

Ques. Difference b/w Pop and Remove.

Ans. Pop is used to remove on the basis of index

Remove is used to remove that particular element in the list.

P.T.O...

4) **Dictionary:**

A dictionary is a container object that stores a collection of key/value pairs.

it enables fast retrieval, deletion and updating of values by using the key.

Here the keys are like an index operator. A Dictionary cannot contain Duplicate keys. (ie Each key maps to one value).

Dictionary is mutable

a) Creating a Dictionary:

Firstly the items are enclosed inside Pair of curly braces ({})

Each item consists of a key, followed by a colon, followed by a value

items are separated by commas.

Syntax: {key: value}

eg: fruit = {"APPLE": 10, "Orange": 20}

fruit

fruit = {"APPLE": 10, "orange": 20}

type (fruit)

fruit.keys()

fruit.values()

Output: {'APPLE': 10, 'Orange': 20}

dict

dict-keys (['APPLE', 'Orange'])

dict-values ([10, 20])

b) Adding a new element

eg: fruit = {"APPLE": 10, "Orange": 20}
fruit['Mango'] = 50
fruit

Output: {'APPLE': 10, 'ORANGE': 20, 'MANGO': 50}

c) Changing an existing element

eg: fruit = {"APPLE": 10, "ORANGE": 20}
fruit['Mango'] = 500
fruit

Output: {'APPLE': 10, 'ORange': 20, 'Mango': 500}

d) Updating Dictionary:

eg: fruit 1 = {"APPLE": 10, "Orange": 20}
fruit 2 = {"Banana": 30, "Guava": 40}
fruit1.update(fruit2)
fruit 1

OutPut: {'APPLE': 10, 'ORANGE': 20, 'BANANA': 30, 'GUAVA': 40}

e) Poping an element

eg: fruit 1 = {"Apple": 10, "Orange": 20}
fruit1.pop("Orange")
fruit1

Output: {'APPLE': 10}

f) clear command:

eg: fruit1 = {"APPLE": 10, "ORANGE": 20}
fruit1.clear()
fruit1

Output: {}

Difference b/w list, Tuple, set and dictionary:

	List	Tuple	Set	Dictionary
a)	List is a non-homogeneous data structure that stores the element in single row and Multiple rows and columns.	Tuple is a non-homogeneous data structure that stores single row and multiple rows and columns.	Set is also non-homogeneous data structure But stores in single row.	Dictionary is also non-homogeneous data structure which stores key value pairs
b)	List can be represented by [].	Tuple can be represented by ().	Set can be represented by {}.	Dictionary can be represented by {}.
c)	it allows Duplicate elements.	it allows Duplicate elements	it doesn't allow Duplicate elements.	Dictionary doesn't allow Duplicate keys.
d)	can be created using list() function	can be created using tuple() func	can be created using set() func.	can be created using dict()
e)	it is mutable ie changes can be made.	it is immutable	it is mutable But elements are not Duplicated.	It is mutable. But keys are not duplicated
f)	Creating empty list → list[]	t = ()	a = set()	d = {}

Function:

Function Provide a way to break problems or processes down into smaller and independent blocks of code

it can be used to define reusable code

Function is a block of code which Performs a specific Task

Types of functions:

Python support two types of Function:

a) **Built in function:**

The function which come along with Python itself are called a built in function or Predefined function.

eg: range(), id(), type(), print(), eval(), input() etc

b) **User-defined function:**

Functions which are created by Programmer explicitly according to the requirement are called a user-defined function.

Defining a function:

A function definition consists of function's name, parameters and body.

Syntax: def functionName (list of Parameters):

a) def: def is a Keyword for defining a function

b) Function Name: Function name is the name of the function that can be give any name to function.

c) List of Parameters: Arguments through which we pass values to a function.

d) Colon: To mark the end of function header

e) Calling a Function: it executes the code in the function

To call a function, use the name of the function with the Parenthesis.

(And if function accepts parameters, then pass those parameters in the Parenthesis. (it returns control to the caller when return statement is executed or func. is finished)

f) Return statement:

The return statement is used to exit a function and go back to the place from "where it was called."

Syntax: return [expression-list]

eg:

```
def max(num1, num2):
    if num1 > num2:
        result = num1
    else:
        result = num2
    return result
```

→ Defining of a function.

```
x = eval(input("enter 1st No."))
y = eval(input(" " 2nd No"))
z = max(x,y) # calling of function
print("Greater No", z)
```

g) Formal Parameters: The variables in the function header are known as formal Parameters or simply Parameters.

h) Actual Parameters: When function is invoked, you pass a value to the Parameter. The value is refferid to as an actual Parameter or argument.

Important Concept:

Difference b/w infinite loop & self calling function in Python:

★
```
while True:
    print ("Python Programming")
```

here It would Print the output infinite No of times As loop is never ending as without counter

★
```
def message():
    Print ("Python Programming")
    message()
message()
```

Output: it would print around 3000 times, (or depending on compiler).

But we can also increase limit.

Some Programs:

WAP TO add & find sum of three numbers using Python function:

```
def add():                                  # function Definition
    a,b,c = eval(input("Enter Numbers"))    # function body
    d = a+b+c
    print("Sum is", d)
add()                                       # function calling
```

Scope and Lifetime of Variables:

Scope of a variable is the portion of a Program where the variable is recognised.

* Parameters and variables defined inside a function are not visible from outside the function. Hence, they have a "local scope".

* life time of a variable: it is the period throughout which the variable exists in the memory. The lifetime of variables inside a function is as long as the function executes.

They are destroyed once we return from the function. Hence, a function does not remember the value of variable from its previous calls.

* A Global variable is a variable that declares outside of the function. The scope of a variable is broad. it is accessible in all functions of the same Module.

Example of local and global variables:

eg:

```
def func():
    x = 20                          # Local variable
    Print ("value inside function :", x)
x = 100
func()
print ("value outside function:", x)
```

output:
value inside function: 20
value outside function: 100

eg:

```
global_var = 5                      # Global Variable

def function 1():
    Print ("value in 1st function:", global_var)

def function 2():
    global_var = 555                # Modify global variable
                                    # function will treat it as a
                                      local variable
    print ("value in 2nd function:", global_var)

def function 3():
    print ("Value in 3rd Function:", global_var)

function 1()
function 2()
function 3()
```

Output:
Value in 1st function: 5
Value in 2nd function: 555
Value in ...

Docstrings: These are strings used right after the definition of a function, method, class or Module.

They are used to document the code. This function is used to print Multi line comment in output.

Just declare docstring in next line of function Declaration

* To access the docstrings using the --doc-- attributes

eg:
```
def ave (a, b):
    ''' This is a function which calculate average of two Number '''
    average = (a+b)/2
    print (average)
ave (5, 5)
print (ave.__doc__)
```

Output: 5.5
This is a function which calculate average of two Numbers.

Difference b/w Comments & Docstrings:

Comments	Docstrings
* Comments explain code in simple language	Docstring describes what the code does.
* In Python, we can write comments using '#'.	we can write docstrings as string literals using Triple quotes (''' ''')
* eg: # This is a comment	''' This is a Docstring'''
* There is no restriction on where to write a comment in Python.	we can write docstrings only for modules, classes, function and methods in Python.
* Comments can not be accessed from the Python Shell.	we can access docstring using either the --doc-- attribute or help() Function.

Return Values from a functions

In Python, to return value from the function, a return statement is used. it returns the value of expression following the return keyword the return statement should be inside of the function block

eg:
```
def is_even(list1)
    even_num = []
    for n in list1
        if n % 2 == 0:
            even_num.append(n)
    return even_num        # return a list

even_num = is_even([2,3,42,51,62,70,5,97])
                           # Passing list to the function
print("Even numbers are:", even_num)
```

Output: Even numbers are: [2, 42, 62, 70]

☆ To Return Multiple Values :

if return multiple values from a function, use the return statement by seperating each expression by a comma.

T.I.D for Programm.

eg: def arithmetic (num1, num2):
    add = num1 + num2
    Sub = num1 - num2
    Multiply = num1 * num2
    division = num1 / num2

    return add, sab, multiply, division # return four values

a, b, c, d = arithmetic (10, 2) # read four return values in four variable

print ("Addition: ", a)
print ("Subtraction: ", b)
print ("Multiplication: ", c)
print ("Division: ", d)

OUTPUT: Addition: 12
Subtraction: 8
Multiplication: 20
Division: 5.0

Pass statement in function:
When the interpreter finds a pass statement in the Program. it return no operation

eg: def addition (num1, num2):
    # Imple mentation of addition function in coming release
    Pass # Pass statement
addition (10, 2)

P T O ...

a) Default Arguments

A function can have default values in Python.

eg:
```
def message (name = "Guest"):
    print ("Hello", name)
message ("John")      # calling function with argument
message ()            # Calling function without argument
```

Output: Hello John
Hello Guest

eg:
```
def add (a=10, b=20)
    c = a+b
    print (c)
add()                 # without argument:
                      # it would take default values
add (20, 12)          # with two arguments:
                      # it overwrites previous value
add (100)             # it overwrites the value of a.
                      # b would be taken 20.
add (b=12, a=20)      # if we want to take b's
                      # value 1st & a's second.
```

output: 30
32
120
32

b) Keyword Arguments:

Keyword Argument are used to give values for specific Parameters.

P.T for eg..

eg: def Print_Area (width = 1, height = 2) # Default argument values
area = width * height
print ("width ", width, "\t height ", height, "\t area ", area)

\t used to give a space of Tab. in the output

Print_Area()
Print_Area (height = 6.2) # keyword Arguments... Default width = 1
Print_Area (width = 6.2) # Default height = 2

Output: width: 1 height: 2 area: 2
width: 1 height: 6.2 area: 6.2
width: 6.2 height: 2 area: 12.4

eg: def add (a=10, b=20)
c = a+b
print (a)
print (c)
print (b)
add (b=12, a=20) # if we want to enter b first, we can use keyword argument as it would overwrite previous values by new one.

c) Arbitrary Arguments:
Sometimes, we do not know in advance the number of arguments that will be passed into a function.

here we use asterisk (*) before the parameter name to denote this kind of argument.

eg: def addition (*numbers)
    total = 0
    for no in numbers:
        total = total + no
    Print ("Sum is :", total)
addition () # Zero no. argument
addition (10, 5, 2, 5, 4) # 5 no of argument
addition (78, 7, 2, 5) # 5 no of argument

Output: Sum is : 0
Sum is : 26
Sum is : 87.5

eg: def add (*numbers):
    for i in numbers:
        print (i)
add (1, 3)
add (5, 2, 7, 9, 8, 12)
add (3)

Output:
1 3
5 2
7
9
8
12
3

Can we pass normal argument as Arbitary Arguments:

def add (a, *numbers):
    print a
    print (i)
add (1, 3)

output: 1 => NOTE: This default value would be stored in a.
3

But Why we can't do def add (*number, a)
(This would give error as, The arbitary would take all elements into it itself, as we can pass as many in it)

Some Programs:

WAP To find Area of rectangle with function & default Arguments:

```
def rectangle (l=10, b=20):
    Area = l x b
    Print ("Area =" Area)
rectangle ()
```

WAP To implement Arbitary Argument:

```
def Implementation ( *number):
    for i in number:
        Print (i)
Implementation (5)
```

WAP To find Area of rectangle with runtime value:

```
a,b = Eval (input ("Enter length, breadth")
def rectangle (a,b):
    Area = a x b
    Print (Area)
rectangle ()
```

Recursive function:

A recursive function is a function that calls itself, again and again.

Consider calculating Factorial of a no.

Output: 120

eg:

```
def factorial (no):
    if no == 0:
        return 1
    else
        return no * factorial (no - 1)
Print ("factorial of a number is : ", factorial (5))
```

Execution would be:

5 * factorial(4)

5 * 4 * factorial (3)

5 * 4 * 3 * factorial(2)

5 * 4 * 3 * 2 * factorial(1)

5 * 4 * 3 * 2 * 1 = 120

Lambda functions

* it is a Pre-defined function. In Python an anonymous function is a function. That is defined without a name.

* while normal functions are defined using the def keyword in Python, Anonymous function are defined using lambda keyword. ie without function name.
(Hence, anonymous functions are also called lambda func.)

* Syntax: lambda argument : expression

* Lambda function can have any number of argument but only one expression. The expression is evaluated and returned.

* Lambda function can be used wherever function objects are required.

* Syntax: a) g = lambda x : x * x * x
g (10)
eg: 1000

b) Max = lambda a, b : a if (a > b) else b
print (Max (1,2))
Output: 2

c) x = lambda a, b, c : a + b + c
x (5, 6, 2)
Output: 13

d) x = lambda a, b : a * b
x(5, 6)
output: 30

e) def cube(y):
    return y*y*y
lambda_cube = lambda y: y*y*y
print(cube(5))
print(lambda_cube(5))

Output:
125
125

f) x = lambda a, b, c : a+b+c
a = eval(input("Enter value of a"))
b = eval(input("Enter value of b"))
c = eval(input("Enter value of c"))
Print("Sum=", x(a, b, c))

Difference between def & lambda functions:

def cube(y):
    return y*y*y
lambda_cube = lambda y: y*y*y

print(cube(5))
print(lambda_cube(5))

Output: 125
125

P.T.O...

three ways to implement lambda:

1) **with filter:** This function in Python takes a list as argument.

This offers an elegant way to filter out all the elements in sequence:

eg: l1 = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(filter(lambda x: (x%2 != 0), l1))
print(final_list)

output: [5, 7, 97, 77, 23, 73, 61]

eg: To Print people above 18 years?

ages = [13, 90, 17, 59, 21, 60, 5]
adults = list(filter(lambda age: age > 18, ages))
print(adults)

output: [90, 59, 21, 60]

2) **with Map:** if we normally multiply a no. Then it get copy of list.

∴ to implement Multiplication on list. we use map function. ie we can Multiply a single element with whole list Element using MAP function.

MAP() function in Python takes in a function and a list as an argument.

if Normal list *2

eg: list 1 = [5, 7, 8, 9, 10, 12]

2 * list 1

it would create copy of test.

Output:

[5, 7, 8, 9, 10, 12, 5, 7, 8, 9, 10, 12]

when we use map functions

```
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list (map (lambda x : x * 2, li))
print (final_list)
```

Output: [10, 14, 44, 194, 108, 124, 154, 46, 146, 122]

By we can implement +, -, *, / etc.

eg:

```
li = []
for i in range (5):
li . append (int (input ()))
print (li)
final_list = list (map (lambda x : x * 3, li))
print (final_list)
```

3) with reduce (): In this a huge list is given, then the output is given in single expression here a new reduced result is returned.

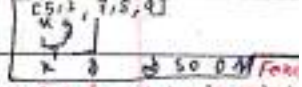
The reduce () function belongs to the functools modules.

eg:

```
from functools import reduce
li = [5, 8, 10, 20, 50, 100]
sum = reduce ((lambda x,y : x+y), li)
print (sum)
```

Output:

193



NOTE: we can't implement by single variable. i.e. x. As this output be

By we can apply x-y.

Some Program using Lambda function:

Want to enter value runtime:

```
li = [ ]
for i in range (5):
    li.append (int (input ()))
    print (li)
final_list = list (map (lambda x: x * 3, li))
Print (final_list)
```

Difference blw insert / append:

- insert: it insert value at mentioned index.
- append: it add element at end without disturbing other elements.

eg: insert (5, 20)

it would add element at 5th position.

ie 4th as starts from 0, 1, 2, 3, 4.
index

Module in function:

A file containing a set of function, we want to include in our application.

It helps to form a dictionary or file & store Multiple function or Programs and import them whenso ever we want it.

Syntax: dictionary (or file name) . function (or Program name) [attribute to be called.

P.T.O.

* Creating a Module:

To create a module just save the code you want in file, with file extension .py

Eg: Jupyter aa last checkpoint 11 min ago

Run Stop Restart Kernel

```
person1 = {
        "name": "John",
        "age": 36,
        "country": "Norway"
        }
```

Now whenever we open a new Notebook

```
Import aa
a = aa.person1["age"]
print(a)
```

Output: 36

To import,

Go To This PC → c drive → user → hp → anaconda 3 → aa

(There we can find our saved .py extension file)

PTC doc

Nested functions: A nested function is simply a function within another function and is sometimes called an "inner function"

eg:
```
def function1():                    # outer function
    print("Hello from outer function")
    def function2():                # inner function
        print("Hello from inner function")
    function2()
function1()
```

Data encapsulation: it means hiding data

There are times when you want to prevent a function or the data it has access to from being accessed from other parts of your code, so you can encapsulate it within another function.

When you nest a function like this, it's hidden from the global scope. Because of this behaviour, data encapsulation is sometimes referred to as data hiding or data privacy.

eg:
```
def outer():
    print("I'm the outer function.")
    def inner():
        print("And I'm the inner function")
    inner()
inner()
```

Note: here it is a local function → can't be called globally, as not declared globally

classes: A class defines the properties and behaviours for objects. Object oriented programming (oops) involves uses of objects to create Program.

An object represents an entity in the real world that can be distinctly identified

An object has a unique identity, state and behaviour.

An object identity helps us to uniquely identify objets during runtime. Python automatically assigns each object "a unique id"

features about class objects:

* An object's state (also known as its Properties or attributes). is represented by variables, called data fields

eg: a) A circle object.
would have a data field radius. (which is a properly that characterizes a rectangle). circle

b) A rectangle object,
would have data field width and height. (which is a property that characterizes a rectangle).

* Python uses Methods to define objects behavior
ie we make a object perform an action by invoking a function or Method on that object.

Creating an instance of the class:
A class needs to be instantiated if we want to use the class attributes or class methods.

In simple words, it refers to declaring of objects with or without arguments for that class.

Syntax: <object-name> = <class_name>(arguments)

Some programs:

WAP to print a value using class:

```
class Myclass:          # class declaration
    x=5
p1 = Myclass()          # p1 is object of Myclass class
print (p1.x)
```

Output: 5

WAP to print using function calling in class:

```
class phone:                          # class declaration
    def make_call (self):             # Function declaration
        print ("Making Phone Call")
    def play_game (self):
        print ("Playing Game")

p1= phone ()            # object declaration
p1. make_call()         # calling function using object
p1. play_game ()
```

WAP to print default values in class:

```
class Employee:
    id=10
    name = "John"
    def display (self):
        print ("ID: %d \n Name: %s" % (self.id, self.name))
# Creating Emp instance (or objects) of Employee class
        Emp = Employee()
        Emp.display ()
```

(annotations: "\n" — new line; "%" — to assign vectors to %d & %s)

Output: ID = 10
Name = John

M T W T F S S

Page No.

Date

YOUVA

#. __init__() Function: it is built-in function. All classes have a function called __init__(), which always executed when the class is being initiated

* The __init__() function is called automatically every time the class is being used to create a new object.

i.e we don't need to call it, it calls itself. (it works like a Constructor in C++)

eg: class Person:

```
def __init__(self, name, age):
    self.name = name
    self.age = age

P1 = Person("John", 36)
print(P1.name)
print(P1.age)
```

Outputs
John
36

Eg: More than one __init() function in single class:

```
Class student:
    def __init__(self):
        print("Hello World")
    def __init__(self):
        print("Python Programming")
St = student()
```

Output: Python Programming

NOTE: Here Prev. value of self is Over writed. & New Value is Printed.

Delete an Object Properties:

eg:
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def my_func(self):
        print("Hello my name is " + self.name)

P1 = Person("John", 36)
del P1.age                    # To delete this object property
print(P1.age)                 # would generate error as
print(P1.name)                age property removed.
P1.my_func()                  'Person' object has no attribute 'age'
```

Output: Error

DELETING Entire Object:

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

    def my_func(self):
        print("Hello my name is " + self.name)

# P1 = Person("John", 36)
P1.my_func()
```

Output: Hello my name is John

```
# print(id(P1))
```

Output: 161365462848

```
# del P1
print(id(P1))                 # Object deleted
```

Output: Error

M T W T F S S
Page No.
Date
YOUVA

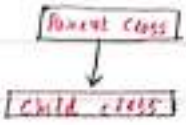
Inheritance:

When a child class inherits the properties of a parent class.

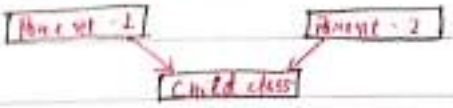
ie when a new class is derived from a child class

Types of inheritance:

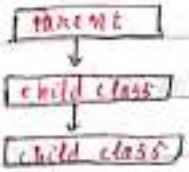
1) **Single level inheritance:** A single Parent and single child.



2) **Multiple Inheritance:** Multiple Parent & single child

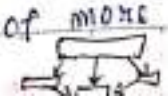


3) **Multi level Inheritance:** There are Multiple levels. ie child class is derived from another child class.



4) **Hierarchical Inheritance:** One Parent & Multiple childrens

5) **Hybrid Inheritance:** Combination of more than two inheritance



(i) Single level inheritance: (one parent, one child)

eg WAP to show execution of single level inheritance:

```
class Parent:
    def func1(self):
        print("this is function one")
class child(Parent):
    def func2(self):
        print("this is function two")
ob = child()
ob.func1()
ob.func2()
```

Output: this is function one
this is function two.

eg WAP to find sum of two no using single level inheritance:

```
class Parent:
    def sum(self):
        self.a = 10
        self.b = 20
        self.c = self.a + self.b
class child(Parent):
    def sum1(self):
        print(self.a, self.b)
        print("The sum is:", self.c)
obj = child()          # here we can make object of either Parent or child class.
obj.sum()
obj.sum1()
```

Output: 10 20
The sum is: 30

Back if we make object from Python class then we can't use features of child class.

2. Write a program to find sum of two numbers using single level inheritance using value during run time

```
class Parent:
    def sum(self):
        self.a = eval(input("Enter first No:"))
        self.b = eval(input("Enter second No:"))
        self.c = self.a + self.b
class child(Parent):
    def sum1(self):
        print("The sum of No is", self.c)
ch = child()
ch.sum()
ch.sum1()
```

output: Enter first No: 40
Enter second No: 50
The sum of No is: 90

3) Multiple inheritance:

4. Write a program to print a message using Multiple inheritance:

```
class Parent1:
    def func1(self):
        print("this is function 1")
class Parent2:
    def func2(self):
        print("this is function 2")
class child(Parent1, Parent2):
    def func3(self):
        print("This is child class")
ch = child()
ch.func1()
ch.func2()
ch.func3()
```

output:
this is function 1
this is function 2
this is function 3

Q: WAP To Print addition of two number and taking value from user

Ans

```
class Parent1():
    def function1(self):
        self.a = int(input("Enter value in Parent1"))
class Parent2():
    def function2(self):
        self.b = int(input("Enter value in Parent2"))
class child(Parent1, Parent2):
    def function3(self):
        self.c = self.a + self.b
        print("The sum in child class:", self.c)
```

Output: Enter first no. in Parent 1: 50
Enter second no. in Parent 2: 50
The sum of child class is: 100

iii) Multilevel Inheritance:

Q: WAP To show execution of Multi-level Inheritance:

Ans

```
class Parent():
    def func1(self):
        print("This is first class")
class Parent2(Parent):
    def func2(self):
        Print("This is 1st derived class")
class child(Parent2):
    def func3(self):
        print("This is 2nd derived class")
obj = child()
obj.func3()
obj.func2()
obj.func1()
```

Output: This is first class
This is 1st derived class
This is 2nd derived class

iii) Hierarchical Inheritance:

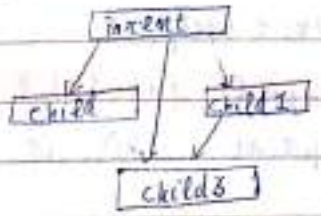
4. WAP to show execution of hierarchical Inheritance:

```
class Parent:
    def func1 (self):
        print ("this is function 1")
class child (Parent):
    def func2 (self):
        print ("this is function 2")
class child2 (Parent):
    def func3 (self):
        print ("this is function 3")

ob= child()
ob1= child2()
ob.func1()
ob.func2()
ob1.func3()
```

Output:

this is function 1
this is function 2
this is function 3

iv) Hybrid Inheritance:



P.T O

eg:
```
class Parent:
    def func1 (self)
        print ("this is function one")
class child (Parent)
    def func2 (self):
        print ("this is function two")
class child1 (Parent):
    def func3 (self)
        print ("this is function three")
class child3 (Parent, child1):
    def func4 (self):
        print ("this is function four")
ob = child3()
ob.func1()
```

Python Super() function:

it allows us to call a Method from Parent class i.e it helps To call Parent Member function in child Member function. Then we just can run child function. so it would include Parent function also.

eg:
```
class Parent:
    def func1 (self)
        print ("this is function 1")
class child (Parent):
    def func2 (self):
        super().func1()
        print ("this is function 2")

ob=child()
ob.func2()
```

output: this is function 1.
this is function 2.

Over-riding: if we use the same function name in Parent & child.

Then on calling it would call child class.

eg: WAP to show execution of overriding:

```
class Parent:
    def func1 (self):
        print ("this is parent function")
class child (Parent):
    def func1 (self):
        print ("this is child function")
ob = child()
ob.func1()
```

Output: This is child function

Access specifiers or Access Modifiers:

Access specifiers in Python have an important role to play in securing data from unauthorized access and in preventing it from being exploited.

(ie it basically helps to hide data from the user).

There are three types of access specifiers or access modifiers:

1) Public access Modifier
2) Private access modifier
3) Protected access modifier

1) Public access Modifiers:

* All the variables and methods (member function) in Python are by default public.

* Any instance variable in a class followed by the 'self' keyword.
  ie self.var_name are public accessed

* eg: syntax => self.a = 10

* eg: class student:
  def __init__(self, age, name):
  self.age = age
  self.name = name
  obj = student(21, "Python_lobby")
  print(obj.age)
  print(obj.name)

Output: 21
Python_lobby

2) Private Access Modifier:

* Private members of a class (ie either variable or methods) are those members which are only accessible inside the class.

* Private members cannot access outside of class
* i.e we can't call it through child class
* so declare object of private class & have to call within function.

Syntax → self.age = 32

```
class student:
    def __init__(self):
        self.name = 'Adam Sol'
        self.age = 25

class subject(student):
    pass

obj = student()
obj1 = subject()

# calling using object reference of student class
print(obj.name)          # No error
print(obj1.name)         # No error.

# calling using object reference of subject class
print(obj.__age)         # Error
print(obj1.__age)        # Error
```

Output: Adam Sol
Adam Sol

Error
Error

Protected Access Modifier

Class can be accessed by member function & class member of same class or by inherited class.

Syntax → self._age = 34

★ eg: if class a
↓
b
↓
c

here b can use as Protected data member But c can.

★ eg: class student:
def __init__(self):
self._name = "Python-lobby.com"

def _funName(self):
return "Method there"

class subject(student):
Pass

obj = student()
obj1 = subject()

calling by obj. ref of student class
print(obj._name) # Python-lobby.com
print(obj._funName()) # Method here
calling by obj ref of subject class
print(obj1._name) # Python-Lobby.com
print(obj1._funName()) # Method here

Output: Python Lobby.com
Method there
Python Lobby.com
Method there

Lab Programs :

1. Write a program to create a student class with information such as roll no, name, class. The information must be entered during run time

```
class student:
    def __init__(self):
        for i in range(1):
            self.roll no = int(input("Enter the no"))
            self.name = input("Enter Name")
            self.class = input("Enter class")

class student1(student):
    def details(self):
        for i in range(1):
            print("Details of student",i)
            print("Name of student is", self.name)
            print("Roll No. of student is", self.rollno)
            print("Class of student is", self.class)

obj = student1()
obj.details()
```

2. Write a program to implement Library management system using classes and objects

```
```

```
class library:
    def book_details():
        for i in range(5):
            self.bookname = input("enter book
            self.bookprice = eval(input("Enter
            self.bookpage = int(input("Enter
            self.date_of_issue = input("Enter
            self.date_of_return = input("Enter
class library_info(library):
    def details_print():
        for i in range(5):
            print("Book name =", self.bookname)
            print("Book price =", self.bookprice)
            print("Book Pages =", self.bookPages)
            print("Date_of_issue=", self.date_of_issue)
            print("Date_of_return =", self.date_of_re
obj = library_info()
obj.Details()
obj.Detailsprice()
```

Difference b/w function over-riding and function overloading

☆ function overriding: it refers to the situation where the Parent class & child class has a function with same name. And on calling preference is given to func in child class. it generates run time error

★ function overloading: Overloading is the process where same function can be used multiple times, by passing different no. of parameters/arguments.

NOTE: we can't take default arguments in overloading.

Example of function overriding:

```
class A:
    def func1 (self):
        print ("value of func1")
    def func2 (self):
        print ("value of func2")
class B (A):
    def func2 (self):                    # here this function would over ride Parent classes func2.
        print ("Modified value of func2")
obj = B()
obj.func2()
```

output: Modified value of func2.

Example of function overloading:

```
def area (l, b):
    c = l * b
    return c
def area (size):                         # Now as both function have same name, in function overloading Priority is given to second function.
    c = size * size
    return c

# area (4)      (here it would return output 16)
# area(5,6)     (here it would return an error as one Parameter & two values).
```

P.T.O :...

Poly morphism:

it is taken from Greek words Poly (many) and morphism (forms)

It means that same function can be used for different types.

eg: Print (5+6)
Print ("5"+"6")

with difference in data type the output changes its form.

Output: 11
56

eg: class French:
    def say_hello (self):
        print ("Bonjour")
class Chinese:
    def say_hello (self):
        print ("Ni-hao")
def intro (lang):
    lang.say_hello() # function to call function in class module.

Sarthak = French() # Chg. say_hello() function
John = Chinese() # object of class French
intro (Sarthak) # object of class Chinese
intro (John) # To pass value in intro function

Output: Bonjour
Nihao

UNIT-2

Data Analytics NVD Introduction to Matplotlib...

Sw: what do you mean by Python libraries? Explain its Diff. Types.

Ans: Python library is a collection of codes and methods that allows to perform many actions without writing codes.

Different Types of libraries in Python:

- Numpy (Numerical Computing)
- Matplotlib (Data visualisation)
- Pandas (Data Manipulation)

Numpy:

* Numpy is a Python Package. it stands for 'Numerical Python'

* it is a library consisting of Multi dimensional array objects and a collection of routines (or functions) for processing of array.

* Numpy is a Python library, written partially in Python, but most of the parts that require fast computation are written in C or C++.

* Numpy is basically used for working in domain of linear algebra, fourier transform and matrices.

* Numpy array helps us to store heterogeneous data elements ie of various data types.
eg: [1, 2, 3, 'a', 3.9]

* To install in any platform (Windows, Ubuntu, Mac)

Syntax: pip install numpy

How to check version of numpy:

```
import numpy as np
print (np.__version__)
```

importing numpy library and giving it an alias of numpy

Output: 1.21.5

How to declare numpy as single dimensional array:

```
import numpy as np
a= np.array ([1,2,3,4,5])
print (a)
print (type (a))
```

here 'a' is variable to store value

output:

[1, 2, 3, 4, 5]

<class 'numpy.ndarray'>

WAP To Print Multi-dimensional Array:

```
import numpy as np
a= np.array ([[1,2,3], [4,5,6]])
print (a)
```

output: [[1, 2, 3]
[4, 5, 6]]

PTO

M T W T F S S
Page No.
Date
YOUVA

11 Automatic type conversion of data in array:

a)
```
import numpy as np
a = np.array ([2, 3, 4, 5.5])
print (a)
```
here all values would be converted into float.

output: 2.0
3.0
4.0
5.5

b)
```
import numpy as np
a = np.array ([2, 3, 4, 'a'])
Print (a)
```
here all values would be converted into string.

Output: '2'
'3'
'4'
'a'

```
import numpy as np
a = np.array ([5, 6.7, 8.9, 2, 'a'])
Print(a)
```
here also data would be printed in string format.

Output: '5'
'6.7'
'8.9'
'2'
'a'

P.T.O

How To insert space b/w Array:

```
import numpy as np
a = np.array([5, 6, 7, ' ', 12, 13])
Print(a)
```

Output: ['5', '6', '7', ' ', '12', '13']

But this converts whole data in string.

∴ To avoid this we use 'NaN' function:
it stands for 'not an number'.

numpy defines NaN as a constant value.

As we all know in numeric data type we can use To only represent real Numbers.

Therefore whenever we want to compute any value (such as square root of a -ve no. which would result in imaginary values).

To represent such values in data frames we use 'nan function'.
By default data type Produced by nan is float.
Conclusion:
This function displays 'nan' as out Put at the place of usage in Program.

WAP to show execution of NaN:

```
import numpy as np

a = np.array (["Ritam", "yashi", np.nan])
print(a)
print (type (a))          # Print datatype of a variable
print ("\n")

b= np array ([45, 7, 67, np.nan])
print(b)
print (type (np.nan))     # To print data type of nan
```

output:
```
["Ritam", "yashi" "nan"]
<type 'numpy.ndarray'>

[45.0  7.0  67.0  nan]
<type 'float'>
```

Ques what is the difference between array and list?

Ans

List	array
* it is used to collect items that usually consists of elements of multiple data types	Array collects several items of same data type. & if not then it converts them into same data type.
* list cannot manage airthematic operations	Array can manage airthematic operations.

Initializing Numpy Array with Zeros:

for this we use zeros function it sets all the elements in the matrix equal to zero.

eg:
```
import numpy as np
n1 = np.zeros((2,2))
print(n1)
```
2 x 2 → represents columns; ↓ represents row

output:
```
[[0.0 0.0]
 [0.0 0.0]]
```
As By default nature of numpy is float

Initializing Numpy Array with same Number:

```
import numpy as np
n1 = np.full((5,5), 10)
print(n1)
```

output:
```
[[10 10 10 10 10]
 [10 10 10 10 10]
 [10 10 10 10 10]
 [10 10 10 10 10]
 [10 10 10 10 10]]
```

Ques: what is difference between range and arange function.

Ans: D.T.

range ()	arange ()
* it comes as a default function of Python.	it comes under Numpy library of Python.
* it is independent of library or module	it depends on Numpy library.
* it generates simple series of number depending on range	it generates a series of numbers but in form of nd array (or numpy array).
* As it generate simple values, Hence faster	Since it works with array. & Programer need to import libraries Hence slower
* it is used to traverse lists etc	it is used to traverse array.

NOTE: Both range and arange have same Parameters.

i.e (start, stop, step)

- start: where to start from
- stop: till where to exceed
- step: How much to increase (By default step = 1)

WAP To Print array using arange :

```
import numpy as np
n1 = np.arange(10, 20)
print(n1)
```

Output : [10 11 12 13 14 15 16 17 18 19]

WAP using numpy arange function using step 2:

```
import numpy as np
n1 = np.arange(10, 20, 2)
print(n1)
```

Output: [10, 12, 14, 16, 18]

NOTE: Here if we want to insert comma in blw output. Then we can't

As it would either in front or completely at last and not in blw outputs.

As in arange function it would completely traverse the array.

Numpy Array Indexing:

★ WAP to get first element in array:

```
import numpy as np
arr = np.array([1,2,3,4])
print(arr[0])
```

Output: 1

★ WAP to get Third & Fourth element from Array and add them...

```
import numpy as np
arr = np.array([1,2,3,4])
print(arr[2] + arr[3])
```

Output: 7

M T W T F S S

Page No. | YOUVA

Date

* WAP To get a element from 2-Dimensional array

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st row:', arr[0,1])
```

Output: 2nd element on 1st row: 2

NOTE: In

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr)
```

Output: [[1 2 3 4 5] → row 0

[6 7 8 9 10]] → row 1

6 → 0 Col, 7 → 1 Col, 8 → 2nd col, 9 → 3rd col, 10 → 4th col

here in Python index always starts with Zero.

* Negative Indexing:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr[1,-1])
```

Output: 10

Numpy Array Slicing:

Slicing is a method to extract some elements from a array using index.

Syntax: [starting index : ending index]

NOTE: The value is Taken one less than last mentioned index.

★ W&P To slice elements from index 1 to 5.

```
import numpy as np
arr = np.array([1,2,3,4,5,6,7])
print(arr[1:5])
```

Output: [2 3 4 5]

★ W&P To return elements using slice with a step.

```
import numpy as np
arr = np.array([1,2,3,4,5,6,7])
print(arr[1:5:2])
```

Output: [2,4]

★ W&P To slice elements from 2-D array:

```
import numpy as np
arr = np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(arr[0,1:4])   # This means first row & from index 1 to 4
```

Output: [7 8 9]

★ Extension:

```
import numpy as np
arr = np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(arr[0:2,1:4])
```

Output: [[2 3 4]
[7 8 9]]

Data types in Numpy:

* WAP to get data type of an array object:

```
import numpy as np
arr = np.array([1,2,3,4])
print(arr.dtype)
```

Output: int32

* WAP to create an array with data type 4 Byte INTEGER:

```
import numpy as np
arr = np.array([1,2,3,4], dtype='i4')
print(arr)
print(arr.dtype)
```

Output: [1 2 3 4]
int32

* WAP to change data type from float to integer in array:

```
import numpy as np
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype(int)
print(newarr)
print(newarr.dtype)
```

Output: [1 2 3]
int32

Note: if we define data type as Bool.
Then if no. is 0.
Then output false.
⇒ if any +ve no. then output is True

Shape method: This method is basically used to check no of rows and columns in array.

* WAP to return shape of 1-Dimensional array:

```
import numpy
arr = numpy.array ([1,2,3,4,5])
print (arr.shape)
```

As 1-Dimensional, in output either only row or column i.e rows = 0, col = 5

Output: (5,)

* WAP to print the shape of 2-Dimensional array:

```
import numpy as np
m1 = np.array ([(1,2,3), (4,5,6)])
print (m1.shape)
```

output: (2, 3)

* WAP to create array using shape function and Pre-defined dimensions

```
m1.shape = (3,2)
Print (m1)
```

In output the values are generated from 1,2,3.... & so on it based on previous program values of m1. It reshapes the rows & column

output: [[1 2]
[3 4]
[5 6]]

ndim method: ndim function is used to return the Dimension of an array.

* WAP to check how many Dimension the array have:

```
import numpy as np

a = np.array(42)
b = np.array([1,2,3,4,5])
c = np.array([[1,2,3],[4,5,6]])
d = np.array([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]])

Print(a.ndim)
print(b.ndim)
Print(c.ndim)
Print(d.ndim)
```

Output: 0
1
2
3

NOTE: To check no. of dimension by just seeing, just see or check the no. of square brackets.

Re-shape function: This function helps to convert dimension of an array.

Syntax: (, ,)
- no. of array
- no. of rows
- no. of cols.

* Convert 1-D array with 12 elements into 2-D array.

```
import numpy as np
arr = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
newarr = arr.reshape(4,3)
print(newarr)
```

Output:
```
[[1 2 3]
 [4 5 6]
 [7 8 9]
 [10 11 12]]
```

* Convert 1-D array into 3-D array

```
import numpy as np
arr = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
newarr = arr.reshape(2,3,2)      # it means outermost dimension will have Two arrays, that would contain 3 arrays of 2 elements in each
print(newarr)
```

Output:
```
[[[1 2]
  [3 4]
  [5 6]]
 [[7 8]
  [9 10]
  [11 12]]]
```

P.T.O.

Iterating Arrays:

* WAP to iterate on the elements of following 2-D array:

```
import numpy as np
arr = np.array([[1,2,3], [4,5,6]])
for x in arr:
    print(x)
```

Output: [1 2 3]
[4 5 6]

* WAP to iterate on each scalar element of 2-D array:

```
import numpy as np
arr = np.array([[1,2,3], [4,5,6]])
for x in arr:
    for y in x:
        print(y)
```

Output: 1
2
3
4
5
6

F T ...

Transpose of the array:

This function helps to convert rows into columns and columns into row

eg: import numpy as np
arr = np.arange(12).reshape(3,2,2)
Print('Array before Transpose /n', arr)
Print('Array After Transpose /n', arr.transpose
Array before Transpose:

Output: [[[0 1 2 3]
[4 5 6 7]
[8 9 10 11]]

Array After Transpose:
[[[0 4 8]
[1 5 9]
[2 6 10]
[3 7 11]]

To swap elements of Array:

* Swapping of two no. in Python:

a = 10
b = 20
print(" Before Swaping", a, b)
a, b = b, a
print(" After Swaping", a, b)

P.T.O

Page No.: YOUVA
Date:

* Swapping the columns:

$$arr = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \end{bmatrix}$$

if we want to swap column 0 & 1 without changing row no.

```
arr [ :, [0, 1] ] = arr [ : , [1, 0] ]
print (arr)
```

output: [[1 0 2 3]
[5 4 6 7]
[9 8 10 11]]

* Swapping two rows:

```
import numpy
arr = numpy.arange(12).reshape(3,4)
print (arr)
arr [ [1,2], : ] = arr [ [2,1], : ]
print (arr)
```

Out Put: [[0 1 2 3]
[4 5 6 7]
[8 9 10 11]]

[[0 1 2 3]
[8 9 10 11]
[4 5 6 7]]

How to read dataset using numpy

* WAP to import a data set with numbers and

if we want to read our Notepad file

Step 1: Open notepad
Step 2: Create a matrix
Step 3: Store data → 1 2
3 4
5 6
7 8
step 4: Save file on desktop as text

Now to read this file, we right click on Text file click on property and then copy path of file

* Then in Jupyter Note Book:

file_data = ("C:\user\91988\one drive\Desktop\Text.txt")
↓
file name with extension

This print would show error

print (file_data)

assigning datatype to file data.

To avoid error change all backward slash to forward and we'll use loadtxt. → this is used to read file from desktop.

Main Program in Jupyter:

file_data = numpy.loadtxt ("C:/User/91988/one drive/Desktop/Text.txt", dtype = int)

print (file_data)

Output: [[1,2]
[3,4]
[5,6]
[7,8]]

Or

We can directly save file in anaconda folder & from there we can simply call file, by just using its name.

ie file_data = numpy.loadtxt ("text.txt", dtype = int)
print (file_data)

Output: [[1 2]
[3 4]
[5 6]
[7 8]]

Joining NumPy Arrays:

There are three different ways that in which we can combine (or concatenate arrays):

1) v-stack → (it joins two arrays vertically)
2) h-stack → (it joins two arrays horizontally)
3) column-stack → (it joins two arrays column wise)

WAP To show execution of v-stack, h-stack, column-stack:

```
import numpy as np
n1 = np.array ([(1,2,3)])
n2 = np.array ([(4,5,6)])
print (np.vstack((n1, n2)))
print (np.hstack((n1, n2)))
print (np.column_stack((n1, n2)))
```

Outputs:

[[1, 2 3]
[4 5 6]]

[[1 2 3 4 5 6]]

[[1 4]
[2 5]
[3 6]]

ADDITION of NumPy arrays:

for addition in Python, we use Sum() function which is an in built function of numpy ∴ we don't need to declare

* Addition of NumPy array:

```
import numpy as np
n1 = np.array([1,2,3])
n2 = np.array([4,5,6])

print(np.sum([n1, n2]))
```

This would add all elements of array into single one

Output: 21

* To get addition's output in array format

for this we use axis.

```
import numpy as np
n1 = np.array([1,2,3])
n2 = np.array([4,5,6])
print(np.sum([n1, n2], axis=0))
```

Output: [5 7 9]

Extension:

```
print(np.sum([n1, n2], axis=1))
```

Output: [6 15]

↓ Here it would add column wise.

To Add Two Arrays..

```
import numpy
arr1 = numpy.array([[1,2],[3,4]])
arr2 = numpy.array([[5,6],[7,8]])

arr3 = numpy.sum([arr1, arr2], axis=0)
Print(arr3)
```

Output:
```
[[ 6  8]
 [10 12]]
```

Subtraction, Multiplication & Division of Two Arrays:

```
import numpy as np
n1 = np.array([1,2,3])
n2 = np.array([4,5,6])
print(np.sum([n1, n2]))
print(n1 * 2)
n3 = n1.dot(n2)
print(n3)
print(n1/2)
print(n1+10)
print(n2-n1)
```

To multiply whole array elements by 2

To Perform Matrix Multiplication

[dot function is used for Matrix Multiplication only condition is, No of columns of 1st Matrix Should be equal to no of rows of 2nd Matrix.]

Output:
```
21
[2 4 6]
[4 10 18]
[5 1.0 1.5]
[11 12 13]
[3 3 3]
```

NOTE: For Multiplication of Three matrix, firstly Perform Multiplication on array 1 & 2 => Then Multiply with Third Matrix.

Numpy Math functions (i.e Mean, Meadian, std. De

★ WAP To find mean of array:

```
import numpy as np
n1 = np.array ([10, 20, 30, 40, 50, 60])
print (np.mean (n1))
```

output: 35.0

★ WAP To find median of array:

```
import numpy as np
n1 = np.array ([10, 20, 30, 40, 50, 60])
print (np.median (n1))
```

output: 35.0

★ WAP To find standard Deviation:

```
import numpy as np
n1 = np.array ([10, 20, 30, 40, 50, 60])
print (np.std (n1))
```

Output: 17.078251

Splitting Numpy Arrays:

The split function allows us to split a single arrays into Multiple arrays.

Syntax: n1 = np.array.split (,)

WAP To split array in Three Parts:

```
import numpy as np
arr = np.array([1,2,3,4,5,6])
newarr = np.array_split(arr, 3)
Print(new arr)
```

Output: [array([1,2]), array([3,4]), array([5,6])]

if we want To Print any one of these newly formed array

```
Print(newarr[1])
```

Here for new arr[0] -> [1,2]
new arr[1] -> [3,4]
new arr[2] -> [5,6]

output: [3 4]

WAP To split 2-Dimensional arrays into Three Two Dimensional arrays:

```
import numpy as np
arr = np.array([[1,2],[3,4],[5,6],[7,8],[9,10],
                [11,12]])
newarr = np.array_split(arr, 3)
Print(new arr)
```

Out Put: [array([[1,2],
[3,4]]),
array([[5,6],
[7,8]]),
array([[9,10],
[11,12]])]

Searching and Sorting Numpy Arrays:

1) Searching: it is used to find index of a particular item

★ WAP to find indexes where the value is 4:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)
```

output: (array([3, 5, 6], dtype = int64)

[NOTE: a) for searching we use "where clause".]

b) In two dimensional array it would return row and column.

★ Find the indexes where the values are even:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 0)
print(x)
```

output: (array([1, 3, 5, 7], dtype = int64)

2) Sorting: this function helps to sort the array elements.

Syntax: Print(numpy.sort( ))

array name

NOTE: i) In list when we sorted we sorted we didn't needed or used any library.
i.e List1 = [5, 9, 8, 12, 2]
print (list.sort())

ii) whereas in Array for sorting we have to mention numpy to call the sort function residing in numpy library.

* WAP TO sort the array:

```
import numpy as np
arr = np.array([3,2,0,1])
print (np.sort(arr))
```

Output: [0 1 2 3]

* WAP To Sort array containing strings:

```
import numpy as np
arr = np.array(['banana', 'cherry', 'apple'])
print (np.sort(arr))
```

Output: ['apple' 'banana' 'cherry']

* WAP To Sort 2-D array:

```
import numpy as np
arr = np.array([[3,2,4], [5,0,1]])
print (np.sort(arr))
```

Output: [[2 3 4]
[0 1 5]]

Numpy filter array:

* Way To create a filter array That will return Values higher Than 42:

```
import numpy as np
arr = np.array ([41, 42, 43, 44])
filter_arr = []                    # create an empty list

for element in arr:                # go through each element in arr
  if element > 42:                 # if element is higher than 42, set it
    filter_arr.append(True)        values to True, otherwise false.
  else:
    filter_arr.append(False)
newarr = arr [filter_arr]
print (filter_arr)
Print (new arr)
```

Output: [False, False, True, True]
[43, 44]

What is difference b/w List and arrays:
ie why do we Prefer numpy array over list...

1) Less memory
2) Fast
3) Convenient

P t O for explaination...

1) Less Memory: The numpy array occupies very less space in comparison to list.

LIST...

```
import sys
S = range (1000)
print (sys getsize of (S) * len(S))
                  ↓          ↓
                 =28        1000
```

Out Put: 28000

ARRAY...

```
import numpy as np
import sys
D = np. arange (1000)
Print ( D. size * D. itemsize)
         ↓          ↓
         4          4
                   = 1000
```

out Put: 4000

Conclusion: if we want to store 1000 variables. Then List Takes 28000 memory and numpy occupies 4000 only.

NOTES Concept Acc to Viva point of view

In C, C++ we use "size of" command is used to get the size of a particular variable or function. whereas in Python we "import sys" library and we use "sys.getsize of ()" command.

```
import Sys
a = 10
b = 10.5
c = 'Hello'
print (sys. getsize of (a))
print (sys getsize of (b))
print (sys. getsize of (c))
```

Output: 28
24
54

2) Fast and convenient:

List

```
# Time Taken by list
SIZE = 10 00000
L1 = range(SIZE)
L2 = range(SIZE)
L3 = []
import time
import sys

start = time.time() # used to check time complexity
for i in range(len(L1)):
    L3.append(L1[i] + L2[i])
Print((time.time() - start) * 1000)
```

Output: 983.49142

Array

```
import time
import sys
size = 100 000
A1 = np.arange(size)
A2 = np.arange(size)
start = time.time()
result = A1 + A2
print((time.time()-time())-
      start) * 1000)
```

Output: 56.81903

x ——x——x

for multiply by 1000 to convert output in micro seconds from mano seconds.

Conclusion: a) List took 983 mili seconds whereas the numpy array took almost 56 ms. Hence numpy array is faster than list.

b) By a 'for loop' is used in list for getting sum. whereas for numpy array we simply add two arrays.

that's why working with numpy is much more convenient than list.

Diff b/w Getsizeof() and itemsize commands:

Getsizeof() command is used to return size of entire list
itemsize() command is used to get size of single element in an array.

WAP To show Difference:

* Getsizeof() is used to get size of element in list.

```
a = 10
print (sys.getsizeof(a))
```

* item size() used to get size of single array element

```
a = numpy.array ([1])
print (a.itemsize())
```

Out Put: 4

Random numbers:

This function is used to generate random numbers.

* WAP to generate random integer from 0 to 100:

```
from numpy import random
x = random.randint (100)
print (x)
```

Out Put: 70

* WAP to create one Dimensional array containing 5 random number to print values from 0 to 10.

```
from numpy import random
a = random.randint(10, size=(5))
print a
```

or

```
a = random.rand()          # here it will give default values
print(a)
```

Note: By Default range of random no. is in float & in blw 0 to 1.
(i.e if nothing is mentioned Then random no. would be in blw 0 to 1.

why by default data type is float
① As float can store more data & its range is quite large in comparison to other data type
② it helps storing more acurate value.

* Generate a 2-D array with 3 rows, each row containing 5 random numbers:

```
from numpy import random
u = random.rand(3,5)
print(u)
```

Output: [[0.35495 0.728313 0.21282 0.19465 0.149 16]
[0.794003 0.28913 0.15827 0.55834 0.0]
[0.50994 0.73185 0.92685 0.32926 0.0]

M T W T F S S

Page No.

Date:

YOUVA

Generate a 2-D array with 3 rows, each row containing 5 random integer from 0 to 100:

```
from numpy import random
x = random.randint(100, size=(3,5))
print(x)
```

Output: [[27 80 84 38 12]
[85 0 60 23 3]
[6 85 62 24 38]]

if 2-D array & we want to print 10 elements

```
from numpy import random
a = random.randint(10, size=(5,2))
print(a)
```

range from where to where

no. of rows

no. of columns

other possible Dimensions with 10 elements

(2,5)
(1,10)
(10,1)

WAP to return one value in an array:

```
from numpy import random
x = random.choice([3,5,7,9])
print(x)
```

Output: 7

* WAP To generate random float from 0 to 1:

```
from numpy import random
x = random.rand()
print(x)
```

output: 0.46572038

* WAP To Generate a 1-D array containing 5 floats:

```
from numpy import random
x = random.rand(5)
print(x)
```

output: [0.30137 0.38477 0.20316 0.19807 0.93368]

MatPlotLib...

* it is one of the most popular Python Packages used for data visualisation i.e drawing graphs, shapes etc.

* it is a cross-platform library used for making 2-D plots from data in arrays.

* Matplotlib is mostly written in Python, a few segments are written in C, objective C and Javascript.

* it has procedural interface named Pylab, which is designed to resemble Matlab. Matplotlib along with numpy can be considered equivalent to MATLAB.

Page No. YOUVA
Date

LINE:

for drawing a line we basically need two points. ie (x_1, y_1) & (x_2, y_2).

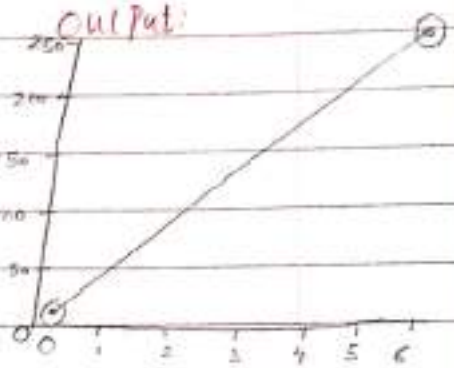
Line is a interaction of two points.

WAP To draw a line in a diagram from pt (0,0) to position (6, 250):

Python Plot

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])      # x1, x2
ypoints = np.array([0, 250])    # y1, y2
plt.plot(xpoints, ypoints)
plt.show()
```

Output:



here we have used array instead of list so as to represent 2 dimension of points. ie x_1, y_1.

Note: Show & Plot are library functions of matplotlib library.

WAP To Plot a line without giving x-Points or x-axis dimension:

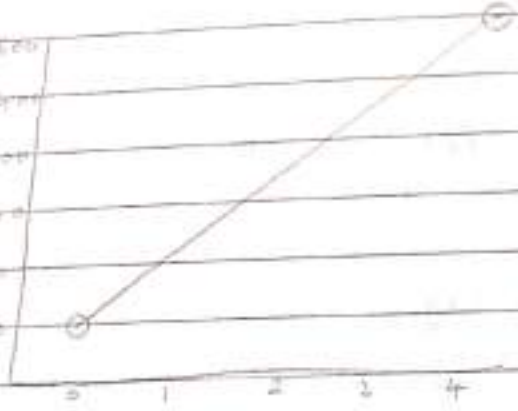
```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([100, 200, 300, 400, 500])   # 0 1 2 3 4
plt.plot(ypoints)
plt.show()
```

here by default the values of x-axis are taken as index values. Depending on length of y-Points

Extension: What if we take x-Points.

```
import Matplotlib.pyplot as plt
import numpy as np

xpoints = numpy array ([100, 200, 300, 400, 500])
plt plot (xpoints)
plt show()
```

Out Put:

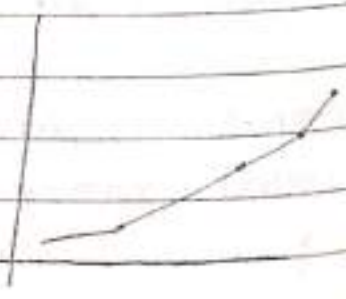


Conclusion: As we can see There is no change in Comparison To Previous Program.
Reason Being The xpoints and ypoints are Just Variable names

✗ WAP To Plot/Pass Multiple Points in array:

```
import Matplotlib.pyplot as plt
import numpy as np

xpoints = np.array ([10, 20, 30, 40])
ypoints = np.array ([20, 30, 60, 100])
plt.plot (xpoints, ypoints)
plt.show()
```

Output:



NOTE: But if in x-axis 4 points & in y-axis 3 points. Then it would show an error.

Displaying Points / or Plotting without line:

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1,8])
ypoints = np.array([3,10])

plt.plot(xpoints, ypoints, 'o')
plt.show()
```

'o' → Marker

Out Put:

To change color of dot,

```
plt.plot(xpoints, ypoints, 'o', color = 'k')
```

k → BLACK

Various Matplotlib Markers, Line Refrences, Colors Refrences:

Line Refrences → The Line Joining Dots

Colors → color of dots.

Line Refrences:

Line syntax	Description
'-'	Solid line
':'	Dotted line
'--'	Dashed line
'-.'	Dashed / Dotted line

P.T.O for Color Refrences and Markers...

2) Markers:

Markers	Description
'o'	Circle
'*'	star
'.'	Point
','	Pixel
'x'	X
'X'	X (filled)
'+'	Plus
'P'	Plus (filled)
's'	square
'D'	Diamond
'd'	Diamond (thin)
'p'	Penta gon
'H'	Hena gon
'h'	Hena gon
'v'	Triangle down
'^'	Triangle UP
'<'	Triangle left

3) Color References:

Color syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	yellow
'k'	Black
'w'	White

Line Properties :

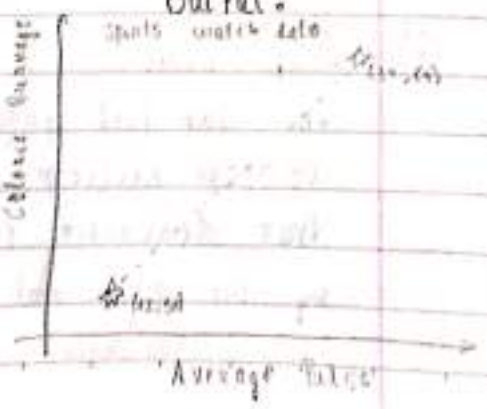
There are the following line properties

- linestyle
- color
- width / line width
- marker
- ms (marker size)
- mfc (marker color)
- title
- xlabel
- ylabel
- grid

Prg To show execution of line style, marker, marker size marker color, marker edge color

```
import MatplotLib.Pyplot As plt
xpoints = numpy.array([5,34])
ypoints = numpy.array([56,61])
plt.figure(figsize=(20,10))          # To increase size of display image
plt.plot(xpoints, ypoints, linestyle=':', color='r', marker='*',
linewidth='5', ms=25, mfc='r', mec='k')
plt.title("Sports watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.show()
```

Output:



NOTE: marker = To increase size and
markersize = size of marker face
mfc = marker color
mec = marker edge color
title = To give title to graph
xlabel = To give label to x axis

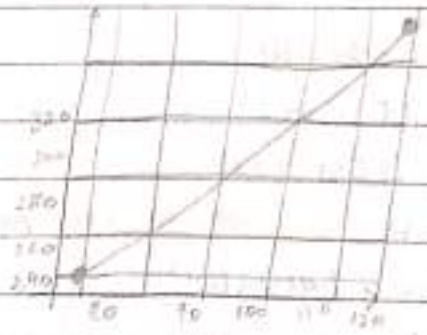
NOTE: In plt.plot(,)
- 1st one is considered as x-axis
- 2nd one is considered as y-axis

if plt.plot([1, 2, 3, 4])
- 0 1 2 3
- data represents x-axis
- data represents y-axis

* WAP to Add Grid To a plot:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x,y)
plt.grid()
plt.show()
```

output:



Sub-Plot:

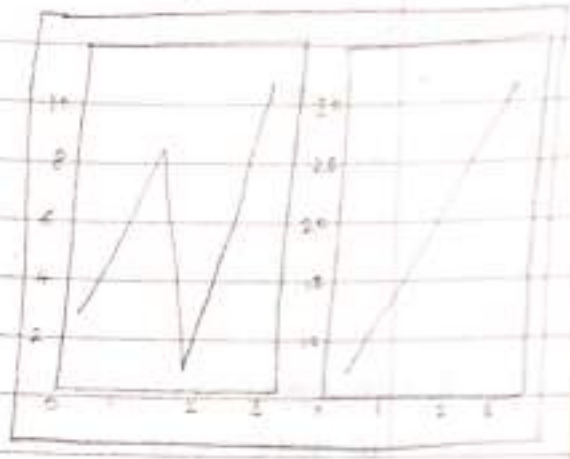
The sub-plot function divides output screen into various section. This function takes three arguments that describes the layout of the figure.

by the first and second argument and third argument it is represented
- first → row index
- second → column index
- third → figure index of current plot.

WAP To show execution of sub plot:

```
import matplotlib.pyplot as plt
import numpy as np

# plot 1
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x, y)
# plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x, y)
plt.show()
```

Outputs



NOTE: here (1,2,1) & (1,2,2) indicates
row=1, column=2 (As for two figure)
& this represents figure no.

Ex: If we want to print in two rows & single col.

```
plt.subplot(2,1,1)
plt.subplot(2,1,2)
```

or

```
plt.subplot(2,1,2)
plt.subplot(2,1,1)
```

output:

What would be subplots to represent fig in following

Answers:

1) ① ②
③ ④

Ans plt.subplot(2,2,1)
plt.subplot(2,2,2)
plt.subplot(2,2,3)
plt.subplot(2,2,4)

(here (2,2,-) represents the matrix structure)

2) ① ② ③ ④

Ans plt.subplot(1,4,1)
plt.subplot(1,4,2)
plt.subplot(1,4,3)
plt.subplot(1,4,4)

3) ①
②
③
④

Ans plt.subplot(4,1,1)
plt.subplot(4,1,2)
plt.subplot(4,1,3)
plt.subplot(4,1,4)

The Various Types of Plots:

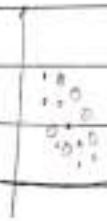
1) Bar Graph: A bar graph uses bars to compare data among different categories. it is well suited when we want to measure the changes over a period of time.

it can be represented horizontally or vertically.

2) Histograms: Histograms are used to show a distribution whereas a bar chart is used to compare different entities.

Histograms are useful where arrays in very large size.

3) scatter plot: This method is used to compare variables, i.e how much one variable is affected by another variable to build a relation out of it.



The data is displayed as a collection of points, each having value of one variable.

4) Area plot: Area plots are pretty much similar to the line plot. They are also known as "stack plots".

These plots can be used to track changes over time for two or more related groups. For a particular category.

5) Pie chart: A pie chart refers to circular graph which is broken down into segments. i.e slices of pie. it is basically used to show the percentage or proportional data, where each slice represents a category.

Scatter Plot:

* WAP To show execution of Scatter Plot & various function that can be applied on it:

```
x = np.array ([10, 20, 60, 30, 90])
y = np.array ([60, 80, 90, 100, 110])
plt.scatter (x, y)
plt.show()
```

Extensions:

a) To change color of dots; title ; xlabel & ylabel:

```
x = np.array ([10, 20, 60, 30, 90])
y = np.array ([60, 80, 90, 100, 110])
plt.scatter (x, y, color = "green")

x1 = np.array ([2, 10, 30, 60, 80])
y1 = np.array ([10, 20, 30, 40, 50])
plt.scatter (x1, y1)

plt.title ("Scatter plot")
plt.xlabel ("age")
plt.ylabel ("value")
```

This scatter plot is used to plot two sets of data for comparision

b) Legend, Marker, Marker style, Marker size, Change color of plot, label in scatter plot

F T E ...

import numpy as np
import Matplotlib.pyplot as plt

x = np array ([10, 20, 30, 40, 50])

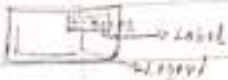
y = np array ([60, 80, 70, 100, 110])

Colors = ["green", "BLACK", "BLUE", "RED", "PINK"] # creating array of colors to use in scatter plots color fuction to get various colored dot

plt Scatter (x, y, color = colors, label = "marks", marker = "^", s = 10)

plt legend() # legend create a box over figure describing label name included to color

plt show()



Q) WAP To draw scatter plot for

x	y1	y2
A	100	150
B	200	250
C	300	350
D	400	450

and Label

a) x-axis as x points
b) y-axis as y-points
c) Also draw lines blw These scatter

import numpy as np
import Matplotlib.pyplot as plt

x = Np array (['A', 'B', 'C', 'D'])

y1 = np array ([100, 200, 300, 400])

y2 = np array ([150, 250, 350, 450])

plt scatter (x, y1)
plt scatter (x, y2)
plt.plot (x, y1) # To plot line blw scatter dots
plt plot (x, y2) # To plot line blw scatter dots

plt xlabel (x-points)
plt ylabel (y-points)

plt.show()

Bar graph:

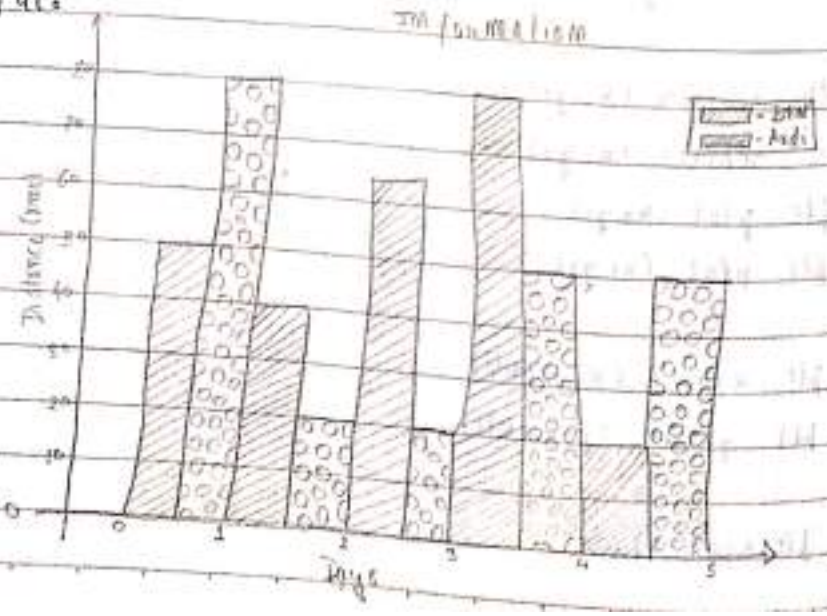
for drawing bar graph we use plt.bar(x,y). (instead of plt.scatter(x,y).

* Just like scatter plot all other properties are same & similarly applied.

* WAP To show execution of bar graph:

```
from matplotlib import pyplot as plt

plt.bar([0.25, 1.25, 2.25, 3.25, 4.25], [50, 40, 70, 80, 20],
        label="Bmw", width=0.5)

plt.bar([0.75, 1.75, 2.75, 3.75, 4.75], [80, 20, 20, 50, 60],
        label="Audi", color='r', width=0.5)
plt.legend()
plt.xlabel('Days')
plt.ylabel('Distance (kms)')
plt.title('Information')
plt.show()
```

Output:



To Draw horizontal Bar graph:

plt.barh(x,y)

or

* As vertical graphs: plt.bar(x,y, color="Red", width="0.5")
* As horizontal bars: plt.barh(x,y, color="Green", height="0.5")

Note: By Default width in graph 0.5.

linespace

linespace is a function of numpy library.
we just pass the three arguments: linespace(1, 20, 5)

i.e x = np.linespace(10, 20, 5)
Start End No. of elements

it is just as range & arange function, But here we don't have to give values, we just have to give starting & end value, and no. of elements we want, rest it takes random values for those elements in b/w initial and end.

eg: x = np.linspace(10,20,5)
plt.plot(x)
plt.show()

Output: [11, 13, 15, 18, 12]

PTO...

Histogram:

Histogram is used with intervals and datasets. i.e. for large datasets.

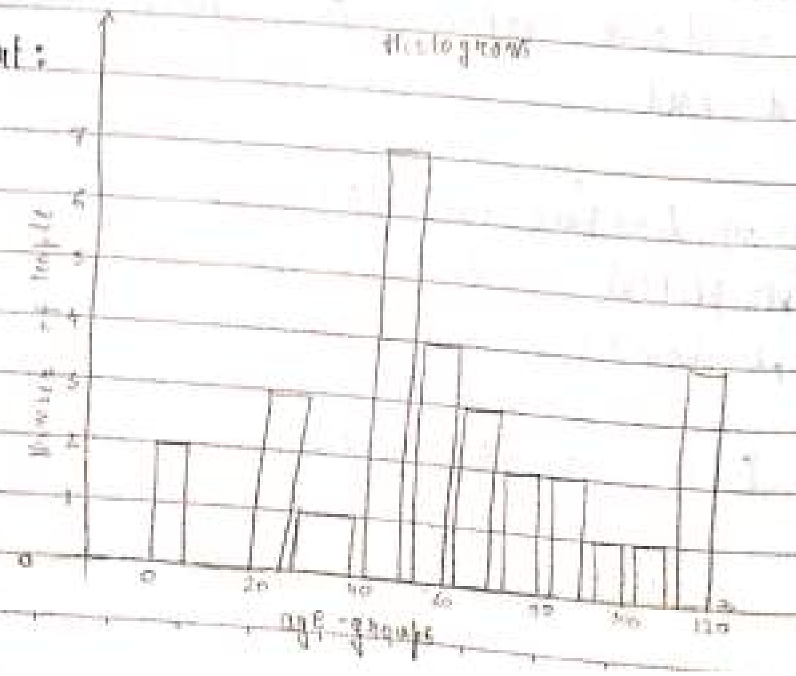
As scatter and other plotting are not best suited as it would create confusion.

here we use bins to divide into intervals.

★ WAP to show execution of Histogram:

```
import matplotlib.pyplot as plt
population_age = [22, 55, 62, 45, 21, 22, 34, 42, 42, 4, 2, 102, 95, 85,
        110, 120, 70, 65, 55, 111, 115, 80, 75, 65, 54, 44, 43, 42, 48]
bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]

plt.hist(population_age, bins, histtype='bar', rwidth=0.5)
plt.xlabel('age groups')
plt.ylabel('Number of people')
plt.title('Histogram')
plt.show()
```

Output:



NOTE: from bins we use interval (0-10), (10-20), (20-30)...
now it scans population age.
and we get values in b/w of the 2 plots from mean...

Various properties that we can apply on histograms:

NOTE: All the properties that we applied on the scatter plot would apply in same manner.

* Use color property to change color

* 'legend()': it is used to define Multi color bars
  eg: (RED) → age > 10.
  (GREEN) → age > 20.

* For Multi-color: Just make a list

```
x = ['A', 'B', 'C', 'D']
color = ['BLACK', 'GREEN', 'PINK', 'BLUE']

plt.hist(population_age, bins, histtype='bar', c=color)
```

NOTE: a) Mention no. of color's acc to no of bins (or bars)
b) In histogram we can't give multi color's i.e we can only give one color

* To change the interval just in bin mention interval accordingly eg: bins = [0, 20, 40, 60, 80, 100].

NOTE: The interval period must be same for each interval otherwise it could generate error.

P.T.O

Pie-chart():

line plot, scatter plot and every other plot works on only two series i.e x-axis & y-axis.

whereas Pie works on only one series That is either x-axis, y-axis.

i.e it works on slices

Eg:

```
a = np.array([10, 20, 30, 30, 20])
plt.pie(a)
plt.show()
plt.title('Pie chart')
```

#

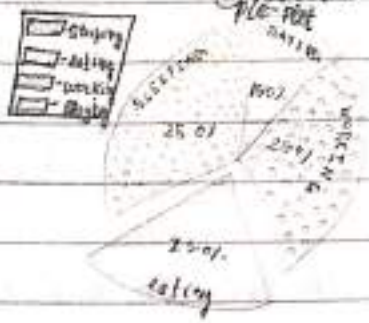
```
a = np.array([10, 20, 30, 30, 20])
mylabels = ['Sleeping', 'Playing', 'eating', 'reading', 'watching']
plt.figure(figsize = (8, 8))          # To change figure size
plt.pie(a, labels = mylabels)
plt.legend(loc = "upper left")

plt.show()
```

For every Property:

```
import matplotlib.pyplot as plt
slices = [35, 25, 25, 15]
activities = ['sleeping', 'eating', 'working', 'playing']
cols = ['c', 'm', 'r', 'b']          # mentioning various
plt.pie(slices,
        labels = activities,
        colors = cols,
        start angle = 90,
        shadow = True,
        explode = (0, 0.1, 0, 0)
```

Output:

autopct = '%.1.1f %%')
plt.legend(title = 'hobbies')
plt.title('Pie Plot')
plt.show()



≡ Qualities of various properties (w.r.t Prev Program)

* Explode: To show the 2nd slice a little different & it counts A.C.W (Anti-clock wise).

Eg:

* Autopct: Syntax: autopct = "%.1.1f %%"

auto percentage — integer value — no of decimal's or no. of float values we want to show after decimal

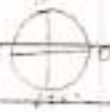
eg: would be
30.0%
25.0%
25.0%
20.0%

* Shadow: Syntax: shadow = True

when we pass boolean value first letter should be capital. rest should be small.

Shadow function generates small amount of shadows under Pie-chart.

* Start angle: By default start angle is 0° i.e



But with this function we can start plotting from 90° and it plots anticlockwise.

Area chart:

Area plot is used to compare same values. Area plot is similar to line plot.

But there we just draw a line whereas in area chart with line we define a region also mention the area under the line.

eg:

```
import matplotlib.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()
```

Out Put

Q. WAP To show execution of Area chart:

```
import Numpy as np
import matplotlib.pyplot as plt

x = np.array ([5, 7, 8, 9, 10])
y1 = np.array ([1, 7, 8, 5, 6])
y2 = np.array ([6, 8, 7, 4, 2])
y3 = np.array ([7, 8, 9, 10, 12])

plt.title ('Area chart')
plt.label ( )
plt.legend (loc = 'upper right')
c = ["yellow", "orange", "Blue"]
plt.stackplot (x, y1, y2, y3, colors=c)

plt.show ()
```

Working: The plotting of x, y1 remains as usual. But, whensoever we want To execute y2, Then Value of x-axis remains same. But for y-axis it would Take value $(y_1 + y_2)$. ie here first Plot would be $(5, 1+6) = (5, 7)$. As it would Take y_2 as base & y_3 while we'll Plot y_3.

P.T.O

Word Cloud:-

```
Import Matplotlib.pyplot as plt
from wordcloud import WordCloud

text = "Python Programming"          # (Where text is just a variable)
wc = WordCloud().generate(text)      # it is a function of wordcloud library

plt.imshow(wc)
plt.axis('off')                      # it displays axis or not
plt.show()
```

To install wordcloud library:
⇒ Go to anaconda Prompt
⇒ PiP install wordcloud

How To make wordcloud of a text file :->

☞ copy Text from wikipedia and Paste on notepad
Save this file on Desktop and save this file where anaconda folder.

ie C-Drive -> user -> 9188 -> Anaconda

Or

if not saved in anaconda folder Then write whole source address of file

ie when generating word cloud:

P.T.O...

// We a function to convert whole Text file into random word cloud:

Acc to word cloud function it shows only those words that are repeated again & again

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud

text = open('Python txt', 'r').read()          # Three modes: 1) read() 2) write() 3) append()
wc = WordCloud().generator(text)               # To enable read mode of a function of read()

plt.imshow(wc)                                 # if file is not saved in anaconda folder then give whole file source path here
plt.axis('off')
plt.show()
```

// If we want to remove a word from word cloud:

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS

text = open('Python txt', 'r').read()
wc = WordCloud(stopwords = {"PYTHON"}).generate(text)
plt.imshow(wc)
plt.axis("off")
plt.show()
```

// To show all the stop words in file that are not included in wordcloud (From the Text file)

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud, stopwords
text = open('Python.txt', 'r').read()
wc = WordCloud(stopwords = STOPWORDS).generate(text)
print(STOPWORDS)                               # This commands is used to display all stopwords
plt.imshow(wc)
plt.axis('off')
plt.show()
```

How can we read image file:

☆ Download Twitter logo. (Save with extension .jpg in

if we want to read image, then we are using array as image is 2-Dimension

can be replaced using row & column

minimum image size 512 x 512.

WAP to show execution of wordcloud on image

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS

import PIL.Image                    # library to read image
import numpy as np                  # here change all backward slash into forward. if source path is in file

text = open('Python.txt', 'r').read()
python_mask = np.array(PIL.Image.open('...'))   # mask is a library to create matrix of image
wc = WordCloud(stopwords=STOPWORDS).generate(text)   # execution will create image & give acc to my text

plt.imshow(wc)
plt.axis('off')
plt.show()
```

Extra properties that we can write:

```
mask = Python_mask,
background_color = "BLACK",
contour_color = "RED",
contour_width = 3
```

contour means out line

Pandas:

Pandas library is used for data manipulation.
Pandas is a Python library used for working data sets.

it is used for data analysis in Python & it was developed by Wes McKinney in 2008.

it has functions for analysing, cleaning, exploring and manipulating data.

Pandas is well suited for different kinds of data such as tabular data with heterogeneous typed columns, ordered and unordered time series, data arbitary Matrix data with rows and columns labels and also work on unlabeled data or any other form of observation or statistical data sets.

The Pandas Provides two data structures for processing the data, i.e ① Series
② Data frame

① Series: it is defined as a one-dimensional array that is capable of storing various data types.
The row labels of series are called index.
it is one-Dimensional.

② Data frame: it is a widely used data structure of Pandas and works with a Two Dimensional array with labeled axes (rows and columns).
it can be seen as a dictionary of series structure where both rows and columns are indexed. it is denoted as "columns" in case of columns and "index" in case of rows. it is Two-Dimensional.

WAP to create a simple Pandas series from a list:

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

Output:

```
0  1
1  7
2  2
dtype: int64
```

WAP to create series with own labels: or (Index).

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a, index = ["x", "y", "z"])
print(myvar)
```

Output:

```
x  1
y  7
z  2
dtype: int64
```

[NOTES: Whenever we give index as string then data type changes in object. As object has features of all data types

eg:

Name	id	fee	Grade
A	2	200.50	9.5
↓ string	↓ int	↓ float	↓ float

WAP to create series without passing any values:

```
import pandas as pd
import pandas as np
s = pd.Series([1, 2, 3, np.nan, 5])
print(s)
```

Output:

```
0  1.0
1  2.0
2  3.0
3  nan
4  5.0
dtype = float64
```

WAP To create a series using dictionary:

```
import Pandas as Pd
calories = { "day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories, index = ["day1", "day2"])
print(myvar)
```

Mentioning dataset name, then dictionary index we want to call

output:

day1	420
day2	380
dtype:	int64

WAP To create a data frame (2-D) from two series:

```
import Pandas as Pd
data = {
    "calories": [420, 380, 390],
    "duration": [50, 40, 45]
}
myvar = pd.DataFrame(data)
print(my var)
```

Here we passed two individual series under single head.

Output:

	calories	duration
0	420	50
1	380	40
2	390	45

NOTE: To convert Data frame to csv

```
myvar.to_csv('data')
```

here data is a user defined file name.

WAP To print date_range as index (series (1-D)) using Pandas:

```
import Pandas as Pd
import numpy as np

d = pd.date_range('20200301', periods=10)
print(d)
```

output: DateTime Index(['2020-03-01', '2020-03-02', '2020-03-03', ... '2020-03-10'],

WAP To Print series with random No's...

```
import numpy as np
var = np.random.randn(10)
print(var)
```

this is used to generate various no here it float output less than 1

if we want to print integer randint(, ,)
↓ starting range ↓ ending range ↓ no. of nos. we want to print

WAP to convert random no. into dataframe:

NOTE: way to write "DataFrame".

```
import panda as pd
import numpy as np
var = np.random.randint(10,20,6)
a = pd.DataFrame(var)
print(a)
```

OutPut

0	15
1	19
2	15
3	13
4	19
5	19

WAP to create dataframe using date as index:

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(10,4), index=d, columns=
        ['A','B','C','D'])
print(df)
```

↓ no. of rows ↓ no. of col

Out Put:

	A	B	C	D
2020-03-01	[illegible]	[illegible]	[illegible]	1.8143
2020-03-02	1.12770	1.6580	0.340	0.3918
2020-03-03	2.37191	1.0139	1.413	1.4262
2020-03-04	0.25310	-0.3256	0.257	-0.5786
2020-03-05	320465	1.377	1.7544	0.6753
2020-03-06	-0.541231	-0.2018	-0.14087	2.1482
2020-03-07	0.561195	[illegible]	-0.94912	0.4449
2020-03-08	0.621350	0.1123	0.10483	-0.3408
2020-03-09	0.400041	1.023518	-0.69?	-0.02855
2020-03-10	-1.694718	0.2??1?	0.1?56	0.12548

WAP to create series using date index:

```
import Pandas as Pd
import Numpy as NP
df1 = pd Series (np.random.randn(10), index = d)
```

output:

2020-03-01	-0.668863
2020-03-02	1.278441
2020-03-03	-0.276605
2020-03-04	-1.48807
2020-03-05	0.74688
2020-03-06	1.62410
2020-03-07	-0.766112
2020-03-08	-0.6455
2020-03-09	-1.7184
2020-03-10	-0.11215

WAP to change the name of columns:

```
import numpy as np
var = np.random.randint (10, 20, 5)
a = pd.DataFrame (var, column = ['a', 'b', 'c', 'd', 'e'])
print(a)
```

output:

	a	b	c	d	e
0	-1.2559	.	.	.	-0.12237
1	.				
2	.				
3					
4					
5					
6					
7					
8					-0.1087
9	-0.02545				

WAP to create Dataframe using dictionary:

```
import Pandas as Pd
import Numpy as NP
df = Pd.Dataframe ( {'A' : [1, 2, 3, 4],
'B' : Pd.Timestamp ('2020326'),
'C' : Pd.Series (10, index = list (range(4))),
'D' : np.array ([50]*4),
'E' : pd.Categorical (['Train', 'False', 'True', 'False']),
'F' : 'Hello World'})
Print (df)
```

Output:

	A	B	C	D	E	F
0	1	[illegible]	[illegible]	[illegible]	True	hello world
1	2	[illegible]	[illegible]	[illegible]	False	hello world
2	3	[illegible]	[illegible]	[illegible]	True	hello world
3	4	[illegible]	[illegible]	[illegible]	False	hello world

What if we want to fetch first specific no of rows

```
import pandas as pd
import numpy as np
df = pd.dataframe (np.random.randn(5))

print (df.head(5))          # it returns first 5 rows by default if no parameters are given else it returns that 5 elements
print (df.tail(3))          # it returns last 3 elements by default if returns last 5 rows i.e. if no parameter passed
```

Data views:

To view the data frame elements

eg: df.index

output:

Int64Index [0, 1, 2, 3], dtype='int64'

eg: df.columns

output

Index ['A', 'B', 'C', 'D', 'E', 'F'], dtype='object')

SELECT a single column:

```
df['C']
```

output:

```
0    10
1    10
2    10
3    10
Name: C, dtype: int64
```

Describe (): find mean, Standard deviation, count, min, max, Quartile deviation 25%, 50%, 75%.

WAP To execution of Describe:

```
import numpy as np
import Pandas as Pd

var = np.random.randn(10,5)
a= pd.DataFrame(var, columns=['a','b','c','d','e'])

Print(a.Describe())
```

Output:

	A	B	C	D	E
count					
mean					
std					
min					
25%					
50%					
75%					
max					

WAP To fetch single value

```
print(a['a'][1])
```

'a' → column name; 1 → row name

PTO...

if we want to fetch specific row & columns

```
Print (df [0:3])
```

output:

	A	B	C	D	E	F
0	1	2022-01-26	10	50	True	Hello World
1	2	2022-01-26	10	50	False	Hello World
2	3	2022-01-26	10	50	True	Hello World

if we want to fetch based on some condition:

```
Print (df [df ['A'] > 0])
```

it would Print all rows & columns. But only where the value of column 'A' is greater than 0.

output:

	A	B	C	D	E	F
0	1	2022-01-26	10	50	True	Hello World
1	2	2022-01-26	10	50	False	Hello World
2	3	2022-01-26	10	50	True	Hello World
3	4	2022-01-26	10	50	False	Hello World

if we want to fetch specific columns & all rows:

```
print (df.loc [ :, ['A','C']]
```

(: → this means all rows; ['A','C'] → two columns)

output:

	A	C
0	1	10
1	2	10
2	3	10
3	4	10

INFO (): This function returns all information about Dataframe i.e. no. of entries, date of creation, no. of columns, Datatype of each column etc, & memory used by dataframe also.

Syntax: `Print (df.info())`

Shape (): This function returns the no. of rows and columns.

Syntax: `Print (df.shape)`

output: (10,5)

If we want to insert data into dataset:

df . loc [1, 'A'] = 200

(df → data frame; loc → location; 1 → index of row; 'A' → index of column; 200 → value)

It updates value at location [1, 'A']

If we want to insert column:

df . loc [1, 'F'] = 300 # insert a column

('F' → new column name)

Output:

	A	B	C	D	E	F
0	-	-	-	-	-	nan
1	-	-	-	-	-	300
2	-	-	-	-	-	nan
3	-	-	-	-	-	nan

Rest values are filled with nan.

Other way:

df = df . reindex (index = d[0:5], columns = list (df.columns) + ['E'])

df . loc [d[0] : d[2] , 'E'] = 1,2,3

Print (df)

Output:

	A	B	C	D	E
2020-03-01	-	-	-	-	1.0
2020-03-02	-	-	-	-	2.0
2020-03-03	-	-	-	-	3.0
2020-03-04	-	-	-	-	nan
2020-03-05	-	-	-	-	nan

WAP To find missing valuse

Print (df . isnull ())

If value is nan it returns True otherwise False in same table.

Output:

	A	B	C	D	E
2020-03-01	False	False	False	False	False
2020-03-02	"	"	"	"	False
2020-03-03	"	"	"	"	True
2020-03-04	"	"	"	"	True

→ क्योंकि nan जहाँ था वहाँ True. nan.

WAP To fill all nan values with single value

```
df.fillna(value=5)
```

Output:

	A	B	C	D	E
2020-01-01	-	-	-	-	1.0
2018-03-02	-	-	-	-	2.0
2020-05-03	-	-	-	-	3.0
2020-07-04	-	-	-	-	4.0
2018-09-05	-	-	-	-	5.0

WAP To drop Null values:

```
df.dropna()
```

Output:

	A	B	C	D	E
2020-01-01	-	-	-	-	1.0
2018-02-02	-	-	-	-	2.0
2018-03-03	-	-	-	-	3.0

WAP To insert various values at nan:

for This we use slicing:

```
df.loc[0:5, 'F'] = [1, 2, 3, 4, 5, 6]
```

(0:5 → row index; 'F' → column name)

if we want To delete a row from dataset:

```
df:drop([row index])   # To delete a row we use drop function & give row index
df
```

Extension:

if we write 'df' again Then The dataframe would show deleted value also.

i.e drop function doesn't delete The value Permanently.

WAP To Permanently delete a value:

```
df.drop([1], inplace=True)
```

By default the inplace value is false. when True this function Permanently delete a value.

WAP To delete a column:

```
df.drop(ColName, axis=1, inplace=True)
or df.drop(['F'], axis=1, inplace=True)
```

NOTE: axis=1 (it defines column)
axis=0 (it defines row) (by default 0 only).

WAP To find mean, median of a Particular column:

```
df[('A')].mean()
    column
df[('A')].median()
```

Word Cloud...

NOTE: Already Discussed...

Open Github and download dataset named tips:

Steps → 1) Type on browser github/seaborn/tips ↵
2) Right click on Raw
3) And then click on save as.
4) And open in Excel
5) Open Jupter notebook and import Python library

WAP To read a data set using PANDA:

```
import pandans as pd
var = pd.read_csv("tips.csv")
```

- read_csv: File extension i.e .xls, .txt etc
- "tips.csv": File name if saved in Anaconda library. otherwise source address of file.
- Stands for: Common separated values

Violin graph:

data we can't represent into bars or bar form i.e. day, night, evening.

To deal with "Categorical data". Like total_bill, tip, sex, day, time

In this type of data we can't implement scatterplot graph etc. As they work with only two columns.

Secondly data is not numerical and they don't work on data other than Numerical. (bar, scatter plot, areaplot etc).

∴ To overcome this issue we use violin graph. it help To compare & represent violin graph, it also helps compare categories for a specific time period.

To show data set:

import pandas as pd
var = pd.read_csv("tips.csv")
var

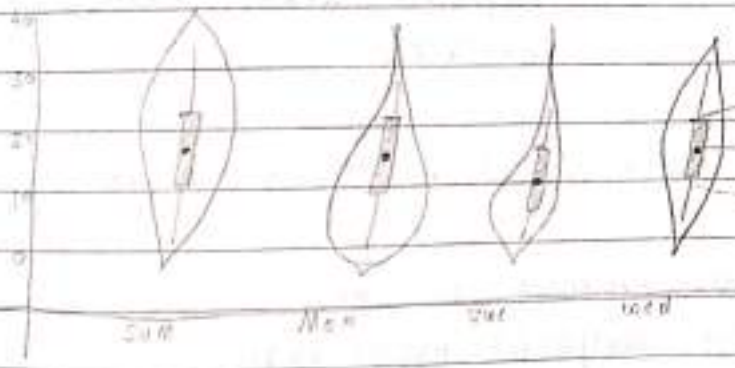
WAP To implement violin graph:

Seaborn: library used for plotting graph with the help of matplotlib, Pandas and Numpy. it helps To present univariate & bi-variate data. it is a advance package of matplotlib as here with visualisation it provides with more enhanced functions. (like more accurate data, categorical comparisons etc.).

import seaborn as sns
import matplotlib.pyplot as plt

sns.violinplot(x="day", y="total_bill", data=var)
plt.show()

Output:



here took x variable as days
y variable as total_bill.

IQR (Inter Quartile Range): it is way To measure the spread of the middle 50% of a dataset.
Quartile 1 → 25%
Quartile 2 → 50%
Quartile 3 → 75%

Extra: to change axis, using linewidth.
Just x='total_bill', y="day"
line width = 5

NOTE: we can create violin graph of a single column also.

WAP to create violin graph according to Time:

(here two attributes i.e. dinner, lunch based on downloaded database tips.)

(# i.e to create a separate Time graph including days & bill column.)

```
Import seaborn as sns
Import matplotlib.pyplot as plt
sns.violinplot (x="day", y="total-bill", data=var,
                hue="time")

plt.show()
```

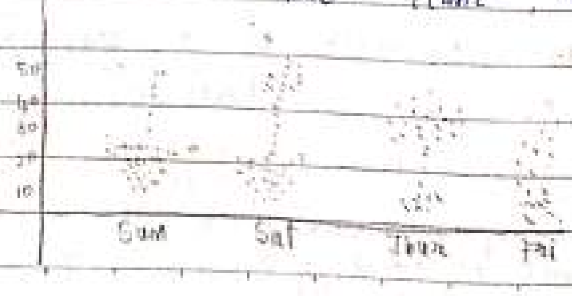
hue creates a box at top right corner
▨ - Lunch
▭ - Dinner

Swarm plot: here there is no over lapping in comparison of scatterplot. rest properties remain same like color, width etc.

WAP to show execution of swarm plot:

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.swarmplot (x="day", y="total_bill", data=var,
               hue="time", marker="x")
```

Output:



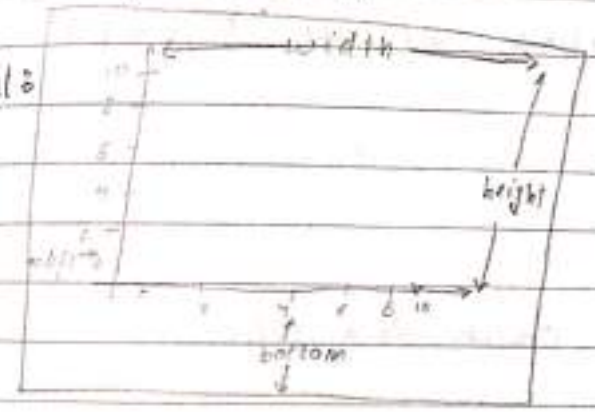
NOTE: if data set,
only numerical & size very large, then use histogram.
if categorical with numerical, then we have two ways i.e violin plot, swarm plot.

Working with Text:

How to print text in a graph:

```
import matplotlib.pyplot as plt
fig = plt.figure()
axis = fig.add_axes([0.0, 0, 1, 1]) # it defines [l, b, w, h]; i.e it helps to do indentation of figure.
ax.axis([0, 10, 0, 10])  # range of x-axis → [0,10]; y-axis → [0,10]
plt.show()
```

Output:



To give title, x-label, y-label & print text at specific coordinates:

```
import Matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axis([0, 0, 1, 1])
ax.set_title('chart')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
```

WAP To Print "Python Programming at (2,2) Pts:

```
ax.text(2, 2, 'Python Programming') # Coordinate
ax.axis([0, 10, 0, 10])
plt.show()
```

with the help of text function we can write a Te

WAP To change size of Text, Color:

```
ax.text(2, 2, 'Python Programming', fontsize=20, color='
```

Extension: WAP To Print $e = mc^2$

```
ax.text(3, 3, '$e = mc^2$')
```

To print equation we use '$ Symbol'.

Seaborn:

* WAP To read online data set:

```
import Pandas as pd
import Matplotlib.Pyplot as plt
import seaborn as sns

tips = sns.load_dataset("tips")
tips
```

Let know if we use To see downloaded dataset we just give its Path

load dataset is a function from seaborn which helps to read data set online.

Output:

	total_bill	tip	sex	smoker	day	Time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Page No.:
Date:
YOUVA

Seaborn: Seaborn is a Python data visualization library based on Matplotlib

it Provides a high level interface for drawing attractive and informative statistical graphics.

It is used for categorical data, That can be represented in strings

whereas in Matplotlib it deals with numerical data.

Difference between Seaborn and Matplotlib?

Matplotlib:	Seaborn:
• it is a Python library used for Plotting graphs with the help of others libraries like numpy and Pandas.	• it is also a Python library used for plotting graphs with the help of Matplotlib, Pandas and Numpy. it is a superset of Matplotlib library.
it deals with univariate data	it deals with bi-variate data.
it deals with numerical data.	it deals with categorical data, That can be represented in strings also.

Ques Difference between Categorical Plot and relation Pl

Ans Relational Plot: it is used for visualizing the statistical relationship between two data points

it is used for numer... Type of data

Categorical Plot: it allow us to choose a numer... variable like age.

and plot distribu... of age for each category.

Visualizing categorical data:

In relational plot we used various visual representations to show the relationship between Multiple variables in a data.

But in those examples we focused on cases where the main relationship was b/w two numerical variable

But if one of the main variable is "categorical" (Divided into Discrete groups) then comes the need for more specialized approach

In seaborn there are several ways of visualizing data, i.e categorical in nature just like relational plots (rel-plot, scatterplot, line plot).

There are many ... level functions for plotting categorical data using cate...

There are Three different families:

Categorical scatterplot: a) stripplot() (with kind="strip") — The default
b) swarmplot() (with kind="swarm")

Categorical Distributions: a) boxplot() (with kind="box") for
b) violinplot() (with kind="violin")
c) boxenplot() (with kind="boxen")

Categorical Estimate plot: a) PointPlot() (with kind="point")
b) barPlot() (with kind="bar") re
c) countplot() (with kind="count")

Categorical scatterplots:

```
tips = sns.load_dataset("tips")
sns.catplot(data=tips, x="day", y="total_bill")
```

Categorical plot

=> In seaborn.
Here we need not, need To mention labels on x-axis
My y-axis as we did in matplotlib,
here it automatica
-lly labels The data.

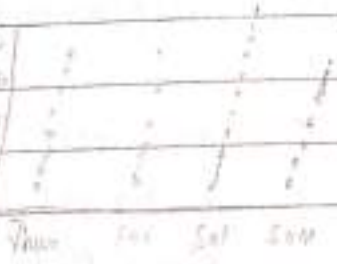
Out Put:

F.T.O.

To remove over lapping:

sns.catplot(data=tips, x="day", y="total_bill", jitter

Output:

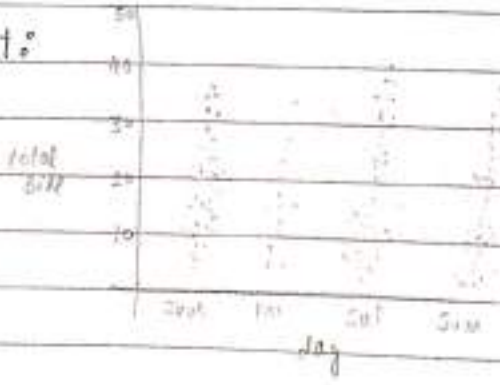


The jitter parameter controls the magnitude of jitter or disables it altogether.

This type of plot is sometimes called a "beeswarm" and is drawn in seaborn by swarmplot() which is activated by setting kind="swarm" in catplot.

sns.catplot(data=tips, x="day", y="total_bill", kind="swarm")

output:



we can use another property hue. NOTE: categorical plot do not currently plot size or style semantics.

sns.catplot(data=tips, x="day", y="total_bill", hue="sex", kind="

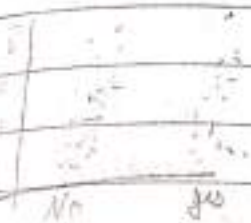
it defines the two types of male, female with different colour plots

* a small box appears in out put.

WAP To order Category:

SNS: catplot (data = tips, x = "smoker", y = "tips", order = ["No", "yes"])

output:

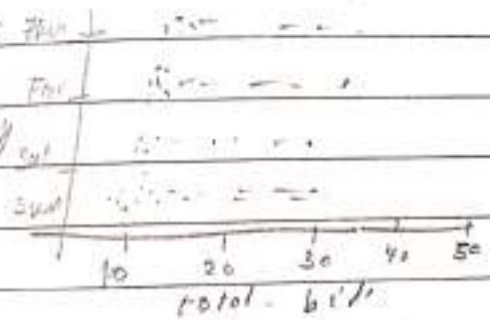


WAP To represent names on Vertical axis → categorical data

used when sometimes names too long.

SNS. catplot (data = tips, x = "total_bill", y = "day", hue = "time", kind = "swarm").

output:



Comparing Distributions:

As the size of dataset grows, categorical scatter plots become limited in the information.

when this happens, there are several plot that facilitate easy comparison at categorical level.

P.T.O:

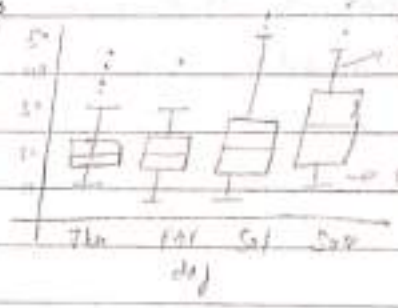
Box Plot:

This kind of plot shows the three quartile values of the distribution along with extreme values.

The "whiskers" extends to points that lie within 1.5 IQR (Inter Quartiles Range) of the lower and upper quartile, and the observations that fall outside this range are displayed independently.

eg: sns.catplot(data=tips, x="day", y="total_bill", kind="box")

output:



Boxen Plot:

it draws a plot that is similar to a box plot but optimized for showing more information about the shape of the distribution (it is best suited for larger datasets)

eg: diamonds = sns.load_dataset("diamonds")
sns.catplot(data=diamonds.sort_values("color"), x="color", y="Price", kind="boxen")

output:

Violin plot:

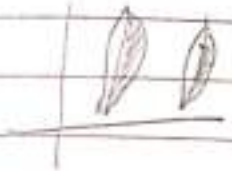
* combines a box plot with Kernel density estimation procedure described in distribution tutorial.

eg: sns.catplot(data = tips, x = "total_bill", y = "day", hue = "sex", kind = "violin",)

* output box plot is shown within violin plot

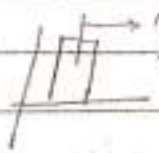
* if we add extra Property, split = True)
Then the violin graph gives splitted value that tips in comparison

output:



* Just like violin, box plot we can plot bar, count, point.

NOTE: Output in Box Plot:



* NOTE: if we want to compare two values then following plot:

strip, points, box, violin, box, swarm.

Relational Plot: here we can describe relation btw data

Download the dataset carprice Prediction from Inte... (Kaggle), Then save on system & read it

```
Import matplotlib.pyplot as Plt
IMport seaborn as sns
Import pandas as Pd

df = pd.read_csv ('CarPrice.csv')   # i.e if above Anaconda libs otherwise pip... which lib.
```

Output: Car_ID Symboling CarName fueltype aspiration ...

0						
1						
2						
3						
.						
.						
203						
204						

205 rows × 26 columns

Wap to print top five entries only:

```
df.head()
```

Output:

	Car_ID	...	...	...
0	-	-	-	-
1	-	-	-	-
2	-	-	-	-
3	-	-	-	-
4	-	-	-	-

By default it returns 5 rows, otherwise we can mention accordingly.

NOTE: We can also draw some with matplotlib using line plot, scatter plot. But it deals with only numerical datatype.

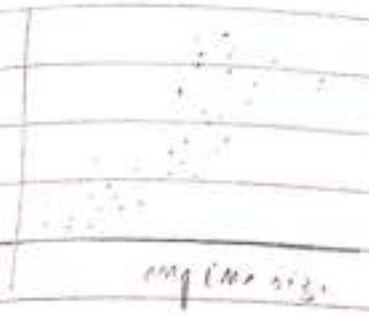
∴ we use relational plot of seaborn library as it can handle larger data size than matplot lib.

regression. & it is also used to describe relationship between two attributes. Relation plot is also known as

WAP To compare two attributes in a dataset

```
sns.replot(x="enginesize", y="Price", data=df)
```

output:

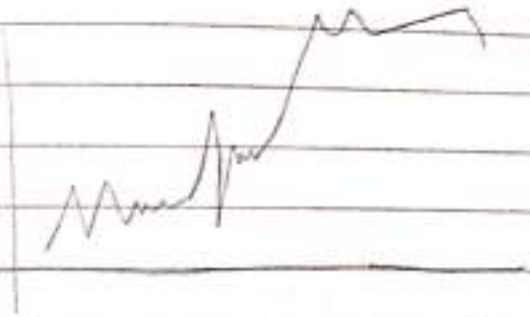


NOTES: Here both attributes, i.e. engine size and price are directly proportional to each other automatically increases also.

WAP To Define relational data in line format:

```
sns.replot(x="enginesize", y="Price", data=df, kind="line")
plt.show()
```

output:

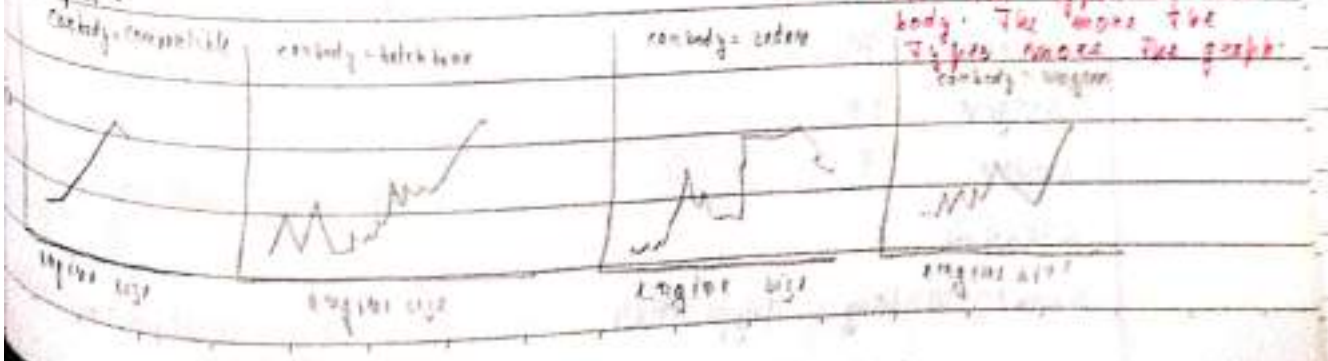


Extensions:

```
sns.replot(x="enginesize", y="Price", data=df, kind="line",
           col="carbody")
plt.show()
```

it defines facet grid column with multiple rows line graphs. i.e it represents all the types of car body. The more the types more the graphs.

Output:



WAP To show relational using bar graph:

```
sns.barplot (y = "enginesize", x = "carbody", data=df)
plt.show()
```

Output:

WAP To show relational using bar graph using one attribute

```
sns.countplot (x= "carbody", data =df)
plt.show()
```

Output:

WAP To show count no. of categories of carbody:

```
df ["carbody"]. value_count ()
```

Output:

sedan	96
hatchback	70
wagon	25
hardtop	8
convertible	6

Name: carbody, dtype: int64

it defines data in 2D form value_count helps it count.

with countplot we can count no. cars & visualize them

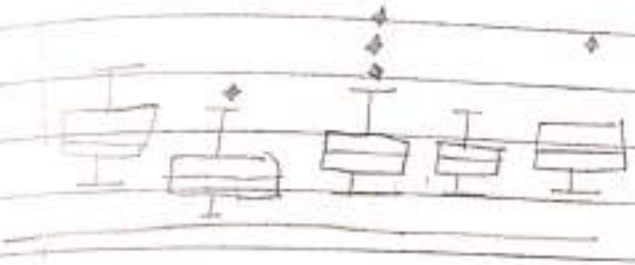
sns. countplot (x= "carbody" ...

plt.show()

Pgm to show boxplot to show relational plots

```
import numpy as np
sns.boxplot(y="enginesize", x="carbody", data=df)
plt.show()
```

output:



Pgm to relational plot using histogram:

```
import numpy as np
sns.histplot(y="enginesize", x="carbody", data=df)
plt.show()
```

Output:

As two values (attributes) are given
∴ The output is blurry.
If we give single value of attribute then it would plot histogram.

Distribution Plots:

It is used for uni-variant and bi-variant data.

- uni-variant: when we pass just 1 column → it represents only one axis i.e x-axis or y-axis
- bi-variant: when we pass two columns → it represents both axis i.e x & y-axis both

Why do we use?
Help to visualize data on basis of given data.

functions of Distribution Plot:

1) Distplot()

2) jointplot()

3) Pairplot()

WAP to read dataset tips:

```
df = pd.read_csv('tips.csv')
df
```

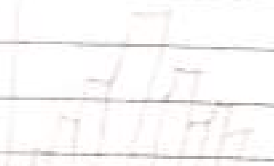
Output:

	total_bill	tip	sex	smoker	day	time	size
0	[illegible]	[illegible]	Female	No	Sun	Dinner	[illegible]
1	[illegible]	[illegible]	Male	No	Sun	[illegible]	[illegible]
2	[illegible]	[illegible]	Male	No	Sun	[illegible]	[illegible]
3	[illegible]	[illegible]	Male	No	Sun	[illegible]	[illegible]
4	[illegible]	[illegible]	Female	No	Sun	[illegible]	[illegible]
...	...	...	...	...	...	...	...
[illegible]	[illegible]	[illegible]	Male	Yes	Sat	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	Female	No	Sat	[illegible]	[illegible]

WAP to show execution of Distplot()

```
sns.set_style('whitegrid')
sns.distplot(x='total_bill', data=df)
```

Out Put:

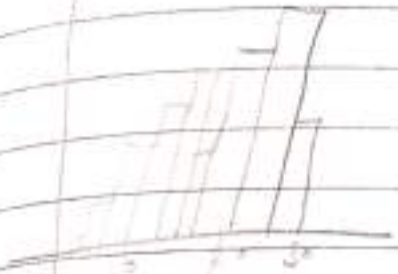


NOTE: it displays output in form of histogram. histogram is general bi-variant i.e it takes two columns whereas distplot is based on univariant & if we pass two columns then graph gets mutilated is blurred. & area with darker region means more values lie in that area.

to control number of bars.

sns.displot(x='total_bill', data=df, bins=20)

output:



bins refers to no of bars

The range having maximum no. of bars shows that most values lie in that range.

to show outline rather than histogram:

here we use KDE for it. KDE stands for Kernel Density Estimation.

it is used to show outlines rather than histogram.

Code: sns.displot(x='total_bill', data=df, kind='kde')

output:

WAP To draw distinguished graphs of Particular column

sns.displot(x='total_bill', data=df, col='smoker')

output:

Joint Plot:

it shows output as scatterplot & histogram by default.

it helps to join Multiple graphs at a single time at single dataset.

sns.jointplot(x='total_bill', y='tip')

Out Put:

NOTE: whereas ever the plot would intersect there we should have scatterplot & for histogram as it a univariant. therefore hugs is drawn on base of tips attribute

Regression lines: it is used to represent Prediction it helps to Predict values for a Particular points.

Pair Plot: it also demonstrate the data in form of scatterplot & histogram (i.e by Default).

it draws various graphs for each comparison.

Syntax: sns pairplot(df)

Correlation:

It tells that how much one attribute is correlated to a value.

we can't find correlation of data which are under labels.

i.e correlation works only on Numerical data.

heat map(): Just like correlation heat map also deals with just numerical data.

it draws a heat graph.

The more darker the color The less the relation with two variants.

i.e numerically (< then 5).

Inversely the lighter the shade, stronger the relation.

VAP To show execution of correlation & heat map:

```
tc = df.corr()
tc
sns.heatmap(data= tc)
```

Out puts

	total_bill	tip	size
total_bill	1.0000	0.6757	0.5983
tip	0.6757	1.0000	0.4893
size	0.5983	0.4893	1.0000

Extension:

sns.heatmap (tc, annot = True) # with the value ... given ...

output:

Extension:

Extra Properties that we can use:

sns.heatmap (tc, annot = True, linewidth = 3, linecolor = ...
cmap = 'coolwarm')
or
'Plasma' → Three kinds of default color maps
or
'Magma'

Practise Questions:

1) Download the dataset "Population" from kagle

2) Implement the following function

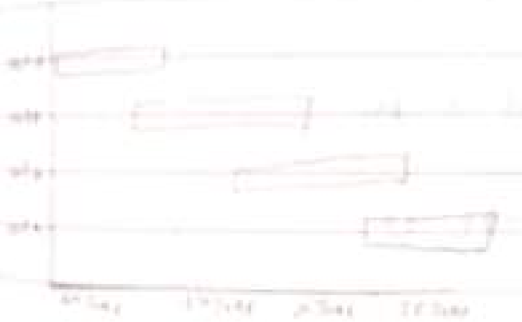
a) shape
b) drop
c) head
d) add a column into dataset
e) how to replace nan with any value
f) Draw histogram and boxplot
g) Calculate mean, median & q1 & q2

P.T.O

Statistics Dimension # how data is a ... [illegible]

```
data.shape
data.tail(2)
data.head()
data.loc[2, 'new column'] = 300
data.fillna(30)
...data = data.head()
sns.catplot(x='stroke', y='price', data=data, kind='box')
data.mean()
data.median()
data.describe()
sns.histplot(x='stroke', data=data)  # ...
sns.histplot(x='stroke', y='price', data=data)  # ...
```

Gantt Chart:



```
import matplotlib.pyplot as plt
fig, gnt = plt.subplots()
```

To set limit of x axis, y axis

```
gnt.set_ylim(0, 50)
gnt.set_xlim(0, 150)
```

To set label on x axis, y axis

```
gnt.set_xlabel('seconds')
gnt.set_ylabel('Processor')
```

```
gnt.set_yticks([10, 25, 25])
gnt.set_yticklabels(['1', '2', '3'])
gnt.grid(True)
gnt.broken_barh([(40,50)], (30,9),
    facecolor('tab:red'))
```

for another bar `gnt.broken_barh([(60,5), (110, 10)], (30,9), facecolor='tab:blue')`

3-D Plots:

WAP To Print a 3D grid:

Out Put:

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection="3d")
plt.show()
```


Out Put:

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection="3d")
x = [1,2,3,4,5,6]
y = [2,4,6,8,10,12]
z = [3,5,7,9,11,13]
ax.plot3D(x,y,z)
plt.show()
```

To draw 3D plot

Extensions: `ax.scatter3D(x,y,z)` # for scatter 3D