3° 0

B. Dheeraj Chandan
1807265
EEE

Python

20/12/20

Python Objects and Data Structure Basics

Data Types of Python (intro) :-

Name	Type	Examples
Integer	int	3 300 200
Floating point	float	2.3 4.6 10.0
strings	str	"Hello" "Dheeraj" "%&%"
Lists	list	[10, "Hi", 200.3]
Dictionaries	dict	{"a": 2, 'b': 3, "c": "Dheeraj"}
Tuples	tup	(10, "Hi", 200.3)
sets	set	{"a", "b"}
Boolean	bool	True or False

Python Numbers :-

↳ 2+1 3
2-1 1
2*2 4
3/2 1.5

* Modulo or Mod operator : %
7%4 3
50%5 0
23%2 1, 23//2 11
23%6 5, 23//6 3

Powers:
2*3 6
2**3 8

Arthematic:
2+10*10+3 105
(2+10)*(10+3) 156

Variable assignments :-

↳ a=2 # we can't use symbols /?@$%----

is used for comment

Rules ↳ Consider names in lower case
↳ avoid words such which has special meaning
i.e. 'str', 'list' ---

→ a = 2
a = ["Dheeraj", "Chandan"]
a

O/P: ["Dheeraj", "Chandan"]

if we write int a = 2
a = ["Hi", "Hello"]
in Python it is wrong

↳
```
a = 5
a          # 5
a = 10
a          # 10
a + a      # 20
a          # 10
a = a + a
a          # 20
a = a + a
a          # 40
```

↳ Type: It is a keyword used to return the type of a value. (Data-Type)

```
type(a)    # int
a = 30.1
type(a)    # float
a = ["a", "b"]
type(a)    # list
```

```
a = [1, 3]
type(a)    # list
a = {1, 3}
type(a)    # set
a = {"b", "c"}
type(a)    # set
a = {"a", 1}
type(a)    # set
```

↳ Python is Dynamic Typing & other languages are statically-Typed.

```
a = ("a", "b")
type(a)    # tuple
```

```
a = ("a", 4)
type(a)    # tuple
```

21/12/20 # Introduction to Strings

Introduction to Strings :-

→ string are sequences of characteres using the syntax of either sigle or double quotes.

ex :- "hello" 'hello' "I don't do that"

→ Strings are oodered sequences.

→ We can use indexing & slicing to grab sub-section

→ ex :- a = 'hello'

character	h	e	l	l	o
Index	0	1	2	3	4
revese Index	0	-4	-3	-2	-1

→ Slicing to grab a subsection of multiple characters

syntax : [start : stop : step]

start - numerical index for slice start

stop - " " that will go upto.

step - size of jump

↳ 'hello' # hello

"hello" # hello

"world" # world

"Hi I'm Dheeraj" # Hi I'm Dheeraj

"Count one" # Count Two

"Count Two"

---

print ("Count one") # Count One

print ("Count Two") # Count Tovo

↳ '\n' is a special character to go to next line.

'\t' " " " to give 5 spaces (Tab).

'\b' " " " to back space.

→ a = 'hellow'

a # print(a) - hellow

a = 'One \n Two \t Three'

a # One
Two Three

→ len() is a function to cont no. of character in a string.

a = 'Dheeraj'

len (a) # 7

len ('hello') # 5

Indexing & Slicing :-

↳ Indexing →

```
a = "Hello world"
a                  # Hello world
print(a)           # Hello world
a[0]               # H
a[4]               # o
a[-2]              # l
```

a[-1] # d
a[-4] # o

a[8] # r

↳ Slicing →

```
a = 'abcdefghijk'
a[2]   # c
a[-2]  # j
```

a[5] # f

a[-5] # g

```
a[2]     # c
a[2:]    # cdefghijk
a[:3]    # abc
a[3:6]   # def
a[1:3]   # bc
a[::]    # abcdefghijk
a[::3]   # adgj
a[2:7:2] # ceg
a[::-1]  # kjihgfedcba
a[-1:-7:-1] # kjihgf
a[-5:-1:-2] # ''
a[-5:-1:2]  # gi
```

23|12|20

String properties & method :-

↳ Immutability → Immutability of a string is cannot immutate or cannot change.

↳ a = "Sam

a # Sam

```
a_Last = a[1:]          # am
a = 'P' + a_Last
a                       # Pam
↳ 'Z' * 10              # ZZZZZZZZZZ
letter = 'D'
letter * 5              # DDDDD
↳ 2+3                   # 5
'2' + '3'               # 23
↳ a = "Hello world"
a.upper()               # HELLO WORLD
a                       # Hello world
a = a.upper()
a                       # HELLO WORLD
a.lower()               # hello world
a                       # HELLO WOLRD
a = a.lower()
a                       # hello world
a.split()               # ['hello', 'world']
                        # split function is to give lists as o/p.
                          string to List or to split a character
                          or delete a character/ter from a string.

a = "Hi this is a string"
a.split()               # ['Hi', 'This', 'is', 'a', 'string']
a.split('i')            # ['H', 'th', 's', 's a str', 'ng']
```

lower() → It is a function to convert a string to lowercase
Upper() — It is a function to convert a string to Upper case
split() — It is a function to split each word in a string to form lists
split('i') — It is a function to split a string by removing a entered character.

Print formating with strings :-

String interpolation — Multiple ways to format strings for printing variable in them. If we want to "inject" a variable into your string for printing

ex:- my_name = "Jose"
print("Hello" + my_name) # Hello Jose

→ 2 two methods for this : (i) .format() method
(ii) f-string (formatted string literals)

• Syntax for : (i) 'string here {} then also {}'.format('str1', 'str2')

ex:-

(i) print("This is a string {}".format('INSERTED'))
This is a string INSERTED

print("The {} {} {}".format('for', 'brown', 'quick'))
# The for brown quick

print("The {2} {1} {0}".format('fox', 'brown', 'quick'))
# The quick brown fox

print("The {0} {0} {0}".format('fox', 'brown', 'quick'))
# The fox fox fox

print("The {q} {b} {f}".format(f='fox', b='brown', q='quick'))
# The quick brown fox

print("The {f} {f} {f}".format(f='fox', b='brown', q='quick'))
# The fox fox fox

↳ a = 100/777
a # 0.12870012870012870

print("The result was {}".format(a))
# 0.12870012870012870

In case of floating no. we can use
floating formatting follows "{value : width . precision f"

print("The result was {r:1.3f}".format(r=a))
# The result was 0.129

print("The result was {r:10.3f}".format(r=a))
# The result was 0.129

width

(ii) Syntax : f'String here {Variable 1} then also {Variable 2}'

ex: name = "Jose"
print(f'Hello, his name is {name}')
# Hello, his name is Jose

name = "Sam"
age = 3
print(f'{name} is {age} years old')
# Sam is 3 years old

→ Oldest method :- (%)

Another way to perform string formatting :
The oldest method involves place holders using the
modulo operator % character
ex:-
print("I'm going to inject % here" % 'something')
I'm going to inject something here

print(
x = 'some'
y = 'more
x, y = 'some', 'more'
print("I'm going to inject %s text here, and %s text here" % (x, y))
I'm going to inject some text here, and more text here

25/12/20

Lists :-

↳ These are ordered sequences that can hold a variety of objects type.
↳ They use '[]' brackets & commas to separate objects.
↳ It can support indexing & slicing

ex:- a = [1, 2, 3, 4, 5] # unique elements
b = ['string', 100, 10.9] # all diff.. types of elements

max() - to find large number
min() - to find small number

↳ q = [1, 2, 3, 4, 5]
b = ['string', 100, 20.5]

len(q) # 5
len(b) # 3

a = ['one', 'Two', 'Three', 'Four', 'Five']

a[0] # One
a[2] # Three
a[5] # Five
a[-3] # Three
a[1:] # ['Two', 'Three', 'Four', 'Five']
a[2:4] # ['Three', 'Four']
a[1::2] # ['Two', 'Four']
a #['one', 'Two', 'Three', 'Four', 'Five']

↳ a = [1, 2, 3]
b = [4, 5]
a+b # [1, 2, 3, 4, 5]
a # [1, 2, 3]
b # [4, 5]
c = a+b
c # [1, 2, 3, 4, 5]

max(c) # 5
min(c) # 1

↳ q = [1, 2, 3, 4, 5]
q[0] # 1
a[0] = 10
q # [10, 2, 3, 4, 5]

max(a) # 10
min(a) # 2

a = ['one', 'Two', 'Three']
a[0] # one
a[0] = '1 count'
a #['1 count', 'Two', 'Three']

appened() is a function to add a new elemet to a list.

```
a=['one', 'Two', 'Three']
                    # ['one', 'Two', 'Three']
a
a.append('Four')
                    # ['one', 'Two', 'Three', 'Four']
a
a.append('Five')
                    # ['one', 'Two', 'Three', 'Four', 'Five']
a
# pop() is a function to delete an element from last
  index of a list or from any index.
a                   # ['one', 'Two', 'Three', 'Four', 'Five']
a.pop()             # 'Five'
a                   # ['one', 'Two', 'Three', 'Four']
a.pop()             # 'Four'
a                   # ['one', 'Two', 'Three']
b=a.pop()
b                   # 'Three'
a                   # ['one', 'Two']
a.append('Three')
a                   # ['one', 'Two', 'Three']
a.pop(1)            # 'Two'
a                   # ['one', 'Three']
a.pop(0)            # 'one'
a                   # ['Three']
# sort() sort() is function to sort the elements in a
  list.
a=['a', 'e', 'x', 'b', 'c']
a.sort()
a                   # ['a', 'b', 'c', 'e', 'x']
b=[4,1,8,3]
b.sort()
b                   # [1,3,4,8]
```

```
max(b)# 8
min(b) #1
```

```
a=[4,1,8,3,6]
b=a.sort()
b                  #[1,3,4,6,8]
a                  #
type(a)            # NoneType
```

None is indicates no value

```
type(b)  # list
a=[4,1,8,3,6]
a.sort()
b=a
b                  #[1,3,4,6,8]
b.reverse()
c=b
c                  #[8,6,4,3,1]
```

reverse() is a fuction to sort an elements in a list in reverse order.

Dictionary :- (Dictionaries in Python) :-

↳ Dictionaries are unordered mapping for storing objects.

Syntax - {'key1':'value1', 'key2':'value2', ...}

↳ uses {} curly braces for indicating as dictionaries.

⊖ difference b/w Lists & Dictionaries →

- ⊙ Dictionaries : objects retrieved by key name
  Unordered & cannot be sorted
- ⊙ Lists : object retrieved by location
  Ordered sequence can be indexed or sliced

↳

```
a={'key1':'value1','key2':'value2'}
a                  # {'key1':'value1','key2':'value2'}
a['key1']          # 'value1'
a['key2']          # 'value2'
```

```
b={'apple':30, 'banana':50, 'grapes':100}
b['apple']          # 30
b['banana']         # 50
b['grapes']         # 100
b                   #{'apple':30, 'banana':50, 'grape':100}
```

dictionaries can hold lists and dictionary itself.

```
a={'k1':100, 'k2':[1,2,3,4,5], 'k3':{'A':10,'B':20}}
a  #{'k1':100, 'k2':[1,2,3,4,5], 'k3':{'A':10, 'B':20}}
a['k1']   #100
a['k2']   #[1,2,3,4,5]
a['k2'][2]   #3
a['k2'][4]   #5
a['k3']      #{'A':10,'B':20}
a['k3']['A']  #10
a['k3']['B']  #20

b={'key1':['a','b','c']}
b                  #{'key1':['a','b','c']}
b['key1']  #['a','b','c']
c=b['key1']
c                  #['a','b','c']
letter=c[2]
letter             #'c'
letter.upper()     #'C'
d['key1']    #['a','b','c']
d['key1'][2].upper()   #'C'
```

```
d={'k1':100 , 'k2':200}
d                          #{'k1':100, 'k2':200}
d['k3']=300
d                          #{'k1':100, 'k2':200, 'k3':300}
d['k1']='new value'
d                          #{'k1':'new value', 'k2':200, 'k3':300}
~~d.keys()~~ ~~# dict_values([~~
d.key()      # dict_keys(['k1', 'k2', 'k3'])
d.values()   # dict_values([100, 200, 300])
d.items()    # dict_items([('k1', 100), ('k2', 200), ('k3', 300)])
```

Tuples :-

↳ Tuples are very similar to lists. They have only one key difference. (immutability)
↳ Element inside a tuple cannot be reassigned.
↳ Indicates by '()' moon parenthesis.

```
a= (1, 2, 3)
b= [1, 2, 3]
type(a)      # tuple
type(b)      # list
len(a)       # 3
len(b)       # 3
a            # (1, 2, 3)
b            # [1, 2, 3]
```

```
max(a)       # 3
max(b)       # 3
min(a)       # 1
min(b)       # 1
```

```
a= ('one', 2)
a[0]         # 'one'
t[-1]        # 2
```

count() is a function to count no. of time an element has repeated in a tuple.

```
a=('a','a','b','c','b','b')
a.count('a')        # 2
a.count('b')        # 3
a.count('c')        # 1
```

index() is a function to grab the index position of 1st occourance of an element in a tuple.

```
a=('a','a','b','c','b','b')
a.index('a')        # 0
a.index('b')        # 2
a.index('c')        # 3

a=(1,2,2,3,1)
a.count(2)          # 2
a.index(3)          # 3
```

→
```
a=(1,2,3)
b=[1,2,3]
b[0]=10
b                   # [10,2,3]
a[0]=10             # error will occour as tuple
                      doesn't support reassigment.
```

difference b/w tuple & list is that:
- Tuple doesn't support reassignement
- List always support reassignment

26/12/20

Sets :-

↳ Sets are unordered collections of unique elements.
↳ There can only be one representative of the same objec

add() is a function to add an element in a set.
clear() is a function to clear all elements in a set.
remove() is a function to remove an geleted elenet in a set.
pop() is a function to delete/eliminate 1st elemenent in a set.
max() is a function to find greater number
min() is a function to find small number

↳ ~~a. Set~~

```
a = set()
type(a)          # set
a.add(1)
a                # {1}
a.add(2)         ||
a                # {1,2}
a.add(3)
a                # {1,2,3}
a.add(5)         ||
a                # {1,2,3,5}
a.add(4)
a                # {1,2,3,4,5}
a.remove(4)
a                # {1,2,3,5}
a.remove(2)
a                # {1,3,5}
a.pop()          # 1
a.pop()          # 3
a.pop()          # 5
a.add(1)
a.add(3)
a.add(2)
a.add(5)
a.add(4)
a                # {1,2,3,4,5}
a.add('a')
a                # {1,2,3,4,5,'a'}
```

```
a = {1,2,3}
printf(max(a)) # 3
printf(min(a)) # 1
max(1,3)  #3
min(1,2) # 1
```

```
a = [1,1,1,1,2,2,3,3,3,3,3]
set(a)           # {1,2,3}
```

↳ Difference b/w dictionary & sets :

- ⊖ Sets do not contain values & it is simple collection of unique keys
- ⊖ Dictionary maps a set of objects (key) to another set of objects (values).

Booleans :-

↳ These are an operators which can convey True or False. Checks True or False.

↳ Very important when we deal with control flow and logic.

ex:-

```
True          # True
type(True)    # bool
False         # False
type(False)   # bool
1 > 2         # False
1 == 1        # True
b = None
b             #
b = 1>2
b             # False
b = 1==1
b             # True
type(b)       # bool
b = None
b
type(b)       # NoneType
```

IO with Basic Files (I/O ≈ Input or Output) :-

↳ Input and Output operations are simply performed by using basic.txt files.

#.open() is a function used to open a file & returns it as a file object.

#.read() is function which will return the specified number of bytes from the file.

%%writefile is used to write a text in a .text format as using .txt.

↳ %%writefile a.txt
hi
Im
Dheeraj

```
b= open('a.txt')
b.read()          # 'hi\nIm\nDheeraj'
b.read()          # ''
```

seek() is a function to reset the cursor & now it can read the text inside text file.

```
b.seek(0)         # 0
b.read()          # 'hi\nIm\nDheeraj'
b.read()          # ''
b.seek(5)         # 5
b.read()          # 'm\nDheeraj'
b.read()          # ''
b.seek(0)         # 0
b.read()          # 'hi\nIm\nDheeraj'
b.seek(0)         # 0
C = b.read()
C                 # 'hi\nIm\nDheeraj'
```

.readlines() is a function to grab or read lines in a text. Each line as an element or object.

```
b.readlines() # ['hi\n','Im\n','Dheeraj']
b.readlines() # []  # List format
b.read()      # ''  # String format
b.seek(0)     # 0
b.readlines(1) # ['hi\n']
b.readlines(2) # ['Im\n']
b.readlines(3) # ['Dheeraj\n']
b.read()      # ''
```

```
b.readlines()  # []
b.seek(1)      # 1
b.readlines()  # ['i\n', 'Im\n', 'Dheeraj\n']
b.seek(0)      # 0
b.read(8)      # 'im\ndheer'
b.read()       # 'aj\n'
b.seek(0)      # 0
b.readlines(2) # ['Hi\n']
b.seek(0)
b.readlines(5) # ['Hi\n', 'im\n']
```

↳ Difference blw .read() & .readlines() :

read() gives

⊖ read() function to grab every element to give as output as a gaint string.

⊖ readlines() function to grab a list where each element represents a line.

27/12/20

pwd is a command in python to know the location where the jupyter notebook is and to know where these text files are going to be saved.

```
pwd  # 'C:\\Users\\Marcial\\Pierian-DataCourses'
```

close() is a function to close a file or to stop accessing a file.

```
b.close()
b.read()  # error : I/o operation on closed file
```

↳ If want to open a closed file we can use a new file name. (using with, as)

```
with open('a.txt') as new_b:
    c = new_b.read()
c                    # 'hi\nIm\nDheeraj\n'
c                    # 'hi\nIm\nDheeraj\n'
new_b.read()         # I/o operation on closed file
```

(or)

```
with open('file.txt', mode='r') as new_file:
    b = new_file.read()
b                    # 'hi\nIm\n Dheeraj \n'
b                    # 'hi\nIm\n Dheeraj \n'
```

↳ mode='r' is to read a file
mode='w' is to write only (overwrite & create new)
mode='a' is to appened only (add on files)
mode='r+' is to reading & writing
mode='w+' is to writing & reading (overwriting existing file)

```
%% writefile new_a.txt
hi
Im
Dheeraj
```

```
with open('new_a.txt', mode='r') as f:
    print(f.read())          # hi
                             # Im
                             # Dheeraj
```

over-writing

```
with open('new_a.txt', mode='a') as f:
    f.write('\n Chandan \n Balivada')
with open('new_a.txt', mode='r') as f:
    print(f.read())          # hi
                             # Im
                             # Dheeraj
                             # Chandan
                             # Balivada
```

```
with open('new_b.txt', mode='w') as f:
    f.write('DHEERAJ CHANDAN BALIVADA')
with open('new_b.txt', mode='r') as f:
    b = f.read()             # or print(f.read())
b                            # DHEERAJ CHANDAN BALIVADA
```

27/12/20

Python - Comparison Operators

(Comparison Operators)

Comparision operators in Python :-

Operators	meaning
==	Equals to
!=	not Equals to
>	Greater than
<	lesser than
>=	Greater than or equal to
<=	lesser than or equal to

↳

```
2 == 2          # True
2 == 1          # False
'hello' == 'bye'  # False
'Bye' == 'bye'    # False
'2' == 2        # False
2.0 == 2        # True
3 != 3          # False
4 != 5          # True
2 > 1           # True
1 > 2           # False
1 < 2           # True
2 < 5           # True
2 >= 2          # True
4 <= 1          # False
'hi' == 'hi'    # True
```

Chaining comparision operators in Python with logical operators :-

↳ We use logical operators. Logical operators in a python

3 types of logical operators : (i) and
(ii) or
(iii) not

Symbol	Meaning
and	AND , if all cond?. are true then True
or	OR , if all cond?. are false then False
not	NOT , if cond?. is true then False if cond?. is false then True

↳
```
1<2                          # True
2<3                          # True
1<2<3                        # True
1<2>3                        # False
(1<2) and (2<3)              # True
1<2 and 2<3                  # True
'h'=='h' and 2==2            # True
('h'=='h') and (2==2)        # True
```

↳
```
1==1 or 2==2                 # True
1==100 or 2==2               # True
1==100 or 2==3               # False
1==100 and 2==3              # False
```

↳
```
1==1                         # True
not (1==1)                   # False
not 1==1                     # False
400 > 5000                   # False
not (400 > 5000)             # True
not 400 > 5000               # False
```

27/12/20 Python Statements

if, elif, else statements :-

↳ To control, flow of logic we use some keywords those are : if, elif, else

(i) if
(ii) elif
(iii) else

Syntax of if statement :
```
if some_condition:
    # some_code
```

Syntax of if/else statement :
```
if some_condition:
    # some_code 1
else:
    # do some_code 2
```

syntax of elif statement :
```
if some_condition 1:
    # some_code 1
elif some condition 2:
    # some_code 2
else:
    # some_code 3
```

↳
```
if True:
    print("Its True")          # 'Its True'

if 3>2:
    print("Its True")          # 'Its True'

hungry = True
if hungry:
    print("Feed me")           # 'Feed me'

hungry = False
if hungry:
    print("Feed me")           #
```

↳
```
hungry = False
if hungry:
    print("Feed me")
else:
    print("I'm not hungry")    # 'I'm not hungry
```

```
if 3>2:
    print("3 is greater")
else: print("2 is greater")   # 3 is greater.
```

↳
```
loc='Bank'
if loc=='Auto shop':
    print("Cars are cool")
elif loc=='Bank':
    print("Money is cool")
else:
    print("Dont know")        # 'Money is cool'
```

```
loc=1
if loc==1:
    print('One')
elif loc==2:
    print('Two')
elif loc==3:
    print('Three')
else:
    print('Four')             # 'One'
```

```
loc=5
if loc==1:
    print('one')
elif loc==2:
    print('Two')
elif loc==3:
    print('Three')
else:
    print('not Matched')      # 'not matched'
```

For Loops :-

↳ We can use for loops to execute a block of statements for one or number times for every iteration.

syntax:
```
a=[1,2,3]
for item in a:
    print(item)
```
1
2
3

```
for item in range(3):
    print(item)
```
1
2
3

↳ item = [1, 2, 3]
for a in item:
    print(a) # 1
              2
              3

item = [1, 2, 3]
for a in item:
    print(f'{a} is a number') # 1 is a number
                               2 is a number
                               3 is a number

item = [1, 2, 3]
for a in item:
    print('{} is a number'.format(a)) # 1 is a number
                                       2 is a number
                                       3 is a number

item = [1, 2, 3]
for a in item:
    print('Dheeraj') # Dheeraj
                      Dheeraj
                      Dheeraj

item = [1, 2, 3, 4, 5]
for a in item:
    if a % 2 == 0:
        print(a) # 2
                  4

item = [1, 2, 3, 4, 5]
for a in item:
    if a % 2 == 0:
        print(f'{a} is even')
    else:
        print(f'{a} is odd') # 1 is odd
                               2 is even
                               3 is odd
                               4 is even
                               5 is odd

item = [1, 2, 3, 4]
sum = 0
for a in item:
    sum = sum + a
print(f'sum of elements {sum}') # 'sum of elements 10'
len(item) # 4

```
for letter in 'Hello World':
    print(letter)          # H
                           e
                           l
                           l
                           o
                           W
                           o
                           r
                           l
                           d
```

(or)

```
st = 'Hello world'
for letter in st:
    print(letter)          # H
                           e
                           l
                           l
                           o
                           w
                           o
                           r
                           l
                           d
```

```
tup = (1,2,3)
for a in tup:
    print(a)               # 1
                           2
                           3

len(tup)                   # 3
```

```
a = [(1,2),(3,4),(5,6),(7,8)]
for b in a:
    print(b)               # (1,2)
                           (3,4)
                           (5,6)
                           (7,8)

len(a)                     # 4
```

```
a = [(1,2),(3,4),(5,6),(7,8)]
for (m,n) in a:
    print(m)
    print(n)               # 1
                           2
                           3
                           4
                           5
                           6
                           7
                           8

for (m,n) in a:
    print(m)               # 1
                           3
                           5
                           7

for (m,n) in a:
    print(n)               # 2
                           4
                           6
                           8
```

```
d = {'k1': 1, 'k2': 2, 'k3': 3}
for item in d.items():
    print(item)          # ('k1', 1)
                         # ('k2', 2)
                         # ('k3', 3)

for (key, value) in d.item():
    print(key)           # k1
                         # k2
                         # k3
    print(value)         # 1
                         # 2
                         # 3
```

While loops :-

↳ Loops will continue to execute a block of code while some conditions remains true.

Syntax:
```
while some_boolean_value:
    # do something
```

↳ You, We can add else part in while statement.

Syntax:
```
while some_boolean_value:
    # do_something
else:
    # do_something_else
```

↳
```
x=0
while x<5:
    print(x)
# 0
# 0
# .
# .
# .
# 0
# .
# .
# .
#infinite Loop
```

```
x=0
while x<5:
    print(x)
    x=x+1        # 0
                 # 1
                 # 2
                 # 3
                 # 4
#finite Loop
```

```
x=0
while x<3:
    print(x)
    x=x+1
else:
    print('wrong')   # 0
                     # 1
                     # 2
                     # 'wrong'
```

```
x=0
while x>3:
    print(x)
    x=x+1
else
    print('wrong')   # 'wrong'
```

↳ Jumping statements / keywords → break, continue, pass

we can use break, continue, pass in loop to add additional functionality for various cases.

break : To break out of current closest enclosing loop.
Continue : To run the loop from starting of iteration. Goes to the top of closest enclosing loop.
pass : Does nothing at all

↳
```
x=[1,2,3]
for item in x:
    pass
print('The end')        # 'The end'
```

↳
```
st = 'Sammy'
for letter in st:
    print(letter)       # S
                        # a
                        # m
                        # m
                        # y
```

```
st = 'Sammy'
for letter in st:
    if letter == 'a':
        continue
    print(letter)       # S
                        # m
                        # m
                        # y
```

```
st = 'Sammy'
for letter in st:
    if letter == 'a':
        break
    print(letter)       # S
```

```
x=0
while x<5:
    print(x)            # 0
    x = x+1             # 1
                        # 2
                        # 3
                        # 4
```

```
x=0
while x<5:
    if x==2:
        break
    print(x)            # 0
    x = x+1             # 1
```

Useful operators :-

↳
```
for num in range(10):
    print(num)
```
0 1 2 3 4 5 6 7 8 9

```
for num in range(3,10):
    print(num)
```
3 4 5 6 7 8 9

```
for num in range(0,10,2):
    print(num)
```
0 2 4 6 8

```
for num in range(0,11,2):
    print(num)
```
0 2 4 6 8 10

```
list(range(0,11,2))
```
[0,2,4,6,8,10]

```
range(1,10)
```
range(1, 10)

```
list(range(1,10))
```
[1,2,3,4,5,6,7,8,9]

```
tuple(range(1,10))
```
(1,2,3,4,5,6,7,8,9)

```
Set(range(1,10))
```
{1,2,3,4,5,6,7,8,9}

```
i=0
for c in 'DHEERAJ':
    print('At index {} character is {}'.format(i,c))
    i=i+1
```
At index 0 character is D
At index 1 character is H
At index 2 character is E
At index 3 character is E
At index 4 character is R
At index 5 character is A
At index 6 character is J

```
i=0
st='DHEERAJ'
for c in st:
    print(st[i])
    i=i+1
```
D H E E R A J

enumerate() is a function which returns a ~~list~~ tuple which has index for each character. displays each character with index in format of tuple

```
word = 'abcde'
for item in enumerate(word):
    print(item)
```

```
# (0, 'a')
  (1, 'b')
  (2, 'c')
  (3, 'd')
  (4, 'e')
```

```
word = 'abcde'
for i, c in enumerate(word):
    print(i)
    print(c)
    print('\n')
```

```
# 0
  a

  b
  c

  d
  e
```

```
word = 'abcde'
for i, c in enumerate(word):
    print(i)
```

```
# 0
  1
  2
  3
  4
```

```
word = 'abcde'
for i, c in enumerate(word):
    print(c)
```

```
# a
  b
  c
  d
  e
```

zip() is a function which returns tuples when any arguments are passed.
Combines each argument or objects from lists to form a tuple.

```
list1 = [1, 2, 3]
list2 = ['a', 'b', 'c']
for item in zip(list1, list2):
    print(item)
```

```
# (1, 'a')
  (2, 'b')
  (3, 'c')
```

```
list1 = [1, 2, 3]
list2 = ['a', 'b', 'c']
list3 = [100, 200, 300]
for item in zip(list1, list2, list3):
    print(item)                          # (1, 'a', 100)
                                         # (2, 'b', 200)
                                         # (3, 'c', 300)

a1=[1,2,3,4,5,6]
a2=['a','b','c']
a3=[100,200,300]
for item in zip(a1,a2,a3):               # (1, 'a', 100)
    print(item)                          # (2, 'b', 200)
                                         # (3, 'c', 300)

a1 = [1]
a2 =
a3 =
list(zip(a1,a2,a3))                      # [

list(zip(a1,a2,a3))  # [(1,'a',100),(2,'b',200),(3,'c',300)]
list(zip(a1,a2))     # [(1,'a'),(2,'b'),(3,'c')]

a1=[1,2,3]
a2=['a','b','c']
a3=[100,200,300]
for a,b,c in zip(a1,a2,a3):
    print(a)                             # 1
                                         # 2
                                         # 3
# print(f'{a},{b},{c}')                  # 1, a, 100
                                         # 2, b, 200
                                         # 3, c, 300

'x' in [1,2,3]                           # False
'x' in ['x','y','z']                     # True
2 in [1,2,3]                             # True
'a' in 'a world'                         # True
'mykey' in {'mykey':100}                 # True

d={'mykey':345}
345 in d.values()                        # True
'a' in d.keys()                          # True
```

max() it is a function that returns max.. value in a collection of num.
min() it is a function that returns min.. value in a collection of num.

```
mylist = [10, 20, 30, 40, 50]
max (mylist)                    # 50
min (mylist)                    # 10
```

#shuffle() is a function to shuffle all elements in a list
randint() is a function that returns an random integer from a list.
input() is a fuction that takes input from an user.

```
from random import shuffle
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
shuffle(a)                      #
a                               # [3, 9, 7, 8, 10, 5, 1, 2, 6, 4]
shuffle (a)
a                               # [4, 9, 2, 1, 10, 6, 5, 3, 7, 8]

random_list = shuffle(a)
random_list                     #
type(random_list)               # NoneType

from random import randint
randint (0, 100)                # 79
randint (0, 100)                # 99
my_num = randint (0, 10)
my_num                          # 3

result = input('Enter name:') # Enter name:
```

Dheeraj

```
result      # Dheeraj  # Enter name: Dheeraj

result = input ('Enter number') # Enter numbe:
```

30

```
result              # '30'
type (result)       # str
```

```
float(result)   # 30.0
int(result)     # 30

result = int(input('Enter numbers'))  # Enter Numbers 20
result          # 20
type(result)    # int
float(result)   # 20.0
type(result)    # float
```

List Comprehensions :-

↳ It is a unique way of quickly creating a list with python.

29|12|20

↳ We can write nested for loop in a single line to give output as list form

↳ We can write if ,if-else statements in a single line to give give output as list form.

↳
```
a = 'DHEERAJ'
b = []
for c in a:
    b.append(c)
b                  # ['D','H','E','E','R','A','J']

a = '12345'
b = []
for c in a:
    b.append(c)
b                  # ['1','2','3','4','5']

b = []
for c in '123':
    b.append(c)
b                  # ['1','2','3']

a = 'abcde'
b = [c for c in a]
b                  # ['a','b','c','d','e']
```

```
b=[c for c in 'CHANDAN']
b                          #['C','H','A','N','D','A','N']

b=[num for num in range(10)]
b                          #[0,1,2,3,4,5,6,7,8,9]

b=[num for num in range(1,10)]
b                          #[1,2,3,4,5,6,7,8,9]

b=[num*2 for num in range(1,5)]
b                          #[2,4,6,8]

b=[num**2 for num in range(1,5)]
b                          #[1,4,9,16]

b=[num for num in range(1,10) if num%2==0]
b                          #[2,4,6,8]

b=[num for num in range(1,10) if num%2==1]
b                          #[1,3,5,7,9]
```

Temperature to Fahrenheit (t→F)

$F = \frac{9}{5}t + 32$

```
t=[0,10,20,34.5]
f=[((9/5)*tp)+32 for tp in t]
f                          #[32.0, 50.0, 68.0, 94.1]

f=[]
for temp in t:
    f.append((temp*(9/5))+32)
f                          #[32.0, 50.0, 68.0, 94.1]
```

```
a = ['even' if x%2==0 else 'odd' for x in range(0,5)]
a  # ['even', 'odd', 'even', 'odd', 'even']

a=[f'{x}:even' if x%2==0 else f'{x}:odd' for x in range(0,5)]
a  # ['0:even', '1:odd', '2:even', '3:odd', '4:even']

a = [x if x%2==0 else f'{x}:odd' for x in range(0,5)]
a  # [0, '1:odd', 2, '3:odd', 4]


a=[]
for x in range(1,4):
    for y in range(1,4):
        a.append(x*y)
a                    # [1,2,3,2,4,6,3,6,9]

a=[x*y for x in [1,2,3] for y in [1,2,3]]
a                    # [1,2,3,2,4,6,3,6,9]

a=[x*y for x in range(1,4) for y in range(1,4)]
a                    # [1,2,3,2,4,6,3,6,9]
```

29/12/20

Methods & Functions

Methods and Python Document / Functions :-

append() is a function to add one more element to list.
pop() is a function to remove last element from list.

↳
```
a = [1, 2, 3]
a.append(4)
a                    # [1, 2, 3, 4]
a = [1, 2, 3, 4, 5]
a.pop()              # 5
a                    # [1, 2, 3, 4]
```

help(a.insert) #

Python Documentation : https://docs.python.org/3/

Functions in Python :-

↳ Functions allow us to create blocks of code that can be easily executed many times.

```
def name_of_function():
    """explains functions"""
    print('Hello')
name_of_function()
```

```
def name_of_function(n):
    """explains functio"""
    print("Hello" + n)
name_of_function('Dheeraj')
```

↳ return keyword allows to send back the result of the function. return allows us to assign the output of fuction to a new variable.

```
def add(a1, a2):
    return a1+a2
result = add(1, 2)    # 3
```

↳
```
def a():
    """Hi
    Dheeraj"""
    print("Name")
a()                   # 'Name'
```

```
help(a)  #  Hi
            Dheeraj

def name(s):
    print("Dheeraj "+s)
name('chandan')          # 'Dheeraj Chandan'

def name(s):
    print(f'{s} is a new name')
t=input('enter name')
name(t)                  # enter name
                         Dheeraj
                         Dheeraj is a new name

def a():
    s=input('Enter the name:')
    name = "Dheeraj "+s
    print(name)
a()                      # Enter the name: Chandan
                         Dheeraj Chandan

def a(num):
    print(f'{num} is a number')
t=input('Enter a number')
a(t)                     # Enter a number: 7
                         7 is a number

def a():
    print('Hello')
a()                      # Hello

def a(name)
    print('Hello' +name)
a('Dheeraj')             # Hello Dheeraj

def a(n):
    print(f'{n} Hi!!')
a('Dheeraj')             # Dheeraj Hi!!
```

```
def a(n):
    print(f'Entered number is {t}')
t = int(input('Enter the number'))
a(t)
```
Enter the numbe 5
Entered number is 5

```
def a(t='Dheeraj'):
    print(t + 'Hello')
a()
```
Dheeraj Hello

→
```
def a(n):
    print('hi' + n)
b = a('Dheeraj')
b
type(b)
```
hi Dheeraj
#
NoneType

```
def a(n):
    return 'hi' + n
a('Dheeraj')
```
'hi Dheeraj'

```
def a(n):
    print('hi' + n)
a('Dheeraj')
```
hi Dheeraj

→
```
def a(n):
    return 'hi' + n
b = a('Dheerj')
b
type(b)
```
'hi Dheeraj'
str

```
def a(n):
    print('hi' + n)
b = a('Dheeraj')
b
type(b)
```
#
NoneType

```
def s(a,b):
    t=a+b
    return t
n=s(10,20)
n                    # 30

def s(a,b):
    t= a+b
    return t
s(10,20)             # 30
type (t)             # int
```

Check whether 'dog' word presen in a string.

```
def a(s):
    if 'dog' in s:
        return True
    else:
        return False
a('dog is a pet animal')   # True

def a(s):
    if 'dog' in s:
        return True
    else
        return False
a('cat is a pet animal')   # False
```

upper() is a function to change characters in a string from lower to upper case.

lower() is a function to change characters in a string from upper to lower case.

```
def a(s):
    if 'DOG' in s.upper():
        return True
    else:
        return False
a('dog is a pet animal') # True

def a(s):
    if 'who' in s.lower():
        return True
    else:
        return False
a('WHO declared covid') # True
```

PIG_LATIN

↳ If a word starts with vowel then add 'ay' at end
↳ If a word does not start with a vowel then put first letter at the end, then add 'ay'

ex:- word → ordway
apple → appleay

↳
```
def pig_latin(word):
    first_letter = word[0]
    if first_letter in 'aeiou':
        pigword = word + 'ay'
    else:
        pigword = word[1:] + first_letter + 'ay'
    return pig_word
pig latin('word')    # 'ordway'
pig latin('apple')   # 'appleay'
```

*args & **kwargs :-

↳ *args used to take any number of argument we can take as many number of arguments. It gives tuple.

↳ **kwargs used to give dictionary. By using this by comparing keys we can find or print values or take values.

↳
```
def myfunc(a,b)
    return sum((a,b))*0.05
myfunc(40,60)                    # 5.0

def myfunc(a,b,c=0,d=0,e=0)
    return sum((a,b,c,d,e))*0.05
myfunc(40,60,100)                # 10.0

def myfunc(a,b,c=0,d=0,e=0)
    return sum((a,b,c,d,e))*0.05
myfunc(40,60,100,100)            # 15.0

def myfunc(a,b,c=0,d=0,e=0)
    return sum((a,b,c,d,e))
myfunc(40,60,100,100,100,3)      # error limit of arguments crossed.
```

↳
```
def myfunc(*args):
    return sum(args)*0.05
myfunc(40,50)                    # 5.0
myfunc(40,50,100)                # 10.0
myfunc(40,50,100,100)            # 15.0
myfunc(40,50,100,100,100)        # 20.0
myfunc(40,50,100,100,100,50)     # 22.5
```

#In *args '*' is indicates multiple arguments
'args' is a keyword, we change with other name like ~~kwargs~~ '*spam' etc..

```
def f(*args):
    print(args)
f(40,60,100,1,34)  # (40,60,100,1,34)
                   # ↳ tuple

def f(*spam):
    print(spam)
f(40, 60, 100, 1, 34)  # (40,60,100,1,34)

def f(*args):
    for i in args:
        print(i)
f(1,2,3,4,5)  # 1
              # 2
              # 3
              # 4
              # 5

→ def f(**kwargs):
    if 'fruit' in kwargs:
        print('My fruit of choise is {}'.format(kwargs['fruit']))
    else:
        print('Fruit not found')
f(fruit='1')  # My fruit of choice is 1

# **kwargs takes dictionaries.

def f(**kwargs):
    if 'fruit' in kwargs:
        print('Choise {}'.format(kwargs['fruit']))
    else
        print('not found')
f(fruit='1', animal='2')  # choise 2
```

```
def f(**jelly):
    print(jelly)
f('a'=1, 'b'=2, 'c'=3)  # {'a'=1, 'b'=2, 'c'=3}
f(a=1, b=2, c=3)        # {'a'=1, 'b'=2, 'c'=3}
```

```
def f(*args, **kwargs):
    print(args)
    print(kwargs)
    print('I would like {} {}'.format(args[0], kwargs['food']))
f(10, 20, 30, fruit='orange', food='eggs', animal='dog')
# (10, 20, 30)
{fruit='orange', food='eggs', animal='dog'}
I would like 10 eggs.
```

6/1/20

→ max() is a function to find larger number or chosrt or
min() is a function to find smaller number or chorater

```
a=(1, 2, 3)
b=[1, 2, 3]
c={1, 2, 3}
d={'a', 'e', 'c'}
max(a)   # 3
min(a)   # 1
max(b)   # 3
min(b)   # 1
max(c)   # 3
min(c)   # 1
max(d)   # e
min(d)   # a
```

```
sum(a) # 6
sum(b) # 6
sum(c) # 6

max(2,3) # 3
min(4, 1) # 1
```

sum() is a function to find total.

Lambda Expressions, Map and Filter Function :-

map() is a ~~function~~ keyword which connects each element with a function.
It makes an iterator that computes the function using arguments from each of the iterables. Takes all objects in a lists & allows to apply a function.

↳
```
def square(num)
    return num**2
my_num = [1,2,3,4,5]
for item in map(square, my_nums):
    print(item)                    # 1
                                   # 4
                                   # 9
                                   # 16
                                   # 25
list(map(square, my_nums))  # [1,4,9,16,25]
```

↳
```
def splicer(mystring):
    if len(mystring)%2 == 0:
        return 'EVEN'
    else:
        return mystring[0]
names = ['Andy', 'Eve', 'Sally']
list(map(splicer, names))   # ['EVEN', 'E', 'S']
```

17|2|21

filter() is a keyword which based on function condition. It filters the input given to a function & gives output from it by the given condition.

It filters to get output based on function condition.
It takes all objects in a lists and runs that though a function to create a new list with all objects that return true in that function

```
def a(n):
    if n%2==0:
        return n
num = [1,2,3,4,5,6]
list(filter(a, num))  # [2,4,6]
```

```
def a(n)
    if n%2==0:
        return n
num=[1,2,3,4,5,6]
for item in filter(a, num):
    print(item)          # 2
                         # 4
                         # 6
```

```
def a(n):
    if n%2==0
        return n
num=[1,2,3,4,5,6]
list(map(a, num))  # [None, 2, None, 4, None, 6]
```

```
def a(n):
    if n%2==0:
        retun n
num=[1,2,3,4,5,6]
for item in list(map(a, num)):
    print(item)          # None
                         # 2
                         # None
                         # 4
                         # None
                         # 6
```

```
def a(n):
    if n%2==0:
        return n
    else:
        return 'ODD'
num = [1,2,3,4,5,6]
list(filter(a, num))  # [1,2,3,4,5,6]
```

↳ def a(n):
    if n%2==0:
        return n
    else: return 'ODD'
num=[1,2,3,4,5,6]
list(map(a,num)) # ['ODD',2,'ODD',4,'ODD',6]

lambda

↳ It is a keyword which is to express a function in a single line.
We can execute a code in a single line.

↳ We use lambda functions when we require a nameless functions for a short period of time. Use use it as an argument to a higher order function. (a function that takes in other functions as arguments. lambda functions are used along with build-in functions like filter(), map() etc..

↳ nums=[1,2,3,4,5]
list(map(lambda n:n%2==0, nums))
# [False, True, False, True, False]

↳ sq= lambda a:a*2
sq(5) # 10

↳ ~~for item in nums:~~

↳ nums=[1,2,3,4,5]
for item in nums:
    map(lambda n:n*2, item)

```
nums = [1, 2, 3, 4, 5]
list(map(lambda n: n%2==0, nums))
                    # [False, True, False, True, False]

sq = lambda a: a*2
sq(5)               # 10

sq = lambda a: a*2
for item in nums:
    print(sq(item))  #  2
                        4
                        6
                        8
                        10

num = [1,2,3,4,5,6]
list(map(lambda n: n*2, num))
                    # [2,4,6,8,10,12]

a = lambda n: n*2
a(4)                # 8

num = lambda n: n**2
num(4)              # 16

a = ['Dheeraj', 'Chandan', 'Ram']
list(map(lambda n: n[0], a)) # ['D', 'C', 'R']

a = [1,2,3,4,5,6]
list(map(lambda n: n%2==0, a))
                    # [Fasle, True, False, True, False, True]

a = [1,2,3,4,5,6]
list(filter(lambda n: n%2==0, a)) # [2,4,6]

n = lambda n: n**2
n(2)                # 4
```

→ n = lambda a: a+2
n(3) # 5

→ a = [1,2,3,4,5]
list(map(lambda n: n*2, a)) # [2,4,6,8,10]

a = [1,2,3,4,5]
list(map(lambda n: n+2, a)) # [3,4,5,6,7]

→ a = [1,2,3,4,5]
list(filter(lambda n: n+2, a)) # [1,2,3,4,5]

→ a = [1,2,3,4,5]
n = lambda n: n+2
for item in a:
    b = n(item)
    print(b)
# 3
4
5
6
7

→ a = [1,2,3,4,5]
list(filter(lambda n: n*2, a)) # [1,2,3,4,5]

a = [1,2,3,4,5]
n = lambda n: n*2
for item in a:
    b = n(item)
    print(b)
# 2
4
6
8
10

→ nums = [1,2,3,4,5]
list(map(lambda n: n%2==0, nums))
# [False, True, False, True, False]

nums = [1,2,3,4,5]
list(filter(lambda n: n%2==0, nums))
# [2,4]

→ names = ['Dheeru', 'Chand', 'Ram']
list(map(lambda x:x[::-1], names))
# ['ureehD', 'dnahC', 'maR']

19/2/21

Nested statements

Difference in map and filter :-

↳ Map takes all objects in a list & allows us to apply a function to it.

↳ Filters takes all objects in a list and runs that through a function to create a new list with all objects that returns True in that function.

Nested Statements and Scope :-

↳
```
x= 10
def a():
    x=50
    return x
x                  # 10
print(x)           # 10
print(a())         # 50

x=[1,2,3,4,5]
def a():
    x=[1,2,3,4,5]
    return x*2
Print(x)           # [1,2,3,4,5]
print(a())         # [1,2,3,4,5,1,2,3,4,5]
```

```
x=[1,2,3,4,5]
def a():
    x=[1,2,3,4,5]
    c = list(map(lambda n:n*2, x))
    return c
print(x)                    # [1, 2, 3, 4, 5]
print(a())                  # [2, 4, 6, 8, 10]
```

```
x=[1,2,3,4,5]
def a():
    x=[1,2,3,4,5]
    c = list(map(lambda a:a*2, x))
    return '\n{}\n{}'.format(x, c)
print(x)                    # [1, 2, 3, 4, 5]
print(a())                  # [1,2,3,4,5]
                            # [2,4,6,8,10]
```

```
x=[1,2,3,4,5]
def a():
    x=10
    return x
print(x)                    # [1, 2, 3, 4, 5]
print(type(x))              # <class 'list'>
print(a())                  # 10
print(type(a()))            # <class 'int'
```

```
x=[1,2,3,4,5]
def a():
    x=10
    return x
type(x)                     # list
x=[1,2,3,4,5]
def a():
    x=10
    return x
type(a())                   # int
```

LEGB rule

↳ It is a kind of name lookup procedure, which determines the order in which Python looks up names.

↳ If you reference a given name, then Python will look that name up sequentially in the local, enclosing, global and build-in scope. If the name exists then you'll get the 1st occurance of it.

↳ LEGB rule :

(i) L - Local → Names assigned in any way within a function (def or lambda), & not declared global in that function.

(ii) E - Enclosing function locals → Names in the local scope of any and all enclosing functions (def or lambda), from inner to outer.

(iii) G - Global (module) → Names assigned at the top-level of a module file, or declared global in a def within the file.

(iv) B - Built-in (Python) → Names preassigned in the built in names module : open, range, SyntaxError, ---

↳
```
x=10
def a():
    x=50
    def b():
        print(x)          # 50
    b()
a()

x=10
def a(n):
    print(x)              # 10
    def b()
        return n
    b()
a(x)

a = lambda n:n*2          # 6
a(3)                      # n is a local Variable
```

→
```
name='This is a Global Variable'
def a():
    name='Dheeraj'
    def b():
        print('Name is '+name)
    b()
a()                       #'Name is Dheeraj
```

```
# G-Global
name='Dheeraj'
def a():
    # E-Enclosing
    name='Chandan'
    def b():
        # L-Local
        name='Balivada'
        print('Name is '+name)
    b()
a()                       # Name is Balivada
```

```
# G-Global
name = 'Dheeraj'
def a():
    # E-Enclosing
    name = 'Chandan
    def b():
        # L-Local
        # name = 'Balivada'
        print ('Name is'+name)  # Name is Chandan
    b()
a()
```

```
#G-Global
name= 'Dheeraj'
def a():
    # E-Enclosing
    # name = 'Chandan'
    def b():
        # L-Local
        # name = 'Balivada'
        print ('Name is'+name)
    b()                          # Name is Dheeraj
a()
```

```
#G-Global
# name = 'Dheeraj'
def a():
    # E- Enclosing
    name= 'Chandan'
    def b():
        # L-Local
        name = 'Balivada'
        print ('Name is '+ name)
    b()
a()                              # Name is Balivada
```

```
# G-Global
# name = 'Dheeraj'
def a():
    # E-Enclosing
    name = 'Chandan'
    def b():
        # L-Local
        # name = 'Balivada'
        print('Name is ' + name)
    b()                          # Name is Chandan
a()

x=10                             # Global
def a(x):
    print(x)        # 10         # Executing global value
    x=20                         # Local Reassignment
    print(x)        # 20         # Excecuting Local value
a(x)
print(x)            # 10         # Executing Global value

x=10                             # Global
def a():
    global x                     # Value of x continously effects/change if reassing to 'x'
    print(x)  # 10               # Executing global value
    x=20                         # local reassignment
    print(x)  # 20
    x='HI DHEE RAJ'              # executing local value
a()
print(x)    # HI DHEERAJ         # executing changed globally
```

```
x=10                          # Global
def a(x):
    print(x)  #10             # Executing Global Value
    x=20                      # Local Reassignment
    print(x)  #20
    x='Hi Dheeraj'            # Executing Local value
    print(x)  # Hi Dheeraj
a(x)
print(x)   # 10               # Executing global value
```

```
x=10                                  # Global
def a(x):
    print(x)   #  10                  # Executing global value
    x=20                              # Local reassignment
    print(x)   #  20
    x='Hi Dheeraj'                    # Executing local value
    print(x)   #  Hi Dheeraj
    return x
b=a(x)
print(x)   #  10                       # Executing global value
b          # 'Hi Dheeraj'              # Returning the value of
                                       last assigned value of
                                       x.
```

20/2/21

Object Oriented Programming (OOPs)

Object Oriented Programming (OOP) :-

↳ It allows programmers to create their own objects that have methods & attributes.

We recall that after defining a string, list, dictionary or other objects, you were able to call methods off of them with the .method_name() syntax.

→ These methods acts as functions that use information about the object as well as the object itself to return results or change the current object.

ex: This includes appending to a list (or) counting the occurance of an element in a ~~list~~ tuple.

⊖ OOP allows users to create their own objects.

⊖ OOP allows use to create code that is repeatable and organised.

→ For much larger scripts of Python code, repeated tasks and objects can be defined with OOP to create code which is usable.

Syntax :

```
class NameOfClass():
    def __init__(self, param1, param2):
        self.param1 = param1
        self.param2 = param2
    def some_method(self)
        ~~print~~
        # perform an action
        print(self.param1)
```

where, ~~init~~ →

__init__ → To create instance of actual objects

21/2/21 Class is a keyword to user defined object to create instances of an object

instances is a particular (or specific) object that created from a particular class.

class names should begin with capital letters.

→ To create a class with classname 'Sample'

```
class Sample():
    pass
```

OOP - 1 (Attributes & class keyword):

pass is a keyword to create an class i.e. Sample() but don't do anything.

→

```
mylist=[1,2,3,4,5]
myset = set()
type(mylist)                    # list
type(myset)                     # set
myset.add(3)
myset                           # {3}
myset.add(4)                    # {3,4}
myset
myset.remove(3)
myset                           # {4}
myset.remove(4)
myset                           # set()
mylist                          # [1,2,3,4,5]
```

→
```
class Sample():
    pass
my_sample = Sample()
type(my_sample)          # __main__.Sample
```

[here code has no attributes]

→
```
class Sample():
    # Attribute
    # We take in argument
    # Assign it using self.attribute_name
    def __init__(self, name):
        self.name = name
a = sample(name = "Dheeraj")
type(a)                  # __main__.Sample
a.name                   # 'Dheeraj'
a                        # <__main__.Sample at 0x...>
```

Here self is a keyword which connects this method to instance of the class. and Refer to itself.

__init__ it is refered as a method inside a class called upon when creating instances of class & acts as constructor

→
```
class Sample():
    def __init__(self, name, roll, section):
        self.name = name
        self.roll = roll
        self.section = section
a = Sample(name='Dheeraj', roll=21, section = 4.0)
type(a)                  # __main__.Sample
```

```
a.name                  # 'Dheeraj'
a.roll                  # 21
a.section               # 4.0
type(a.name)            # str
type(a.roll)            # int
type(a.section)         # float
```

OOP-2 (Class Object, Attributes & methods) :-

↳ Class object Attribute not related with self keyword.
For this we don't use class object Attribute.

```
class Sample():
    # Class Object Attributes
    # Same for any instances of a class
    branch = 'EEE'
    def __init__(self, name, roll, section):
        self.name = name
        self.roll = roll
        self.section = section
a = Sample(name='Dheeraj', roll=21, section=4.0)
a.branch                    # 'EEE'
```

↳ class Sample():
#

Difference b/w Attribute & a Method →

↳ A variable stored in an instance or class is called an attribute.

↳ A function stored in an instance or class is called a method.

```
class Sample():
    # class Object Attributes
    # Same for any instances of a class
    branch = 'EEE'
    def __init__(self, name, roll, section):
        self.name = name
        self.roll = roll
        self.section = section
    def test(self):
        print('Student')
a = Sample(name = 'Dheeraj', roll = 21, section = 4.0)
print(type(a))
print(a.branch)
print(a.name)
print(a.roll)
print(a.section)
a.test()
```

```
<class '__class__.Sample'>
EEE
Dheeraj
21
4.0
Student
```

Classes, instances, types, methods & attributes

Class → A blueprint of an instance

Instance → A constructed object of an class

Type → Indicates the class the instance belongs to

Attribute → Any object value: object.attribute

Method → a callable attribute defined in the class.

→

```
class Sample():
    # Class Object Attributes
    # Same for any instances of a class
    branch = 'EEE'
    def __init__(self, name, roll, section):
        self.name = name
        self.roll = roll
        self.section = section
    def test(self):
        print('student name is {}'.format(self.name))
a = Sample('Dheeraj', 21, 4.0)
print(type(a))
print(a.branch)
print(a.name)
print(a.roll)
print(a.section)
a.test()
```

```
# <class '__main__.Sample'>
EEE
Dheeraj
21
4.0
student name is Dheeraj
```

→ Class Sample():

```
    #Class Object Attributes
    #Same for any instances of a class
    branch = 'EEE'
    def __init__(self, name, roll, section):
        self.name=name
        self.roll = roll
        self.section =section
    def test(self):
        print('student name is {}. His roll {}
              and section {}'.format(self.name,
               self.roll, self.section))
a=Sample("Dheeraj", 21, 4.0)
print(type(a))
print(a.branch)
print(a.name)
print(a.roll)
print(a.section)
a.test()
```

```
# <class '__main__.Sample'>
EEE
Dheeraj
21
4.0
Student name is Dheeraj. His
roll 21 and section 4.0
```

→
```
class Circle():
    # Class Object Attributes
    pi = 3.14
    def __init__(self, radius=1): # If we assign here
                                  # then we can reassign
                                  # all time if we
                                  # want
        self.radus = radius
    def circumf(self):
        return 2*self.pi*self.radius
my_circle = Circle()
print(my_circle.radius)
print(my_circle.pi)
print(my_circle.circumf()) # 1
                           # 3.14
                           # 6.28
```

→
```
class Circle():
    # class Object Attribute
    pi = 3.14
    def __init__(self, radius = 1): # If we assign here
                                    # then we can reassign
                                    # any time if we want
        self.radius = radius
    def circumf(self):
        return 2*self.pi * self.radius
my_circle = Circle(30)   # Here we can give value
                         # for radius
print(my_circle.radius)
print(my_circle.pi)
print(my_circle.circumf())  # 30
                            # 3.14
                            # 188.4
```

→ Class Circle():

```
    # Class Object Attribute
    pi = 3.14
    def __init__(self, radius=1): # If we assign here then we can reassign all time if we want
        self.radius = radius
        self.area = ~~area~~ radius**2 * self.pi
        ~~def~~
    def circumf(self):
        return 2*self.pi*self.radius
my_circle = Circle(30)  # here we can give values for radius
print(my_circle.radius)
print(my_circle.pi)
print(my_circle.circumf)
print(my_circle.area) #  30
                         3.14
                         188.4
                         2826.0
```

→ class Circle():

```
    # class Object Attributes
    pi=3.14
    def __init__(self, radius=1): # If we assign here then we can reassign if we want
        self.radius = radius
    def circumf(self):
        return 2* self.pi * self.radius # we can write Circle.pi as pi is an class object Attribute
    def area(self):
        return self.pi * self.radius **2
my_circle = Circle(30)
print (my_circle.radius
print (my_circle.pi
print (my_circle.circumf())
print (my_circle.area())    # 30
                            3.14
                            188.4
                            2826.0
```

If we assign radius=1 in __init__ with self key word then we can give or reassing any value for radius.

We can write Circle.pi instead of self.pi as pi is a class object attribute.

→ class Circle():

```
    # class Object Attribute
    pi = 3.14
    def __init__(self, radius=1):
        self.radius = radius
    def circumf(self):
        return 2 * Circle.pi * self.radius
    def area(self):
        return Circle.pi * self.radius **2
my_Circle = Circle(30)
print(my_circle.radius)
print(my_circle.pi)
print(my_circle.circumf())
print(my_circle.area())
```

30
3.14
188.4
2826.0

OOP - 3 (Inheritance & Polymorphism)

→ Inheritance allows us to define a class that inherits all the methods and properties from another class.

2 types : (i) Parent class
(ii) Child class

(i) Parent class being inherited from, Also called base class

Creating - Any class can be a parent class.

(ii) Child class that inherits from another class, also called derived class.

Creating - To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class.

example :

(i) class Person():

→

```
class Name():
    def __init__(self):
        print("Dheeraj Chandan")
myname = Name()                    # Dheeraj Chandan
```

```
class name():
    def __init__(self):
        print("Dheeraj Chandan")
    def roll(self, roll):
        print(roll)
    def section(self):
        print("Section EEE-4")
my_name = name()
my_name.section()
my_name.roll(10)                   # Dheeraj Chandan
                                   Section EEE-4
                                   10
```

```
class name():
    def __init__(self, a=1):
        self.a = a
    def roll(self, roll):
        return roll
    def section(self):
        return "Section EEE-4"
my_name = name()
print(my_name.section())
print(my_name.roll(10))            # Section EEE-4
                                   10
```

→ class Sub():
    def __init__(self):
        print("Marks")
    def math(self, MM):
        print(MM)
    def phy(self, PM):
        print(PM)
    def che(self, CM):
        print(CM)
SubM = Sub()
SubM.math(10)
SubM.phy(20)
SubM.che(30)

we can write without __init methode

Marks
10
20
30

class Sub():
    def __init__(self):
        self.SubM = SubM()
    def math(self, MM):
        return MM
    def phy(self, PM):
        return PM
    def che(self, CM):
        return CM
SubM.math(10)
SubM.phy(20)
SubM.che(30)

We can write pass key word to do nothing for removing an error inside __init__

10
20
30

Ex:- def __init__(self):
        pass

class aC():
    def b(self):
        print("Hi")
x = aC()
x.b() # Hi

class aC():
    def __init__(self):
        pass
    def b(self):
        print("Hi")
x = aC()
x.b() # Hi

Inheritance → The child class inherits the methods from the parent class. However, it is possible to modify a method in a child class that it has inherited from the parent class.

→
```
class Animal():
    def __init__(self):
        print("Animal created")
    def who_am_i(self):
        print("I am an animal")
    def eat(self):
        print("I am eating")
myanimal = Animal()          # Animal Created
myanimal.who_am_i()          # I am an animal
myanimal.eat()               # I am eating
```

If Dog class needs same methods of Animal class then

```
class Dog(Animal):  # Base class Animal
    def __init__(self):
        Animal.__init__(self)
        print("Dog Created")
mydog = Dog()                # Animal Created
                             # Dog Created
mydog.eat()                  # I am eating
mydog.who_am_i()             # I am an animal
```

```
class Dog(Animal): # base class Animal
    def __init__(self):
        Animal.__init__(self)
        print("Dog Created")
    def who_am_i(self):
        print("I am a Dog")
mydog = Dog()                    # Animal created
                                 Dog created
mydog.who_am_i()                 # I am a Dog
mydog.eat()                      # I am eating
```

```
Class Dog(Animal): # Base class Animal
    def __init__(self):
        Animal.__init__(self)
        print("Dog Created")
    def who_am_i(self):
        print("I am a Dog and eating")
    def bark(self):
        print("Bhowwwh!!")
mydog = Dog()                    # Animal Created
                                 Dog Created
mydog.eat()                      # I am eating
mydog.who_am_i()                 # I am a dog and eating
mydog.bark()                     # Bhowwwh!!
```

Polymorphism → It defines the methods in the child class that has the same name as the methods in the parent class.

We can redefine certain methods and attributes specifically to fit the child class, which is known as Method overiding.

Polymorphism allows us to access these overridden methods and attributes that have the same name as the parent class.

```
class Dog()
    def __init__(self, name):
        self.name = name
    def speak(self):
        return self.name + " speaks Bhowwwh!!"
a = Dog("Tommy")
a.speak()      # Tommy speaks Bhowwwh!!

class Cat()
    def __init__(self, name):
        self.name = name
    def speak(self):
        return self.name + " speaks meewu"
b = Cat("Catty")
b.speak()      # Catty speaks meewu
```

Note: Here Dog class and Cat class has same methods having same name.

```
for pet in [a, b]:
    print(type(pet))
    print(pet.speak())
    print(type(pet.speak()))
    print("\n")
```

```
# <class '__main__.Dog'>
Tommy speaks Bhowwwh!!
<class 'str'>

<class '__main__.Cat'>
Catty speaks Meewu
<class 'str'>
```

```
def petspeak(pet):
    print(pet.speak())
petspeak(a)
petspeak(b)          # Tommy speaks Bhowwwh!!
                       Catty speaks meewu

petspeak(a)          # Tommy speaks Bhowwwh!!
petspeak(b)          # Catty speaks meewu
```

Abstract Class → It can be considered as a blue print for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class. A class which contains one or more abstract methods is called an abstract class

```
class Animal():
    def __init__(self, name):
        self.name = name
    def speak(self):
        raise error("Subclass must implement the abstract
                     class")

class Dog(Animal):
    def speak(self):
        return self.name + " speaks Bhowwwh!!"
class Cat(Animal):
    def speak(self):
        return self.name + " speaks meewu"
a = Dog('Tommy')
b = Cat('Catty')
a.speak()          # 'Tommy speaks bhowwwh!!"
b.speak()          # 'Catty speaks meewu!!"
```

```
class Dog(Animal):
    def speak(self):
        return self.name + "speaks bhowwwoh!!"
a=Dog('Tommy')
a.speak()          # 'Tommy speaks bhowwwoh!!'
```

```
class Cat(Animal):
    def speak(self):
        return self.name + " speaks meeww"
b=Cat('catty')
b.speak()          # 'catty speaks meeww'
```

17/3/21 17/3/21

OOP-4 (Special (magic dunder) methods

↳
```
mylist=[1,2,3,4,5]
len(mylist)        # 5
```

```
class Sample():
    pass
mysample=Sample()
mysample           # <__main__.Sample at 0x213e5aeea8>
# len(mysample)    # TypeError: 'Sample' has no len()
print(mylist)      # [1,2,3,4,5]
```

↳
```
class Book():
    def __init__(self,title,author,pages):
        self.title=title
        self.author=author
        self.pages=pages
    def __str__(self):
        return '{} by {},{} pages'.format(self.title,
                                          self.author,
                                          self.pages)
    def __len__(self):
        return self.pages
```

```
b = Book('Python', 'Jose', 1000)
b                                  # <__main__.Book at 0x20d2ccc5f08>
```

→
```
class Book():
    def __init__(self, title, author, pages):
        self.title = title
        self.author = author
        self.pages = pages
    def __str__(self):
        return '{} by {}, {} pages'.format(self.title,
                                           self.author,
                                           self.pages)
    def __len__(self):
        return self.pages
b = Book('Python', 'Jose', 1000)
b                                  # <__main__.Book at 0x20d2ccc5f08>
print(b)      # Python by Jose, 1000 pages
b.pages       # 1000
b.title       # 'Python'
b.author      # 'Jose'
str(b)        # 'Python by Jose, 1000 pages'
len(b)        # 1000
# To delete the book 'del' keyword is used
del(b)
b                                  # 'b' is not defined
```

→
```
class Book():
    def __init__(self, title, author, pages):
        self.title = title
        self.author = author
        self.pages = pages
    def __str__(self):
        return '{} by {}, {} pages'.format(self.title, self.author,
                                           self.pages)
    def __len__(self):
        return self.pages
    def __del__(self):
        print("Book deleted")
b = Book('Python', 'Jose', 1000)
b                                  # Python by Jose, 1000 pages
```

```
str(b)      # 'Python by Jose, 1000 pages'
len(b)      # 1000
del(b)      # 'Book deleted
# print(b)      #'b' is not defined"
```

↳ __str__(self) is a special method used to get called by builtin str() method to return a string representation of a type.

__len__(self) is a special method used to return the length of the object. len() used to sequences i.e. string, tuples & collections i.e dicts, lists

__del__(self) is a special method used to destruct the method and delete the method.

__init__(self) is a special method used to get called by builtin int() method to convert a type to an int.

17/3/21

19. Errors and Exceptions

↳
```
a = 10
b = 20
def add(c,d):
    print(a+b)  # or print(c+d)
add(a,b)        # 30
```

```
a = 10
b = input('Enter number')
def add(c,d):
    print(a+b) #or print(c+d)
add(a,b)  # TypeError as 'b' is str format, output is an error.
```

```
a = 10
b = input('Enter number')
def add(c,d)
    print(c+d)
#add(a,b)  # Enter number 10
```

↳ try : It is a keyword for the block of code to be attempted (may lead to an error).

except : It is a keyword for the block of code that will execute in a case there is an error in try block.

finally : It is a keyword for the final block of code to be executed regardless of an error.

```
try:
    a=10
    b=20
    c=a+b
    print(c)
except:
    print("Error in a code")
else: print("No error in a code")  # 30
```

No error in a code

```
try:
    a= 10
    b='20'
    c=a+b
    print(c)
except:
    print("Error in a code")
else:print("No error in a code")  # error in a code
```

```
try:
    a= 10+'10'
except:
    print('Error in a code')    # Error in a code
```

```
try:
    a=10+'10'
except:
    print('Error in a code')
else:
    print(a)                    # Error in a code
```

```
try:
    a=10+10
except:
    print('Error in a code')
else: print(a)                  # 20
```

```
try:
    f = open('Writefile', 'w')
    f.write("Hi")
except Writefile:
    print("Type error")
except:
    print("other Exceptions")
finally:
    print("Code running")    # Code running


try:
    f = open('Writefile', 'e')
    f.write("Hi")
except TypeError:
    print("Type Error")
except:
    print("other exceptions")
finally:
    print("Code running")    # other exceptions
                             Code running


try:
    f = open('Writefile', 'e'
    f.write("Hi")
except:
    print("other exceptions")
finally:
    print("Code running")    # other exceptions
                             Code running

def ask_for_int():
    try:
        result = int(input("number: "))
    except:
        print("error occoured")
    finally:
        print("End of try/except/finally")
```

```
ask_for_int()   # number: 10
                End of try/except/finally
ask_for_int()   # number: a
                error occoured
                End of try/except/finally
```

```
def ask_for_int():
    while True:
        try:
            result = int(input("number: "))
        except:
            print("error occoured")
            continue
        else:
            print("Thank you")
            break
        finally:
            print("end of try/except/finally")
```

```
ask_for_int()   # number: 10
                Thank you
                End of try/except/finally
ask_for_int()   # number: a
                error occoured
                End of try/except/finally
                number: b
                error occoured
                End of try/except/finally
                number: 10
                Thank you
                End of try/except/finally
```

```
try:
    f = open("Writefile", "w")
    f.write("Hi")
except TypeError:
    print("Type error")
except OSError:
    print("Other Exception")
finally:
    print("code running")    # code running
```

22/8/21

2. Pylint Overview : Unit testing (Part-1)

↳ Pylint (unit testing)

There are several testing tools :

(i) pylint — A library that looks at your code and reports back possible issues.

(ii) unittests — This is a builtin library which will allow to test your own program and check you are getting desired output.

Python as a set of style conventions rule known as 'PEP-8'. For this we are creating /we will creating .py scripts in subline text editors. We can still use jupyter notebook for code using the %%writefile magic command.

↳ Steps & execution :

step 1 — Open Subline Text Editor and save file with .py extension on desktop (say : simple.py)

Step 2 — Write the code in simple.py file

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

(or)

```
def myfunc():
    first=1
    second=2
    print(first)
    print(second)
myfunc()
```

Step 3 — Open terminal from jupyter Notebook

command in terminal :

```
pip install pylint
pylint simple.py
```

finally Output will show :

no. of statements analysed
statistics by type
raw matrics
Duplications
messages by category
messages
Global evaluation : Your cod has been rated at .../10

3. Running with Unittest Libraries : Unit Testing Part 2:

↳ Steps & Execution :

Step 1 — Open Sublime Text Editor and create two new files and save it as U1.py and U2.py on desktop.

step 2 — Write code in U1.py :

```
#'''
def a(text):
    return text.capitalize()
#'''
```

Step 3 — Write code in U2.py :

```
#'''
import unittest
import U1
class TestCap(unittest.TestCase):
    def uscript1(self):
        text='dheeraj'
        res = U1.a(text)
        self.assertEquals(res,'Dheeraj')
    def uscript2(self):
        text = 'chandan'
        res = U1.a(text)
        self.assertEquals(res,'Chandan')
if __name__=='__main__':
    unittest.main()
#'''
```

Step 4 — Open terminal from jupyter notebook

step 5 — Run command U1.py

step 6 — Run command U2.py

22/08/21

Modules and Packages

#1. Pip install and PyPi :

↳ 'pip install' is a command line which is to install packages.

↳ The pytho package Index (PyPI) is a repository of software for the python programming language (PyPI) helps us to find and install software developed and shared by python community.

↳ (PyPI) is a repository for open source third party Python packages.

↳ We can use pip install at the command line to intsall these packages. By installing python from python.org or through the anaconda distribution you can also install pip.
pip is the simple way to download packages at your command line directly from the PyPI repository.

↳ We can work on external packages by installing after downloading : (i) Windows User : Comand Prompt
(ii) MacOS/Linux User : Terminal

To open Comand terminal do following steps :

Step 1 – Open Anaconda navigator
step 2 – After opening chick on New
step 3 – Write commands

→ We can use quit() command to end python command session.

↳ Codes/Command lines in command prompt/terminal in Anaconda Navigator :

```
pip install requests
pip install colorama
python
from colorama import init
init()
from colorama import Fore
print(Fore.RED + "HI I AM DHEERAJ")
print(Fore.BLUE + "HI I AM DHEERAJ")
print(Fore.GREEN + "HI I AM DHEERAJ")
print(Fore.YELLOW + "HI I AM DHEERAJ")
print(Fore.BLACK + "HI I AM DHEERAJ")
print(Fore.WHITE + "HI I AM DHEERAJ")
```

↳ Google Search "Python package for excel" then click on pythonexcel.org & click on documentation where you will find different packages.

Open the terminal in Anaconda Navigator

Then write,

```
pip install openpyxl
python
import openpyxl
```

#2. Module & Packages :

↳ Here we are going to learn about "How to create modules & packages". Modules are just .py scripts that you can call in another .py scripts. Using .py script in another .py script, this act is known as "Modules". Packages are collection of modules.

For understand this we need sublime text editor

↳ In Sublime text editor, Using two different files :

Step 1 — Open Subline Text Editor & create two new files.

Step 2 — Save two files with .py script (Say MP91.py and MP92.py)

step 3 — Code in MP91.py,

```
def a():
    print("Hi")
```

step 4 — Code in MP92.py,

```
from MP91.py import a
a()
```

Step 5 — Command in Anaconda Terminal,

location >> python MP92.py

[then press enter to get %p]

↳ Using single file,

Step 1 — Open Subline Text editor and create a new file.

Step 2 — Save this file with .py script (Say MP93.py)

Step 3 — Code MP93.py,

```
def a():
    print("Hi")
a()                    # "Hi"
```

Step 4 — Command in Anaconda terminal,

location >> Python MP93.py

[then press enter to get output]

↳ Calling modules together to form a package.

Follow the steps to understand :

Step 1 — Create a folder file and other file inside this file (same folder) [Say : pack and subpack respectively] on desktop

Step 2 – Open sublime text editor and create and save a two new files '__init__.py' in both files (i.e. pack & subpack)

Step 3 – Again create two new files in sublime text editor named 'p.py' and 'sp.py' and save with folder file and subfile respectively.

step 4 – Write the code in p.py,

```
def a():
    print("Pack")
```

step 5 – Write the code in sp.py,

```
def b():
    print("sub pack")
```

step 6 – Write the code in

step 6 – Create a file in sublime text editor and save outside both folders files, file & subfile (i.e. pack & subpack) named 'psp.py'

Step 7 – Write the code in psp.py,

```
from pack import p
from pack.subpack import sp
p.a()          # pack
sp.b()         # sub pack
```

3. __name__ and '__main__' :

↳ Module functions is being used as import when you are importing from a module.

```
__name__ == '__main__'   # True
```

builtin variable name get assigned a string. This statement is being run directly (assigned + checked : relation)

↳ Explanation & steps –

Step 1 – Create python file named one.py and two.py, three.py on desktop.

step 2 – Write a code in one.py,

```
def a():
    print("a() in one.py")
print("top level is one.py")
if __name__ == '__main__':
    print("one.py is running directly")
else:
    print("one.py is imported")
```

Step 3 – Write a code in two.py,

```
import one
def b()
    print("b() in ~~one~~two.py")
print("top level is two.py")
one.a()
if __name__ == '__main__':
    print("two.py is running directly")
else:
    print("two.py is imported")
```

Step 4 – Write a code in three.py,

```
import two
print("top level is three.py")
two.b()
if __name__ == '__main__':
    print("three.py is running directly")
else:
    print("three.py is imported")
```

Step 5 – Open terminal in Anaconda & use command line

```
ls
~~one.py~~
Python one.py
Python two.py
Python three.py
```

↳ Inside the program we are using import keyword which imports python file in other file then while running the second file, first file is imported and running the third file, firts and second file will be imported.

24/8/21

Python Decorators

↳ Pytho Decorator : It is an advanced python topic which allows us to ~~create~~ or decorate functions in python.

```
def simple_func():  # Do simple task
                    # and it return something
```

↳ We have two options :

(i) Add extra code (Functionality) to your old function

(or)

(ii) Create a new function which contains the old code and new code to that

↳ If we need or want to remove the extra functionality we need to delete it manually or make sure to have old function thats why decorators are used in this case.

Python has decorators that allows you to tack on extra functionality to an already existing fuction. Decorators use the '@' operator and are then placed on top of the original function.

```
@some_decorator
def simple_func():  # we can easily add
                    on extra functionalities
                    with a decorator.
```

↳

```
def func():
    return 1
func()      # 1
func        # <function __main__.func()>
```

```
def hello():
    return "hello"
hello()                 # 'hello'
hello                   # <function __main__.hello()>
greet = hello
greet()                 # 'hello'
hello()                 # 'hello'
del hello
hello()                 # → It will show an error.
greet()                 # 'hello'
```

↳ Lets create function inside anothe function :

→
```
def hello(name = "Jose"):
    print("The hello() function has been executed !!")
hello()   # The hello() function has been executed!!
```

→
```
def hello(name= "Jose"):
    print("The hello() function has been executed!!")
    def greet():
        return "\t This is greet() function inside hello() function"
    def welcom():
        return"\t This is welcome() function inside hello() function"
    print(greet())
    print(welcome())
    print("This is end of the function")
hello()          # The hello() function has been executed!!
                        This is greet() function inside hello() function
                        This is welcome() function inside hello() function
                 This is end of the function.
weclome()        # → It will show an error because
                    welcom function has been created inside
                    hello() function.
```

↳ def hello(name = 'Jose'):
    print("The hello function has been executed!!")
    def greet():
        return "\t This is greet() function inside hello()"
    def welcome():
        return "\t This is welcome() function inside hello()"
    # print(greet())
    # print(welcome())
    print("I am going to return a function")
    if name == 'Jose':
        return greet
    else:
        return welcome
my_new_func = hello('Jose')
print(my_new_func())
my_new_func() # The hello() function has been executed!!
                 I am going to return a function
                 This is the greet() function inside hello()
'\t This is the greet() function inside hello()'

↳ def cool():
    def super_cool():
        return 'I am very cool'
    return super_cool
some_func = cool()
some_func # <function __main__.cool.<locals>.super_cool()>
some_func() # 'I am very cool'

↳ Lets do passing a function as an argument:-
    def hello():
        return 'hi Jose'
    def other(some_def_func):
        print("other code runs here!")
        print(some_def_func())
hello # <function __main__.hello()>

```
hello()      # 'Hi Jose'
other(hello) # Other code runs here!
               Hi Jose
```

→ Return functions and function arguments are main tools to create decorates. We need tools for on/off switch when we add more functionality to the decorator.

```
def new_decorator(original_func):
    def wrap_func():
        print("Some extra code, before the original fuding")
        original_func()
        print("Some extra code, after the original function")
    return wrap_func
def func_need_decorators():
    print("I need to be decorated")
func_need_decorators()  # I need to be decorated

decorated_func= new_decorator(func_needs_decorators)
decorated_func()  #
```

```
Some extra code, befor the original function
I need to be decorated
Some extra code, after the original function.
```

```
@new_decorator
def func_needs_decorators():
    print("I need to be decorated")
func_needs_decorators()  #
```

```
Some extra code, before the original function
I need to be decorated
Some extra code, after the original funct
```

```
#@new_decorator
def func_needs_decorators():
    print("I need to be decorated")
func_needs_decorators()  # I need to be decorated
```

28/8/21

Generators With Python

Generators :-

→ We learned about how to create a functions with def (def : keyword) and the return statement. These Generator functions allows us to write a function that can send back a value and then later resume to pick up where it left off. This type of function is a generator in python which allowing us to generate a sequence of values over time. In this generator function we will be using 'yield' keyword.

→ The main difference in the syntax will be the use of a yield statement. When Generator function is compiled then they become an object that supports and iteration protocol which means they are called in your code they don't actually return a value and then exit. Generator function will automatically suspend and resume for their execution and state around the last point of value generation.

→ Advantages of Generator Functions :

i) Instead of having to compute an entire series of values up front, the generator computes one value waits until the next value is called for.

ex:- range() function doesn't produce a list in memory for all the values from start to stop instead it keeps track of past numbers and step size to provide flow of numbers.

If an user needs the list then they have to transform the generator to a list with list(range(0,10))

↳ Yield is a keyword in python that is used to return from a function without destroying the states of its local variable and when the function is called the execution starts from the last yield statement. Any function contains a yield keyword is termed as generator. Hence, yield is what makes a generator.

↳ How is yield different from return keyword in Python.

Yield is generally used to convert a regular python function into a generator.

Return is generally used for the end of the execution and returns the result to the caller statement. It replace the return of a function to suspend its execution without destroying local variables.

The yield statement is used in python generators to replace the return of a function to send a value back to its caller without destroying local variables.

↳
```
def cube_range(n):
    res = []
    for x in range(n):
        res.append(x**3)
    return res
cube_range(10)          # [0,1,8,27,64,125,216,343,512,729]
for i in cube_range(10):
    print(i)            # 0
                        # 1
                        # 8
                        # 27
                        # 64
                        # 125
                        # 216
                        # 343
                        # 512
                        # 729
```

↳ def cube_range(n):
    res=[]
    for x in range(n):
        yield x**3

cube_range(10) # <generator object ---------- >

list(cube_range(10)) #[0,1,8,27,64,125,216,343,512,729]

for i in cube rang(10)
    print(i) # 0
                 1
                 8
                27
                64
               125
               216
               343
               512
               729

↳ def gen_fibo(n):
    a=1
    b=1
    for i in range(n)
        yield a
        a,b = b, a+b

gen_fibo(10) # <generator object ---------- >

for num in gen_fibo(10):
    print(num) # 1
                   1
                   2
                   3
                   5
                   8
                  13
                  21
                  34
                  55

```
def gen_fibo(n):
    a=1
    b=1
    l=[]
    for i in rang(n):
        l.append(a)
        a,b = b,a+b
    print(l)
gen_fibo(10)   #[1,1,2,3,5,8,13,21,24,55]

def simple_gen():
    for x in range(3):
        yield x
for i in simple_gen():
    print(i)                    #  0
                                   1
                                   2

g=simple_gen()
g                    # <generator object ---------- -->
```

↳ Python next() function is used to fetch next item from the collection. It takes two arguments an iterator and a default value and returns an element. This methods calls an iterator and throws an error if no item is present. To avoid the error, we can set a default value.

iter() → This function is allows us to iterate object automatically.

iter() method is used to convert to an iterab to iterator. This presents another way to iterate the container i.e. access the elements iter() uses next() for accessing values.

```
print(next(g))      # 0
print(next(g))      # 1
print(next(g))      # 2
print(next(g))      # -> it will show error as all the values are yielded
```

```
s='hello'
for i in s:
    print(i)        # h
                    # e
                    # l
                    # l
                    # o
```

```
s_iter = iter(s)
next(s_iter)        # 'h'
next(s_iter)        # 'e'
next(s_iter)        # 'l'
next(s_iter)        # 'l'
next(s_iter)        # 'o'
next(s_iter)        # -> it will show an error
```

17/9/21

Advanced Python Modules

Collection Module - counter :→

Collection module - The collection module is a built-in module that implements specialized container datatypes providing alternatives to Python's general purpose builtin containers.

Counter - It is a dict subclass which helps count hashable objects. Inside of it elements are stored as dictionary keys and the counts of the objects are stored as the values.

If we use counter to a list which contains repeated objects then output will be in dictionary format where keys indicates object and value indicates number of repetations of each object.

```
from collections import Counter
l=[1,1,1,2,2,3,3,3,3,4,5,5,5,5,5]
l      # [1,1,1,2,2,3,3,3,3,4,5,5,5,5,5]
Counter(l)  # Counter({1:6,2:5,3:2,4:6,5:3})
S='asasss qvavav'
Counter(S)  # Counter({'a':5,'s':4,'v':3})

string='HI I AM DHEERAJ FROM KIIT DU. I AM
    FROM ODISHA'
words = string.split()
words          # ['HI','I','AM','DHEERAJ',
               'FROM','KIIT','DU.','I','AM',
               'FROM','ODISHA']
```

```
Counter(words)  # Counter({'HI' : 1
                           'I' : 2
                           'AM' : 2
                           'DHEERAJ' : 1
                           'FROM' : 2
                           'KIIT' : 1
                           'DU' : 1
                           'ODISHA' : 1 })

c = Counter(words)
c
                # Counter({'HI' : 1
                           'I' : 2
                           'AM' : 2
                           'DHERAJ' : 1
                           'FROM' : 2
                           'KIIT' : 1
                           'DU.' : 1
                           'ODISHA' : 1 })

~~c.common~~
.most_common()

c.most_common(1)  # [('I', 2)]
c.most_common(2)  # [('I', 2), ('AM', 2)]
c.most_common(3)  # [('I', 2), ('AM', 2), ('FROM', 2)]
c.most_common(4)  #[('I', 2), ('AM', 2), ('FROM', 2), ('HI', 1)]
c.most_common(5)  #[('I', 2), ('AM', 2), ('FROM', 2), ('HI', 1),
                                                ('DHEERAJ', 1)]
c.most_common(7)# [('I', 2),
                   ('AM', 2),
                   ('FROM', 2),
                   ('HI', 1),
                   ('DHEERAJ', 1),
                   ('KIIT', 1),
                   ('DU.', 1) ]
```

"""Common patterns when using the Counter() object

```
sum(c.values())                 -> Total of all counts
c.clear()                       -> Reset all counts
list(c)                         -> list unique elements
set(c)                          -> Converts to a set
dict(c)                         -> Converts to regular dictionary
c.items()                       -> Converts a list of (element, count) pairs
Counter(dict(list_of_pairs))    -> Converts from a list of (element, count) pairs
c.most_common()[:-n-1:-1]       -> n least common elements
c+=Counter()                    -> remove zero and negative counts
```

'''

```
string = 'HI I AM DHEERAJ FROM KIIT DU. I AM FROM ODISHA'
words = string.split()
c = Counter(words)
c
                # Counter({'HI' : 1,
                           'I' : 2,
                           'AM' : 2,
                           'DHEERAJ' : 1,
                           'FROM' : 2,
                           'KIIT' : 1,
                           'DU.' : 1,
                           'ODISHA' : 1})
sum(c.values())  # 11
list(c)          # ['HI', 'I', 'AM', 'DHEERAJ', 'FROM',
                   'KIIT', 'DU.', 'ODISHA']
set(c)  # {'AM', 'DHEERAJ', 'DU.', 'FROM', 'HI', 'I',
          'KIIT', 'ODISHA'}
dict(c) # {'HI' : 1
           'I' : 2
           'AM' : 2
           'DHEERAJ' : 1
           'FROM' : 2
           'KIIT' : 1
           'DU.' : 1
           'ODISHA' : 1}
```

```
c.items()  # dict_items([('HI', 1), ('I', 2), ('AM', 2),
                        ('DHEERAJ', 1), ('FROM', 2),
                        ('KIIT', 1), ('DU.', 1),
                        ('ODISHA', 1)])
Counter(dict(c))  # Counter({'HI' : 1
                           'I' : 2
                           'AM' : 2
                           'DHEERAJ' : 1
                           'FROM' : 2
                           'KIIT' : 1
                           'DU.' : 1
                           'ODISHA' : 1})
c.most_common()[: -min(c.values()) - 1 : -1]
                        # [('ODISHA', 1)]
c += Counter()
c                  # Counter({'HI' : 1
                           'I' : 2
                           'AM' : 2
                           'DHEERAJ' : 1
                           'FROM' : 2
                           'KIIT' : 1
                           'DU.' : 1
                           'ODISHA' : 1})
c.clear()
c                  # Counter()
Counter(dict(c))   # Counter()
dict(c)            # {}
c.items()          # dict_items([])
set(c)             # set()
list(c)            # []
```

↳ # Collection Module - default dict :→

"""default dict is a dictionary like object which provides all methods provided by dictionary but takes first argument as a default datatype for the dictionary. A defaultdict will never raise a keyerror. Any key that does not exist gets the value returned by the default factory.
"""

```
from collections import defaultdict
d = {'k1' : 1}
d['k1']                    # 1
#d['k2']                   -> It will show an error
d = defaultdict(object)
d['k2']                    # It gives factory condition of the
                             object
                           # <object at 0x1....>

for i in d:
    print(i)               # k2

d = defaultdict(lambda : 0)
d['one']                   # 0
d['Two']                   # 0
d                          # defaultdict(<function _main_.<lambda>,
                                         {'one':0,'two':0})
for i in d:
    print(i)               # one
                             two

from collections import defaultdict
d = {'k1': 1}
d['k1']                    # 1
for i in d:
    print(i)               # k1
```

```
d = defaultdict(lambda : 0)
d['one']                    # 0
d['two'] = 2
d                           # defaultdict(<function __main__.<lambda>
                            >(), {'one': 0, 'two': 2})
```

↳ # Collection Module - OrderedDict :→

""" An OrderedDict is a dictionary that remembers the orders in which its contents are added. """

```
print('Normal Dictionary')
d = {}
d['a'] = 'A'
d['b'] = 'B'
d['c'] = 'C'
for k,V in d.items():          #  Normal Dictionary
    print(k,' : ',V)           #  a : A
                               #  b : B
                               #  c : C
d                  # {'a':'A', 'b':'B', 'c':'C'}

d = {}
d['a'] = 1
d['b'] = 2
d['c'] = 3
d                  # {'a':1, 'b':2, 'c':3}
for i,j in d.items():
    print(i,' : ',j)  #  a : 1
                      #  b : 2
                      #  c : 3
from collections import OrderedDict
d = OrderedDict()  # I
d['a'] = 1
d['b'] = 2
d['c'] = 3
for i,j in d.items():
    print(i,' : ',j)  #  a : 1
                      #  b : 2
                      #  c : 3
```

OrderedDict() → It will arrange pair of keys and values in dictionary format.

→
```
d1 = {}
d1['a'] = 1
d1['b'] = 2
d1['c'] = 3

d2 = {}
d2['c'] = 3
d2['b'] = 2
d2['a'] = 1
print('d1 :', d1)   # d1: {'a':1, 'b':2, 'c':3}
print('d2 :', d2)   # d2: {'c':3, 'b':2, 'a':1}
if (d1 == d2):
    print('True')   # True
else: print('False')
```

→
```
d1 = OrderedDict()
d1['a'] = 1
d1['b'] = 2
d1['c'] = 3

d2 = OrderedDict()
d2['c'] = 3
d2['b'] = 2
d2['a'] = 1
print('d1 :', d1)   # d1 : OrderedDict([('a',1),('b',2),('c',3)])
print('d2 :', d2)   # d2 : OrderedDict([('c',3),('b',2),('a',1)])
if (d1 == d2):
    print('True')
else: print('False') # False
```

↳ # Collection Module - namedtuple :→

A namedtuple is an another class that contains keys and that are mapped to some values. Python string of fields, separated by spaces.

```
t = (1,2,3)
t[0]    # 1
t[1]    # 2
t[2]    # 3
```

→ from collections import namedtuple
- Dog = namedtuple('Dog', 'age breed name')
sam = Dog(age=2, breed='Lab', name='Sammy')
sam # Dog(age=2, breed='Lab', name='Sammy')

sam.age # 2
sam.breed # 'Lab'
sam.name # 'Sammy'
sam[0] # 2
sam[1] # 'Lab'
sam[2] # 'Sammy'

→ Cat = namedtuple('cat', 'fur claws name')
c = Cat(fur='Fuzzy', claws=False, name='kitty')
c # cat(fur='Fuzzy', claws=False, name='kitty')
c.fur # Fuzzy
c.claws # False
c.name # 'kitty'
c[0] # 'Fuzzy'
c[1] # False
c[2] # 'kitty'

↳ # Datetime :-

import datetime
t = datetime.time(5,25,1)
print(t) # 05:25:01
t.hour # 5
t.minute # 25
t.second # 1
t.microsecond # 0
t # datetime.time(5,25,1)
datetime.time # datetime.time
datetime.timezone # datetime.timzone

```
datetime.time.min  # datetime.time(0,0)
print(datetime.time.min)  #  00:00:00
                # It prints initial time of the day.
                  Prints new time just after 24hrs.
print(datetime.time.max) #  23:59:59:999999
                # It prints final time of the day
                  Prints time just before 24hrs
                  Time just before starting of new day
datetime.time.max  #  datetime.time(23,59,59,999999)
```

↳ Time instance only hold values of time.

```
print(datetime.time.resolution) #  00:00:00:000001
                # It will show in microseconds. Time
                  has resolution in microseconds.
today = datetime.date.today() # date.today() will return
                                gives today's date
print(today)  #  2021-09-17

today.timetuple()  # It is a basic structure which has
                     (year, month, day, hour, min, sec, week day
                     , yearday, tm_isdst)
today.year  # 2021
today.month # 9
today.day  # 17
datetime.date.min  # datetime.date(1,1,1)
print(datetime.date.max)  #  9999-12-31
print(datetime.date.min) #  0001-01-01
datetime.date.resolution  # datetime.timedelta(day=1)
print(datetime.date.resolution) #  1 day, 0:00:00
d1 = datetime.date(2021, 12, 7)
print(d1)          #  2021-12-07
d2 = datetime.replace(year=1990) # We can set year
print(d2)          #  1990-12-07   as per our convinience
```

```
d2      # datetime.date(1990, 12, 7)
d2 = d1.replace(month=8, day=30) # we can set month & day as per our convinience
print(d2)          # 2021-08-30
                   # datetime.date(2021, 12, 7)
d1
print(d1)          # 2021-12-07
                   # datetime.date(2021, 8, 30)
d2
print(d2)          # 2021-08-30
d1-d2              # datetime.timedelta(days=99)
                   < For difference in time>
t1 = datetime.date(2020, 12, 8)
t2 = datetime.date(2021, 12, 7)
print(t1-t2)       # -364 days, 0:00:00
print(t2-t1)       # 364 days, 0:00:00
```

→
```
import datetime
t = datetime.time(4, 20, 1)
print(t, '\n')                          # 04:20:01
print('HOUR :', t.hour)                 # HOUR: 4
print('MINUTE :', t.minute)             # MINUTE: 20
print('SECOND :', t.second)             # SECOND: 1
print('MICRO-SECOND :', t.microsecond)  # MICROSECOND: 0
print('TIME ZONE :', t.tzinfo)          # TIMEZONE: None
```

→
```
print('Earliest :', datetime.time.min)  # Earliest : 00:00:00
print('Latest :', datetime.time.max)    # latest : 23:59:59:999999
print('Resolution :', datetime.time.resolution) # Resolution :
                                                  0:00:00:000001
```

→
```
today = datetime.date.today()
print(today, '\n')                      # 2021-09-17
print('CTIME :', today.ctime())         # CTIME: Fri Sep 17 00:00:00 2021
print('TUPLE :', today.timetuple())     # TUPLE:
print('ORDINAL :', today.toordinal())   # ORDINAL : 738050
print('YEAR:', today.year)              # YEAR: 2021
print('MONTH :', today.month)           # MONTH: 9
print('DAY :', today.day)               # DAY: 17
```

Python Debugger :→

→ Python debugger module → pdb (debugging environments for python program)

→
```
import pdb
x=[1,3,4]
y=2
z=3
result1= y+z
print(result1)
pdb.set_trace()      # This set_trace() module will give us why error happened. (debug)
regult2 = y+x
print(result2)       # pdb q is to quit
```

→
```
import pdb
x=[1,2,3]
y=4
z=5
res = y+z
print(res)
pdb.set_trace()
r1=x+y
print(r1)
```

→
```
import pdb
x=[1,3,4]
y=2
z=3
result1 = y+z
print(result1)
result2 = x+y
print(result2)
```

Timing your code :→

→ Python has builtin 'timing module' : timeit

→ This 'timeit' module provides a simple way to time small bits of Python code. It has both command lines interfaces as well as callable one. It avoids number of common traps for measuring execution timeit!

```
import timeit
"_".join(str(n) for n in range(100)) # '0-1-2-...-98-99'
timeit.timeit("_".join(str(n) for n in range(100))) # 0.018...32
timeit.timeit("_".join(str(n) for n in range(100))', number=1000)
                                                    # 0.272...02
timeit.timeit("_".join([str(n) for n in range(100)])', number=1000)
timeit.timeit("_".join(map(str, range(100)))', number=10000)

# builtin magic -> %timeit
%timeit ("_".join(map(str, range(100)))) # 16µs + 116 ns per loop (mean + std. dev. of 7 runs, 100000 loop each)
%timeit ("_".join(str(n) for n in range(100)))
%timeit ("_".join(str(n) for n in range(100))
%timeit "_".join([str(n) for n in range(100)])
%timeit "_".join(str(n) for n in range(100))
```

Regular Expressions - RE in Python :->

-> It specifices a set of strings that matches it and checks if a particular string matches a given regular expression.

```
-> import re
   patterns = ['term1', 'term2']
   text = 'This is a string with term1, but not the other term'
   re.search('hello', 'hello world!') #<re.match object;
                                       span=(0,5), match='hello'>
```

```
-> import re
   patterns = ['term1', 'term2']
   text = 'This is a string with term1, but not term'
   for i in patterns:
       print('Searching for "%s" in : \n"%s"' %(i, text))
       if re.search(i, text):
           print('\n Match found')
       else:
           print('\n Match not found') #
print(re.search('h', 'w')) # None
```

```
match = re.search(patterns[0], text)
type(match)                    # re.match
match.start()                  # Gives the index of the match
                               'term 1' is matched in text at index
                               22.
match.end()                    # 27
re.split('@', 'hi@123')        # ['hi', '123']
```

→
```
split_term = ','
statement = 'My name is Dheeraj, Reading in KIIT, Odisha'
re.split(split_term, statement)   # ['My name is Dheeraj',
                                    'Reading in KIIT', 'Odisha']
'My name is Dheeraj'.split()   # ['My', 'name', 'is', 'Dheeraj']
re.findall('match', 'Here is one match, here is another match')  # it
                                                                   finds all matches
                               # ['match', 'match']
```

→
```
def multi_re_find(patterns, phrase):
    """Takes in a list of regex patterns
       Prints a list of all matches."""
    for i in patterns:
        print('Searching the phrase using recheck: %r' %i)
        print(re.findall(i, phrase))
        print('\n')
test_phrase = 'sdsd..ssssddd...sddddsddd...dsds...sdddd'
# REPITITION
test_patterns1 = ['sd*',       # followed by zero or more d's
                  'sd+',       # s followed by one or more d's
                  'sd?',       # s followed by zero or one d's
                  'sd{3}',     # s followed by three d's
                  'sd{2,3}']   # s followed by two to three d's
multi_re_find(test_patterns1, test_phrase)
# Character CHARACTER SETS
test_patterns2 = ['[sd]',      # either s or d
                  's[sd]+']    # s followed by one or more sd
multi_re_find(test_patterns2, test_phrase)
```

Exclusion → we can use ^ to exclude terms by incorporating it into the bracket syntax notation.

```
test_phrase = 'This is a string! But it has punchuations,
               How can we remove it ?'
re.findall('[^!.?]+', test_phrase)
        # ['This is a string', 'But it has punchuations
            , How can we remove it' ]
re.findall('[^ ]', test_phrase) # ['T',
                                  'h',
                                   :
                                  't',
                                  '?',]
re.findall('[^ ]+', test_phrase) # Syntax:
        # re.findall('[^ ]') split by letters
        # re.findall('[^ ]+') split by words
# ['This',
   'is',
    :
   'remove',
   'it?']
```

CHARACTER RANGES :→

→ A more compact format using charater ranges lets you to define a character set to include all of the contigious characters between a start and stop point. Format used [start-end].

```
test_phrase = 'This is an example sentence. Lets see
               if we can find some letters'
test_patterns = ['[a-z]',      #sequences of lower case letters
                 '[A-Z]+',     #Sequences of Upper case letters
                 '[a-zA-Z]+',  #sequence of upper or lower case letters
                 '[A-Z][a-z]+'] # One uppercase letter followed
                                  by lower case letter.
multi_re_find(test_patterns, test_phrases)
```

↳ # Escape Codes :→

```
"""\d    → A digit
   \D    → A non-digit
   \s    → white Space
   \S    → Non White Space
   \w    → alphanumeric
   \W    → Non Alphanumeric """
print('hello \n new line')   # hello
                               new line
```

→ test_phrase = 'This is a string with some numbers 1233 and a symbol #hashtag'

```
test_patterns = [ r'\d+',  # sequence of digits
                  r'\D+',  # sequence of non-digits
                  r'\s+',  # sequence of white space
                  r'\S+',  # sequence of non-white Space
                  r'\w+',  # sequence of alpha numeric chartes
                  r'\W+',  # sequence of non-alphanumeric characters
                ]
multi_re_find(test_patterns, test_phrase) #
```

↳ # StringIO :→

"""The StringIO module implements an inmemory file like objects. This can be used as input or output to most functions that would expect a standard object."""

```
from StringIO import StringIO
message = 'This is just a normal string' # Arbitary String
f = StringIO.StringIO(message) # <Use StringIO method to
                                 set as file object>

f.read()

f.write('Second line written to file like object')

f.seek(0)

f.read()
```

19/9/21

Advanced Python And Objects & Data Structure

Advanced Numbers :→

"""
hex() → It converts a number into hexadecimal
oct() → It converts a number into octal decimal
bin() → It converts a number into binary format (number)
pow() → It takes two arguments pow(a,b).
a is a base. b is an exponent.
a to the power of b.
abs() → It returns absolute value.
round() → If it takes two arguments then it will round off to decimal points. If it takes one argument then it will give precision of decimal number. (number nearest to decimal number.)
"""

```
hex(12)      # '0xc'
oct(0xc)     # '0o14'
oct(12)      # '0o14'
oct(8)       # '0o10'
hex(512)     # '0x200'
bin(1234)    # '0b10011010010'
bin(128)     # '0b10000000'
bin(0)       # '0b0'
bin(1)       # '0b1'
bin(5)       # '0b101'
bin(12)      # '0b1100'
2**4         # 16
pow(2,4)     # 16
```

→ If pow takes 3 arguments then pow(a,b,c)
a is to the power b is modalo of c.

```
pow(2,4,3)      # 1      → (2**4)%3
pow(2,4,2)      # 0
pow(2,4,4)      # 0
pow(2,4,5)      # 1
pow(2,1/2)      # 2.0
abs(2)          # 2
abs(2.5)        # 2.5
abs(-4)         # 4
abs(-2.5)       # 2.5
abs(-2)         # 2
round(3.1)      # 3
round(3.9)      # 4
round(3.12345463234,4)      # 3.1235
round(10/3,4)               # 3.3333
```

↳ # Advanced String :→

"""

- .upper() → It converts string into upper case
- .lower() → It converts string into lower case
- .count(") → It counts a character that has repeated in a string. It returns number of times a character has repeated.
- .find(" ") → It returns index value where 1st occurance of a character is occoured in a string else -1.
- .title() → It converts 1st letter to upper case of each word of string.
- .capitalize() → It converts 1st letter of an each word of a string to upper case.
- .center(n," ") → It returns in format of <center(sizeof string, "string") It shifts a string to the center position of given range "n".
- .expandtabs() → It expands special charater '\t' in a string.
- .endswith(") → It checks whether a character is in last index position of a string. It returns boolean type value.
- .partition(" ") → It splits a string with 1st occurance of a string character. It returns tuple.

.split(',') → It splits a string by a character separated string and store it as list. It returns list.

.split() → It splits a string into words and store it as list. If returns list.

.isspace() → It returns boolean value. It checks wether a character or a string is spaces.

.isupper() → It returns boolean value. It checks whether a character or a string is in upper case or not.

.islower() → It returns boolean value. It checks whether a character or a string is in lower case or not.

.isalpha() → It returns boolean value. It checks whether a character or a string is alpha (Alphabet) or not.

.isnumeric() → It checks whether a string or character is numeric number or not. It returns boolean value.

.isalnum() → It returns boolean value. It checks whether a character or a string is alpha numeric or not.

.istitle() → It returns boolean value. It checks whether a character or a string is title or not.

```
'''
s='hello world'
s.capitalize()          #  'Hello world'
'123'.isnumeric()       #  True
s='hello world'
s.title()               #  'Hello World'
s                       #  'hello world'
s='hello world'
s.istitle()             #  False
s.upper()               #  'HELLO WORLD'
s                       #  'hello world'
s=s.upper()
s                       #  'HELLO WORLD'
s.lower()               #  'hello world'
```

```
s.count('O')          # 2
s.count('o')          # 0
s.find('o')           # -1
s.find('O')           # 4
                      # 'HELLO WORLD'
s
s.find('O')           # 4
s=s.lower()
                      # 'hello world'
s
s.count('O')          # 0
s.count('o')          # 2
s.find('o')           # 4
s.find('O')           # -1
                      # 'hello world'
s
s.center(20,'z')      # 'zzzzhello worldzzzzz'
s.center(20,'.')      # '....hello world.....'
s.center(50,'.')      # '.....hello world.....'
'hello\thi'           # 'hello\thi'
print('hello\thi')    # hello    hi
'hello\t\nhi'.expands() # 'hello    \n hi'
s='hello'
s.alpha
s.isalnum()           # True
s.isalpha()           # True
s                     # 'hello'
s.islower()           # True
s.isupper()           # False
s                     # 'hello'
s.isspace()           # False
s='hello world'
s.isspace()           # False
s=' '
s.isspace()           # True
s=''
s.isspace()           # False
```

```
s=' '
s.isspace()                          # True
s.istitle()                          # False

s='Hi Hello World'
s.istitle()                          # True

s='Hi Hello World'.upper()
print(s)                             # HI HELLO WORLD
s.title()                            # False

s='Hi Hello World'.capitalize()
print(s)                             # Hi hello world
s.istitle()                          # False
s.isupper()                          # False
'HELLO WORLD'.isupper()              # True
'HELLO WORLD'.islower()              # False
s='hello'                            # 'hello'
s
s.endswith('o')                      # True
s[-1]=='o'                           # True
s.split('e')                         # ['h','llo']
s.split('l')                         # ['h','','o']
s                                    # 'hello'

s='this is a cat'
s=s.split(' ')
s                                    #['this','is','a','cat']

s='this is a cat'
s=s.split('i')
s                                    #['th','s ','s a cat']

s='this is a cat'
s                                    # 'this is a cat'
s.partition('i')                     # ('th','i','s is a cat')
s.partition(' ')                     # ('this',' ','is a cat')
s=s.partition(' ')
type(s)                              # tuple
```

Advanced Sets :→

"""
.add() → It adds new elements to a set. Sets won't take duplicate elements
.clear() → It removes all elements from a set
.copy() → It returns copy of a set. original set don't effect the copy.
.difference() → It returns the difference of two sets.
.difference_updates() → It returns set1 after removing elements found in set2.
.discard() → It removes all elements from a set if it is a in a set.
.intersection() → It returns intersection of two or more sets.
.intersection_update() → It returns intersection. It will update a set with intersection of itself and another.
.isdisjoint() → It will return True if two sets have a null intersection.
.issubset() → It will reports whether another set contains this set.
.issuperset() → It will reports whether set contains other set.
.symmetric_difference() → It will return the symmetric difference of two sets as a new sets.
.Symmetric_update() →

.union() → It will returns the union of two sets.
.update() → It will update a set with the union of itself and others.
"""

```
S = set()
S.add(1)
S.add(2)
S                          # {1,2}
S.add(3)
S                          # {1,2,3}
S.add(2)
S                          # {1,2,3}
S.clear()
S                          # set()
```

```
s={1,2,3}
s.add(4)
s                    # {1,2,3,4}
sc=s.copy()          # (or) sc=s
sc                   # {1,2,3,4}
s.difference(sc)     #   set()
s1={1,2,3,4,5}
s2={1,3,5}
s1-s2                # {2,4}
s1.difference_update(s2)
s1                   # {2,4}
s2.difference(s1)    # {1,3,5}
s3={1,2,4}
s4={1,3,5}
s3-s4                # {2,4}
s4-s3                # {3,5}
s3.difference(s4)    # {2,4}
s4.difference(s3)    # {3,5}
s                    # {1,2,3,4}
s.discard(2)
s                    # {1,3,4}
s.discard(13)
s                    # {1,3,4}
s1={1,2,3}
s2={1,2,4}
s1.intersection(s2)  # {1,2}
s1                   # {1,2,3}
s1.intersection_update(s2)
s1                   # {1,2}
```

```
s1 = {1, 2}
s2 = {1, 2, 4}
s3 = {5}
s1.isdisjoint(s2)   # False  <It returns True if
                    # both sets having different elements
                    # or not having elements otherwise it
                    # returns False.
s1.intersection(s2)          #  {1, 2}
s1.isdisjoint(s3)            #  True
s1.intersection(s3)          #  set()
s1                           #  {1, 2}
s2                           #  {1, 2, 4}
s                            #  {1, 3, 4}
```

s1.issubset(s) # False <It checks whether a set is subset of other set or all elements of set is contained in other set or not then it returns True otherwise False.

```
s2.issubset(s)  #  False
s.issubset(s1)  #  False
s.issubset(s2)  #  False
s1.issubset(s2) #  True
s2.issubset(s1) #  False
```

```
s1.issubset(s1)   # True  <A set is subset of itself)
s2.is subset(s2)  # True
s.issubset(s)     # True
```

```
s1.issuperset(s1) # True <A set is superset of itself
s2.issuperset(s2) # True
s.issuperset(s)   # True
```

```
s                 #  {1, 3, 4}
s1                #  {1, 2}
s2                #  {1, 2, 4}
```

s.issuperset(s1) # It checks whether some elements in a set is must matches with all elements in a set. A set from other set. All elements in second set must present in first set.

#False

```
s.issuperset(s2)        # False
s1.issuperset(s)        # False
s1.issuperset(s2)       # False
s2.issuperset(s)        # False

s1                      # {1,2}
s2                      # {1,2,4}
s                       # {1,3,4}

s1.symmetric_difference(s2)     # {4}
s.symmetric_difference(s2)      # {2,3}
s.symmetric_difference(s)       # set()
s1.symmetric_difference(s)      # {2,3,4}
s2.symmetric_difference(s1)     # {4}
s2.symmetric_difference(s)      # {2,3}

s1.union(s2)                    # {1,2,4}
s1.update(s2)                   # {1,2,4}
s1
```

↳ # Advanced Dictionary :→

```
d={'a':1,'b':2}
d                               # {'a':1,'b':2}

{x:x**2 for x in range(4)}      # {0:0,1:1,2:4,3:9}
                                # <DICTIONARY COMPREHENSION>

{k:v**2 for k,v in zip(['a','b'],range(10))}
                                # {'a':0,'b':1}

{k:v**2 for k,v in zip(['a','b','c','d','e'],range(10)}
                                # {'a':0,'b':1,'c':4,'d':9,'e':16}

for k in d.value()
    print(k)                    # 1
                                # 2
```

```
for k in d.items():
    print(k)                    # ('a', 1)
                                # ('b', 2)

for k in d.keys():
    print(k)                    # a
                                # b

d.items()    # dict_items([('a', 1), ('b', 2)])
d.values()   # dict_values([1, 2])
d.keys()     # dict_keys(['a', 'b'])
```

↳ # Advanced Lists :⟶

"""
- .append() → It will append a new element to a list at the end.
- .count() → It takes in an element and returns the numbers of times it occours in your list.
- .extend() → It extends list by appending elements from the iterable.
- .index() → It will return the index of element which is placed as an argument. The element is not in the list then an error is returned.
- .insert() → It takes two arguments : <syntax - insert(index, object)>
- .pop() → It will allows us to pop the last element of the list.
- .remove() → It will remove the first occurance of a value.
- .reverse() → It will reverse a list
- .sort() → It will sort all elements in a list <ascending order>
- .sort(reverse = True) → It will sort all elements in a list. <descending Order>

```
l = [1, 2, 3]
l.append(4)
l                   # [1, 2, 3, 4]
l.count(10)         # 0
l.count(1)          # 1
l.count(3)          # 1
```

```
x= [1,2,3]
x.append([4,5])                    # [1,2,3,[4,5]]
x
x= [1,2,3]
x.extend([4,5])                    # [1,2,3,4,5]
x                                  # [1,2,3,4]
l                                  # 1
l.index(2)
# l.index(0)    -> It will show an error
l                  # [1,2,3,4]
l.insert(2,'inserted')   # string in index positio 2
l                  # [1,2,'inserted',3,4]
ele=l.pop()
ele                # 4
l.remove(1)
l                  # [2,3]
l=[1,2,3,4,3,2,1]
l.remove(2)              #<It will remove first ocurr
l        # [1,3,4,3,2,1] of an element >
l.reverse()
l              # [1,2,3,4,3,1]
l.sort()
l              # [1,1,2,3,3,4]
l.sort(reverse=True)
l                  # [4,3,3,2,1,1]
k=set(l)
k                  # {1,2,3,4}
l=[1,2,3,4,3,2,1]
l.insert(2,9)          # It will insert new element at index
l                          # [1,2,9,3,4,3,2,1]
l=[1,2,3,4,3,2,1]
l.pop(2)               # It will pop out an element at index
l                          # [1,2,4,3,2,1]
```

21/9/21

Introduction to GUIs

4→ # 1 Ineteract functionality with GUIs : →

""" Python GUI with IPyWidgets :
GUI stands for 'Graphical User Interfaces'. We can build simple graphical user interfaces (GUI) with Jupyter Notebook.
"""

→ from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets

```
def func(x):
    return x
interact(func, x=True)      # True
```

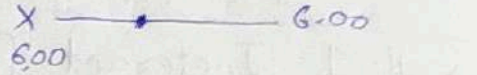
☑ x

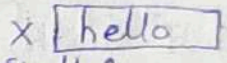
```
# Interact is to know what kind of feature & what kind of
# data type is passed. int or str or checkbox.
interact(func, x=10)        # x ——•—— 10
                            # 10
interact(func, x='hello')   # x [hello]
                            # 'hello'
interact(func, x=fixed('hello'))   # 'hello'
                            # fixed is used to fix a value to the variable.
```

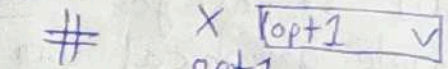
→ # We can do this using a decorator.

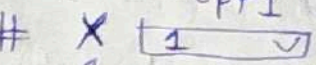
```
@interact(x=True, y=1.0)
def g(x,y):
    return (x,y)
                            # ☑ x
                            # y ——•—— 1.00
                            # (True, 1.0)

@interact(x=True, y=fixed(1.0))
def g(x,y):
    return (x,y)
                            # ☑ x
                            # (True, 1.0)
interact(func, x=10)        # x ——•—— 10
                            # 10
# minimum range of value x: -x, maximum value of x: 3x
interact(func, x=widgets.IntSlider(min=-100, max=100, step=1, value=0))
                            #(min_range, max_range, step_size, initial_value)
                            # x ——•—— 0
                            # 0
interact(func, x=(-100,100,1))   # x ——•—— 0
                            # 0
                            #(min_range, max_range, step_size)
interact(func, x=(-6.0, 10.0, 0.1))   # x ——•—— 2.00
                            # 2.00
```

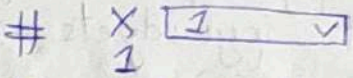
If we want to give initial value then then use interact decorator.

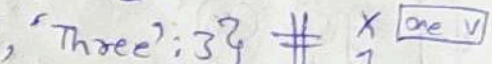
```
@interact(x=0.0,20.0,0.5)
def h(x=6.0):
    return x
```
#



```
interact(func, x='hello')
```
#



```
interact(func,x=['opt1','opt2','opt3'])
```
#



```
interact(func, x=[1,2,3,4,5])
```
#



```
interact(func, x=[1,2,3,4,[5,6]]
```
#



```
interact(func, x={'One':1,'Two':2,'Three':3}
```
#



```
def func(x)
    return x**2
interact(func, x=50)
```
#


Interactive

→
```
from IPython.display import display
def f(a,b):
    display(a)
    display(b)
    display(a+b)
    return a+b
w=interactive(f,a=1,b=2)
w
type(w) #
w.children #
display(w) #
```
#


#

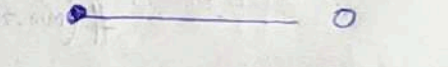

#

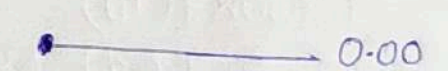
#
1
2
3

ipywidgets.widgets.interaction.interactive

#
a
1
b
2
1
2
3

→ # GUI Widgets basics :→

→
```
import ipywidgets as widget
widgets.IntSlider()
```
#



```
import ipywidgets as widget
display(widgets.FloatSlider()) #
```


```
import ipywidgets as widget
w = widgets.IntSlider()
from IPython.display import display
display(w)                    # ●——— 0
display(w)                    # ●——— 0      } interlinked with each other
w.close()
display(w)                    # IntSlider(value=0)
w = widgets.FloatSlider()
display(w)                    # ●——— 0.00
w.value                       # 0.0
W = widgets.IntSlider()
w.value                       # 0
w.keys                        # [ ... ]

w.value = 50
w.max = 1000
display(w)                    # ●——— 50   (before)
                              # ●——— 350  (after)

w.value = 350
w.min = 200
w.step = 20
w.max = 1000
display(w)                    # ●——— 350
                                  3

a = widget.FloatText()
display(a)                    # [ 0 ]
```

a and b are not interlinked to each other:

```
import ipywidgets as widget
from IPython.display import display
a = widget.FloatText()
b = widget.FloatSlider()
display(a, b)                 # [ 0 ]
                              # ●——— 0.00
```

a and b are interlinked to each other:

```
import ipywidgets as widget
from IPython.display import display
a = widget.FloatText()
b = widget.FloatSlider()
display(a, b)                 # [ 0 ]
mylink = widget.jslink((a, 'value'), (b, 'value'))   # ●——— 0.00
```

```
import ipywidgets as widget
from IPython.display import display
a = widget.FloatText()
b = widget.FloatSlider()
a.value = 100
b.step
b.max = 1000
display(a,b)
mylink = widget.jslink((a,'value'),(b,'value'))
```

100 — 100.0

```
# We can set range or limit , 'max'
import ipywidgets as widget
from IPython.display import display
a = widget.FloatText()
b = widget.FloatSlider()
display(a,b)
mylink = widget.jslink((a,'value'),(b,'max'))
mylink.unlink()  # It breaks intersection blw a and b.
display(mylink) #
```

5 — 5.00

↳ # List of possible widgets :→

"""Complete list → Complete list of GUI widgets available to you, you can list the registered widget types. Widget is the base class""".

→
```
import ipywidgets as widgets
for item in widget.Widget.widget_types.items():
    print(item[0])
```

'Numeric Widget → IntSlider'

```
widget.IntSlider(
    value = 7,
    min = 0,
    max = 10,
    step = 1,
    description = 'Test:',
    disabled = False,
    continous_update = False,
    orientation = 'horizontal',
    readout = True,
    readout_format = 'd'
)
```

Test: —— 7

```
"Numeric Widgets - FloatSlider"
widgets.FloatSlider(
    value = 7.3,
    min = 0, max = 10.0, step = 0.2, description = "Test:"
    disabled = False, continous_update = False, orientation =
    'horizontal', readout = True, readout_format = '.3f'
)                    # Test: ——●—— 7.300

widgets.FloatSlider(
    value = 7.3,
    min = 0,
    max = 10.0, step = 0.2, description = 'Test:', disabled = False,
    Continous_update = False, orientation = 'vertical', readout = True,
    readout_format = '.1f'
)   #                          Test:
                               |
                               7.3

# Numerical widgets - IntRangeSlider
widget.IntRangeSlider(
    Value = [2,8],
    min=0, max = 10, step = 1, description = 'Test:',
    disabled = False, continous_update = False, orientation = 'horizontal',
    readout = True, readout_format = 'd'
)                    # Test: ——●——●—— 2-8

# Numeric Widgets - FloatRangeSlider
widgets.FloatRangeSlider(
    Value = [2.4, 8.3],
    min = 0, max = 10.0, step = 0.1, description = 'Test:',
    disabled = False, continous_update = False, orientation = 'horizontal',
    readout = True, readout_format = '.1f'
)   #            Test: ——●——●—— 2.4 - 8.3

# Numeric Widgets - IntProgress
widgets.IntProgress(
    Value = 3,
    min = 0, max = 10, step = 1, description = 'Loading:',
    bar_style = '',
    orientation = 'horizontal'
)                              Loading: [▨▨▨▨▨▨▨▨▨▨]

# → Color : 'success'  = LIGHT GREEN ,
#           'info'     = SKY BLUE ,
#           'warning'  = ORANGE ,
#           'danger'   = RED        &   '' = BLUE
```

```
# Numeric Widget - FloatProgess
widgets.FloatProgress(
    value = 3.3,
    min = 0, max = 10.0, step = 0.1, description = 'Loading',
    bar_style = '',
    orientation = 'horizontal'
)                                   # Loading: [▨▨    ]

# Numeric widgets - BoundedIntText
widgets.BoundedIntText(
    value = 7, min = 0, max = 10, step = 1,
    description = 'Text :',
    disabled = False
) # Text: [7      ]

# Numeric widget - BoundedFloatText
widgets.BoundedFloatText(
    value = 3.3, min = 0, max = 10.0, step = 0.1,
    description = 'Text :', disabled = False
)                        # Text: [3.3      ]

# Numeric Widget - IntText
widgets.IntText(
    value = 7, description = 'Any :',
    disabled = False
)                        #      Any: [7      ]

# Numeric widgets - FloatText
widgets.FloatText(
    value = 2.6, step = 0.1, description = 'Any :',
    disabled = False
)                        #     Any: [2.6      ]

# Boolean widgets -> There are three widgets that are designed
  to display a boolean value.
# ToggleButton :
widgets.ToggleButton(
    value = False, description = 'click me :', disabled = False,
    button_style = '', tooltip = 'Description', icon = 'check'
)
            #    [✓ click me]
```

```
# checkbox
widgets.Checkbox(
    value=True, description='Check me', disabled=False
)                                  # ☑ Check me
# Valid -> The valid widget provides a read only indicator
widgets.Valid(
    value=True, description='Valid!'
)                                  # Valid! ✓
# widget.valid(
    value=False, description='Valid'
)  # Valid ✗      invalid
```

Selection Widgets

"""We can specify the enumeration of selectable options by passing a list. We can specify the enumeration as a dictionary in which case the keys will be used as the item displayed in the list and the corresponding value will be used when an item is selected."""

```
# Selection widgets - Dropdown
widgets.Dropdown(
    options={'1','2','3'}, value='2', description='Number',
    disabled=False
)                                  # Number [2 ▾]

widgets.Dropdown(
    options={'one':1, 'Two':2, 'Three':3}, value=2,
    description='Number', disabled=False
)                                  # Number [Two ▾]
```

```
# Selection Widget - Radio Buttons
widget.RadioButtons(
    options=['BBSR','KUR','CHE'], value='KUR',
    description='Location:', disabled=False
)                                  # Location : ○ BBSR ● KUR ○ CHE
```

```
# Selection Widget - Select
widgets.Select(
    options=['BBSR','KUR','CHE'], value='KUR', rows=5,
    description='Location:', disabled=False
)                                  # Location : BBSR / KUR / CHE
```

```
# Selection Widgets - SelectionSlider
widgets.SelectionSlider(
    options=[1,2,3,4,5,6,7,8,9,10], value=4, description='cm',
    disabled=False, continous_update=False, orientation='horizontal',
    readout=True                    #   cm ——●—— 4
)
# Selection Widgets - SelectionRangeSlider
""" The value, index & label keys are 2tuple of the min & max
    values selected. The option must be non-empty. """
widgets.ToggleButtons(
    options=['LOW','MEDIUM','HIGH'], value='LOW', description='MARKS:',
    disabled=False, button_style='', tooltips=['1-3','4-7','8-10'],
    #, icon=['check']*3          #   Marks: | LOW | MEDIUM | HIGH |
)
# Selection Widgets - SelectMultiple
widgets.SelectMultiple(
    options=['Apple','Banana','Grapes'], value=['Banana'],
    rows=2, description='Fruits', disabled=False
)                                #   Fruits: Apple / Banana
# String Widgets
""" These widgets can be used to display a string value. Text
    and TextArea widgets accepted input. """
# String widgets - Text
widgets.Text(
    value='hello world', # value=input()
    placeholder='TypeSomething', description='String', disabled=False
)                                #   String [hello world]
# String widgets - TextArea <we can exted area fields>
widgets.Textarea(
    value='Hello world', # value=input()
    placeholder='Typesomething', description='String', disabled
                                 =False
)                                #   String [Hello World]
# String widgets - Label
widgets.HBox([widgets.Label(value='The $m$ in $E=mc^2$:'),
              widgets.FloatSlider()])
                                 # The m in E= ——●—— 0.00
                                        mc²
```

```
# String Widgets - HTML
widgets.HTML(
    value = "Hello <b>World</b>", placeholder = 'Some HTML',
    description = 'Some HTML'
)                                    #  Some HTML  Hello World

# String Widget - HTML Math
widget.HTMLMath(
    value = r'Some math and <i>HTML</i>: \(x^2\) and $$\frac
        {x+1}{x-1}$$ ',
    placeholder = 'Some HTML', description = 'Some HTML'
)                                    #  some HTML some math and HTML: x² and (x+1)/(x-1)

# String widgets - Image
file = open('C:\\Users\KIIT\Desktop\item_none.jpeg', 'rb')
image = file.read()
widgets.Image(
    value = image, format = 'jpg', width = 250, height = 100
)                                    #

# String Widget - Button
widget.Button(
    description = 'click me', disabled = False, button_style = '',
    tooltip = 'click me', icon = 'check'
)                                    #  [✓ click me]
```

Widget Styling & Layout :→

"""To all widgets attribute which exposes non-layout related attribute like button, color and font size - layout is generally to all widgets and containers of widgets. style offers tools specific to each type of widget.
"""

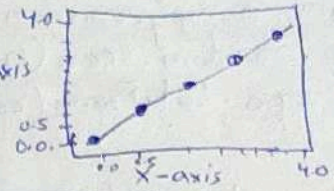
```
→ import ipywidgets as widgets
from IPython.display import display
w = widgets.IntSlider()
display(w)                           #  (before) 0 ——●—— 0

# layout
w.layout.margin = 'auto'
w.layout.height = '75px'
display(w)                           #  after 0 ——●—— 0 (interlinked)

x = widget.IntSlider(value = 15, description = 'New Slider:')
# x.layout.height = '75px'
display(x)                           #  New Slider: ——●—— 15

x.layout = w.layout
widgets.Button(description = 'Ordinary Button', button_style = '')   #  [Ordinary Button]
# 'success' - LIGHTGREEN, 'info' - SKYBLUE, 'warning' - Orange, 'danger' - Red, '' - White
```

```
b1 = widgets.Button(description='Custom colour')
b1.style.button_color = 'darkgreen'   # Custom color
b1
b1.style.keys   # [ - - ]
b2 = widgets.Button(description='Ordinary Button:', button_style='')
b2.style = b1.style                   # Ordinary Button
b2
s1 = widgets.IntSlider(description='My handle')
s1.style.handle_color = 'blue'
s1                                    # My handle •——— 0
s1.style.keys                         # [ - - ]
```

↳ # Examples of what widgets can do:

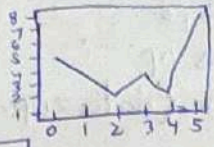
```
"""GUI Examples:
    we need to install some libraries
    we need to import : Numpy, Matplotlib, SciPy, Pandas
"""
# numpy
import numpy as np
dir(np)
a = np.array([1,2,3])          # array([1,2,3])
a
a = np.array([[1,2],[3,4]])
a                              # array([1,2],
                               #       [3,4])
a = np.arange(15).reshape(3,5)
print(a.shape)                 # (3,5)
print(a.ndim)                  # 2
print(a.dtype)                 # int32
print(a.itemsize)              # 4
print(a.size)                  # 15
np.sin(45)                     # 0.85090 ... 184
np.sin(0)                      # 0.0
np.cos(0)                      # 1.0
np.cos(45)                     # 0.5253 ... 297
np.pi                          # 3.1415 ... 793
np.sin(np.pi/2)                # 1.0
np.cos(np.pi/3)                # 0.50000 ... 001
np.sin(np.pi/6)                # 0.499 ... 994
```

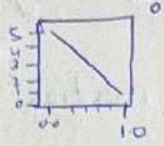
```
# matplotlib
% matplotlib inline
import matplotlib.pyplot as plt
x = range(5)                    # x = range(2,10,2)
plt.plot(x, '-bo')
plt.xlabel('x-axis')
plt.ylabel('Y-axis')
```



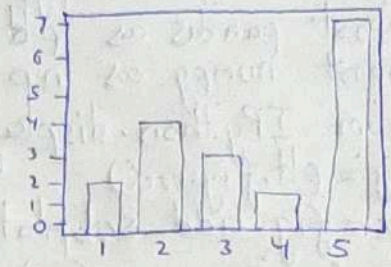

```
# Text(0, 0.5, 'Y-axis')
```

```
import matplotlib.pyplot as plt
plt.plot([5,3,2,2,1,8]
plt.plot([5,0])
```

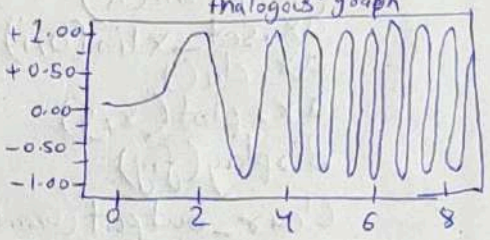




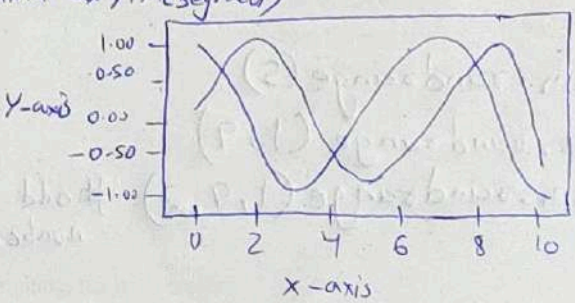
```
import matplotlib.pyplot as plt
x = [1,2,3,4,5]
y = [2,4,3,1,7]
plt.bar(x, y, label = 'first bars')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Graph')
plt.legend()  # for label
plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 3*np.pi, 500)
plt.plot(x, np.sin(x**2))
plt.title('Analogous Graph')
```




```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100) #(angle min&max, linesegment)
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np
import math
math.pi                 # 3.141-------793
dir(math)               #
dir(np)                 #
dir(plt)                #
```

Pandas :—

```
import pandas as pb
data = { 'Names': ['A', 'B', 'C'],
         'Location': ['BBSR', 'KUR', 'CHE'],
         'Age': [21, 23, 25]
       }
data_pandas = pd.DataFrame(data)
data_pandas
```

	Names	Location	Age
0	A	BBSR	21
1	B	KUR	23
2	C	CHE	25

→
```
import pandas as pd
import numpy as np
from numpy.random import randn
np.random.seed(10)
df = pd.DataFrame(randn(5,4),['A','B','C','D','E'],['1','2','3','4'])
df
```

#

	1	2	3	4
A				
B				
C				
D				
E				

→
```
import pandas as pd
import numpy as np
df = pd.DataFrame(data=np.array([[1,2,3],[4,5,6]]), dtype=int), columns=['A','B','C'])
df
# if no columns are mention #
```

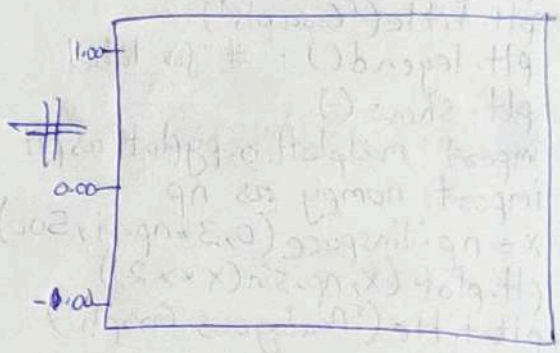
#

	A	B	C
0	1	2	3
1	4	5	6

	0	1	2
0	1	2	3
1	4	5	6

→
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from IPython.display import display, clear_output
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
for i in range(50):
    x = np.arange(0,i,0.1)
    y = np.sin(x)
    ax.set_xlim(0,i)
    ax.cla()
    ax.plot(x,y)
    display(fig)
    clear_output(wait=True)
    plt.pause(0.5)
```

#



→
```
random.randrange(5)
random.randrange(1,9)
random.randrange(1,9,2) #odd number
```

```
import random
a = random.randint(1,10)
print(a)

for i in rang(5):
    print(random.randint(1,6))
```

→
```
import random
from random import seed, random
seed(100)  # random float number
for _ in range(5):
    print(random())
```
#

→
```
import random
s = [1,2,3,4,5]
print(random.choice(s))
```

→
```
import numpy as np
a = np.random.randint(low=1, high=10)
print(a)
```
#
```
arr = np.random.randint(low=1, high=10, size=(6,))
arr
```
#

```
import random
a = [7, 8, 10, 12]
random.shuffle(a)
print(a)
```
#