Comments in Python :

Comments in Python are the lines in the code that are ignored by the interpreter during the execution of the program. Comments enhance the readability of the code and help the programmers to understand the code very carefully. we also comments code statements too.

types of comments : there are various types of comments in python programming. These are giving below.

1. single_line
2. multiple_line
3. string_literal
4. Doc_string

Let's see them one by one with examples, explanations and syntax.

1. Single line Comments : Python single-line comment starts with the hashtag symbol (#) with no white spaces and lasts till the end of the line. It’s useful for short explanations for variables, function declarations.

SYNTAX: # comments EXAMPLE:

```
print("line_1")
print("line_2")
# print("line_3")
print("line_4")
```

```
line_1
line_2
line_4
```

2. Multiline Comments : Python does not provide the option for multiline. comments. However, there are different ways through which we can write multiline comments. like hashtag(#) or using string literals. we can use hashtag in starting of multiple lines that we want to comments.

SYNTAX: # comments_1 # comments_2 # comments_3 # .... EXAMPLE:

```
print("line_1")
print("line_2")
# print("line_3")
# print("line_4")
# print("line_5")
print("line_6")
# print("line_7")
```

```
line_1
line_2
line_6
```

3. String Literals : in this method, we use triple quotes('''comment''') or ("""comment""") to comments single or multiple line

SYNTAX: code...statements '''comments_1 comments_2 comments_3 ....''' code...statements EXAMPLE:

```
print("line_1")
print("line_2")
'''print("line_3")
print("line_4")
print("line_5")'''
print("line_6")
print("line_7")
```

```
line_1
line_2
line_6
line_7
```

SYNTAX: code...statements """ comments_1 comments_2 comments_3 ....""" code...statements EXAMPLE:

```
print("line_1")
print("line_2")
"""print("line_3")
print("line_4")
print("line_5")
print("line_6")"""
print("line_7")
```

```
line_1
line_2
line_7
```

4. Doc_string Comments : Python docstring is the string literals with triple quotes that appeared right after the function. It is used to associate documentation that has been written with Python

modules, functions, classes & methods and describe what they do. it is done using "__doc__attribute".

SYNTAX : def function_name(args): """there is Doc_string""" function defination and result

print(function_name.__doc__) ------> Print the docstring of multiply function EXAMPLE :

```
def multiply(a, b):
    """Multiplies the value of a and b"""
    return a*b

print(multiply.__doc__)
multiply(2,3)
```

```
Multiplies the value of a and b

6
```

modules in python :

- let's see basics of modules in python than we'll dive deep about it later.
- modules are files that contain built-in functions, classes and variables.
- there are three type of modules mostly

1. built-in(internal) : all the modules exists in python standard library
2. external : this is the we add from other sources not exists in python standard libraries.
3. user-define : the modules user create for own purpose to improve own creativity.

- let's see example of external module playsounds to play music.

```
pip install playsound
```

```
Requirement already satisfied: playsound in c:\users\rajen\anaconda3\
lib\site-packages (1.3.0)
Note: you may need to restart the kernel to use updated packages.
```

```
from playsound import playsound

playsound('E:\\PROJECT_MEDIA\\Aathma Rama Ananda Ramana_-_Brodha_V-
No_English_Only_Hindi_Version-Spirituality-ॐ_.mp3')
```

NOTE : always exchange '' and '/' to '\' in path to execute the command coz single slash denoted to black_space character.

- let's try to play video

```
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\rajen\
anaconda3\lib\site-packages (4.9.0.80)Note: you may need to restart
the kernel to use updated packages.
```

```
Requirement already satisfied: numpy>=1.21.2 in c:\users\rajen\
anaconda3\lib\site-packages (from opencv-python) (1.26.3)

pip install numpy

Requirement already satisfied: numpy in c:\users\rajen\anaconda3\lib\
site-packages (1.26.3)
Note: you may need to restart the kernel to use updated packages.

pip install ffpyplayer

Requirement already satisfied: ffpyplayer in c:\users\rajen\anaconda3\
lib\site-packages (4.5.1)
Note: you may need to restart the kernel to use updated packages.

import cv2
import numpy as np
#ffpyplayer for playing audio
from ffpyplayer.player import MediaPlayer


video_path="E:\\PROJECT_MEDIA\\
Best_romantic_Animated_status_video_YouTube.mp4"
def PlayVideo(video_path):
    video=cv2.VideoCapture(video_path)
    player = MediaPlayer(video_path)
    while True:
        grabbed, frame=video.read()
        audio_frame, val = player.get_frame()
        if not grabbed:
            print("End of video")
            break
        if cv2.waitKey(28) & 0xFF == ord("q"):
            break
        cv2.imshow("Video", frame)
        if val != 'eof' and audio_frame is not None:
            #audio
            img, t = audio_frame
    video.release()
    cv2.destroyAllWindows()
PlayVideo(video_path)

End of video
```

let's see OS built-in module.

```
import os
print(os.listdir)

<built-in function listdir>
```

Let's see "print()" function...

print() function :

The print() function prints the specified message to the screen, or other standard output device. SYNTAX : print("this is a section to write...") there are various way to use print() function.

- default way :

```
print("hello world")
```

```
hello world
```

- we can use it as basic calculator

```
print(2+3)
```

```
5
```

- notebook cell also can work as print function if you only execute it with result statements

```
12 + 13
```

```
25
```

- to print result with massage.

1. default way.

```
a = 5
b = 3
c = a + b

print("sum = ",c)
```

```
sum =  8
```

```
a = 5
b = 3
c = a + b

print("sum of ",a," and ",b," is ",c)    # default way
```

```
sum of  5  and  3  is  8
```

- with f-string : in this method you can print {values} in strings with using 'f' in starting of string in print() funtion..

```
a = 5
b = 3
c = a + b

print(f"sum of {a} and {b} is {c}")
```

```
sum of 5 and 3 is 8
```

- format method.

```
name1 = 'ram'
name2 = 'syam'
age = 25
place = 'taj-mahal'

print("{} and {} are friends and both are {} years old and went to {}
in vecetion.".format(name1,name2,age,place))
```

```
ram and syam are friends and both are 25 years old and went to taj-
mahal in vecetion.
```

```
name1 = 'ram'
name2 = 'syam'
age = 25
place = 'taj-mahal'

print("my favorite place {2}, and i'm {0} with {1} years
old.".format(name1,age,place))
```

```
my favorite place taj-mahal, and i'm ram with 25 years old.
```

```
n1 = 'ram'
n2 = 'syam'
txt = "{} and {} are good friends."

print(txt.format(n1,n2))
```

```
ram and syam are good friends.
```

- using placeholder. % with d,i,c and f

```
a = 5.0
b = 3.0
c = a + b
sum = 'sum of %f and %f is'%(a,b)
print(sum,c)
```

```
sum of 5.000000 and 3.000000 is 8.0
```

- concate massage.

```
age = '34'
msg = 'my name is john, and I am ' + age

print(msg)
```

```
my name is john, and I am 34
```

```
s1 = 'i a raj'
s2 = 'am 25 years old'

print(s1+s2)

i a rajam 25 years old
```

- other way.

```
x = 'I'
y = 'am'
z = 'ram'
print(x + y + z)

Iamram

x = 'I'
y = 'am'
z = 'ram'
print(x,y,z)

I am ram
```

type() function :

This is built0in function to print the data type of given variable or value. it take only one argument. This code assigns variable 'x' different values of various data types in Python. It covers string, integer, float, complex, list, tuple, range, dictionary, set, frozenset, boolean, bytes, bytearray, memoryview, and the special value 'None' successively. Each assignment replaces the previous value, making 'x' take on the data type and value of the most recent assignment.

SYNTAX: type(x)

```
a = 12

type(a)

int

b = True
print(type(b))

<class 'bool'>

c = 2 + 3j
print(type(c))

<class 'complex'>
```

Data types:

There are different types of data types in Python. Some built-in Python data types are set, frozenset

- Numeric data types: int, float, complex, long
- String data types: str
- Sequence types: list, tuple, range,
- Binary types: bytes, bytearray, memoryviewe
- Mapping data type: dict
- Boolean type: bool
- Set data types: set, frozenset
- None type : None

Let'a see them by examples.

- Numeric data type:

Python numeric data type is used to hold numeric values like;

1. int-holds signed integers of non-limited length.
2. long-holds long integers(exists in Python 2.x, deprecated in Python 3.x).
3. float-holds floating precision numbers and it’s accurate up to 15 decimal places.
4. complex-holds complex numbers.

1. Integer data type: an integer is a whole number that can be positive, negative, or zero. Integers do not include decimals or fractions. Example: Here are some examples of integers: -5, 0, 1, 5, 8, 97, 3,043. Here are some examples of numbers that are not integers: -1.43, 1 3/4, 3.14, .09, 5,643.1. Integers are also called "round numbers" or "whole numbers". In computer science, an integer is a value that can range from 0 to 65,535.

```
a = 10
b = -5

print("a = ",a)
print("data type of a = ",type(a))

print("b = ",b)
print("data type of b = ",type(b))
```

```
a =  10
data type of a =  <class 'int'>
b =  -5
data type of b =  <class 'int'>
```

2. float data type: A floating point number, is a positive or negative whole number with a decimal point. For example, 5.5, 0.25, and -103.342 are all floating point numbers, while 91, and 0 are not. Floating point numbers get their name from the way the decimal point can "float" to any position necessary.

```
a = 10.6
b = -5.13

print("a = ",a)
print("data type of a = ",type(a))

print("b = ",b)
print("data type of b = ",type(b))
```

```
a =  10.6
data type of a =  <class 'float'>
b =  -5.13
data type of b =  <class 'float'>
```

3. complex number: In mathematics, a complex number is an element of a number system that extends the real numbers with a specific element denoted i, called the imaginary unit and satisfying the equation; every complex number can be expressed in the form, where a and b are real numbers SYNTAX: complex numbers are: a + bi, a - bi, bi.

```
a = 3 + 4j
b = 3j

print("a = ",a)
print(type(a))

print("b = ",b)
print(type(b))
```

```
a =  (3+4j)
<class 'complex'>
b =  3j
<class 'complex'>
```

- some basic math with complex numbers.

```
#  complex number operations
a = 3 + 4j
b = 5 + 9j

c1 = a + b
c2 = a - b
c3 = a * b
c4 = a / b
c5 = a ** b

print("a + b",c1)
print(type(c1))
print("a - b",c2)
print(type(c2))
print("a * b",c3)
print(type(c3))
print("a / b",c4)
print(type(c4))
print("a ** b",c5)
print(type(c5))
```

```
a + b (8+13j)
<class 'complex'>
a - b (-2-5j)
<class 'complex'>
a * b (-21+47j)
<class 'complex'>
a / b (0.4811320754716981-0.06603773584905659j)
<class 'complex'>
a ** b (0.714704513387209+0.19923326288380847j)
<class 'complex'>
```

4. long data type : The long data type stores integers like int , but gives a wider range of values at the cost of taking more memory. Long stores at least 32 bits, giving it a range of -2,147,483,648 to 2,147,483,647. Alternatively, use unsigned long for a range of 0 to 4,294,967,295.

RANDOM NUMBER : to get random number in python we use random module

EXAMPLE: let's see some basic code will dive deep later.

```
import random
a = [1,2,3,4,5]
print(random.choice(a))
# check it again
```

```
5
```

```
import random
num = random.random()
print(num)
```

```
0.5419794621608753
```

- Strings : The string is a sequence of characters. Python supports Unicode characters. Generally, strings are represented by either single or double-quotes.
- string with single qoute

```
a = 'i am string'
print("a = ",a)
print("data type of a = ",type(a))
```

```
a =  i am string
data type of a =  <class 'str'>
```

- string with double qoute

```
a = "i am string"
print("a = ",a)
print("data type of a = ",type(a))

a =  i am string
data type of a =  <class 'str'>
```

- string with multiple line with single quote

```
a = '''i am
a string
with
multiple line '''
print("a = ",a)
print("data type of a = ",type(a))

a =  i am
a string
with
multiple line
data type of a =  <class 'str'>
```

- string with multiple line with double quote

```
a = """i am
a string
with
multiple line"""
print("a = ",a)
print("data type of a = ",type(a))
```

- string types according to quotes used inside or outside of strings.

```
a = '"ram" and "shyam" is my friend.'
b = "mohan is meena's friend"
c = '''"ram" is mohan's brother and a good student'''
d = """"ram" is mohan's brother and a good student"""

print('strings are :')
print("1. ",a)
print(type(a))
print("2. ",b)
print(type(b))
print("3. ",c)
print(type(c))
print("4. ",d)
print(type(d))

strings are :
1.  "ram" and "shyam" is my friend.
<class 'str'>
```

```
2.  mohan is meena's friend
<class 'str'>
3.  "ram" is mohan's brother and a good student
<class 'str'>
4.  "ram" is mohan's brother and a good student
<class 'str'>
```

NOTE :We'll see string functions later.

- Sequence data type :

NOTE :We'll see these data types later.

1. list: A list is a data structure in Python that is a mutable, or changeable, ordered sequence of elements. Each element or value that is inside of a list is called an item. Just as strings are defined as characters between quotes, lists are defined by having values between square brackets [ ] .

```
x = [10, 11, 12, 13, 14, 15]

print(x)
print(type(x))

[10, 11, 12, 13, 14, 15]
<class 'list'>
```

2. tuple: Python tuples are a type of data structure that is very similar to lists. The main difference between the two is that tuples are immutable, meaning they cannot be changed once they are created. This makes them ideal for storing data that should not be modified, such as database records.

```
x = (10, 11, 12, 13, 14, 15)

print(x)
print(type(x))

(10, 11, 12, 13, 14, 15)
<class 'tuple'>
```

3. range: range is a function that returns a sequence of numbers. By default, range returns a sequence that begins at 0 and increments in steps of 1. The range function only works with integers.

```
x = range(6)
for n in x:
    print(n)

0
1
2
```

```
3
4
5
```

- binary data type:

1. byte: byte is a digital information unit that typically consists of 8 bits each of which consists of either a 0 or 1. A byte is a computer architecture term for memory storage that encodes a single character of text in a computer.

```
x = b"Hello"

print(x)
print(type(x))

b'Hello'
<class 'bytes'>
```

2. bytearray: The bytearray type is a mutable sequence of integers in the range between 0 and 255. It allows you to work directly with binary data. It can be used to work with low-level data such as that inside of images or arriving directly from the network. Bytearray type inherits methods from both list and str types.

```
x = bytearray(5)

print(x)
print(type(x))

bytearray(b'\x00\x00\x00\x00\x00')
<class 'bytearray'>
```

3. memoryview: In Python, a memory view is a secure technique to expose the buffer protocol. It allows you to access an object's internal buffers by constructing a memory view object. The memoryview() method in Python returns a memoryview object based on the bytes or bytearray parameter.

```
x = memoryview(bytes(5))

print(x)
print(type(x))

<memory at 0x0000020AE693E380>
<class 'memoryview'>
```

- mapping data type: The mapping data type in Python includes dictionaries. Dictionaries are used to store key-value pairs, where each key is associated with a value.

1. dictionary: Dictionaries are Python's implementation of a data structure that is more generally known as an associative array. A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value.

```
x = {"name" : "John", "age" : 36}

print(x)
print(type(x))
```

```
{'name': 'John', 'age': 36}
<class 'dict'>
```

NOTE :We'll see these data types later.

- bool data type: The python data type bool is used to store two values i.e True and False . Bool is used to test whether the result of an expression is true or false. Let's see them by examples,

```
a = True  #boolean
b = False
print(a)
print(type(a))
print(b)
print(type(b))
```

```
True
<class 'bool'>
False
<class 'bool'>
```

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

```
True
False
False
```

```
print(bool("Hello"))
print(bool(15))
```

```
True
True
```

- set data type:

1. set: A Set in Python is a collection of data type that is unordered and unindexed. It is mutable, i.e., we can remove or add elements to it. Set in python is similar to mathematical sets, and operations like intersection, union, symmetric difference, and more can be applied.

```
x = {"apple", "banana", "cherry"}

print(x)
print(type(x))
```

```
{'cherry', 'banana', 'apple'}
<class 'set'>
```

NOTE :We'll see these data types later.`

2. frozen set: An immutable version of a Python set object is a frozen set. While parts of a set can be changed at any moment, elements of a frozen set don't change after they've been created. As a result, frozen sets can be used as Dictionary keys or as elements of other sets.

```
x = frozenset({"apple", "banana", "cherry"})

print(x)
print(type(x))

frozenset({'cherry', 'banana', 'apple'})
<class 'frozenset'>
```

Quiz: difference between set and frozen set.

- None data type : In Python, None is a special keyword that represents the absence of a value or a null value. It is an object of its own datatype, the NoneType. We can assign None to any variable, but you can not create other NoneType objects. It can be used in a variety of situations, such as when you want to assign a value to a variable but don't have a value yet, when you want to pass a value to a function but don't have a value to pass, or when you want to return a value from a function but don't have a value to return.

```
a = None
print(a)
print(type(a))

None
<class 'NoneType'>
```

Variable :

A Python variable is a reserved memory location to store values. In other words, a variable in a python program gives data to the computer for processing. Every value in Python has a datatype. Different data types in Python are Numbers, List, Tuple, Strings, Dictionary, etc.

Rules for Python variables:

- A Python variable name must start with a letter or the underscore character.
- A Python variable name cannot start with a number.
- A Python variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- Variable in Python names are case-sensitive (name, Name, and NAME are three different variables).
- The reserved words(keywords) in Python cannot be used to name the variable in Python.

Notes:

- The value stored in a variable can be changed during program execution.
- A Variables in Python is only a name given to a memory location, all the operations done on the variable effects that memory location.

[Do] best way to naming variable is to use camel case ex. 'MyName' see 'M' and 'N' and only we can use '_' between characters

```
a = 10
ram = 5
MyName = 'raj'
my_name = 'raj'
_my_name = 'raj'
my_name2 = 'raj'
MYNAME = 'raj'
```

[Don't] no space, no special character

- multiple values in multiple variables

```
a,b,c = 10,11,12

print("a,b,c = 10,11,12")
print("a = ",a)
print("c = ",b)
print("c = ",c)

a,b,c = 10,11,12
a =  10
c =  11
c =  12
```

- put single values in multiple variables

```
x = y = z = 'ram'

print("x = y = z = 'ram'")
print("x = ",x)
print("y = ",y)
print("z = ",z)

x = y = z = 'ram'
x =  ram
y =  ram
z =  ram
```

- if you have collection of value i.e. list, tuple, set than you can extract their values in different-different variable.

```
cars = ['oudi', 'suzuki', 'benz', 'toyota']
a, b, c, d = cars
print("list = ",cars)
print("a = ",a)
print("b = ",b)
print("c = ",c)
print("d = ",d)

list =  ['oudi', 'suzuki', 'benz', 'toyota']
a =  oudi
b =  suzuki
c =  benz
d =  toyota
```

Operators in python :

In Python programming, Operators in general are used to perform operations on values and variables. These are standard symbols used for the purpose of logical and arithmetic operations. In this article, we will look into different types of Python operators.ed.

- OPERATORS: These are the special symbols. Eg- + , * , /, etc.
- OPERAND: It is the value on which the operator is applied.

types of operators:

- Arithmetic operators
- Assignment operators
- Comparison/ Relational operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

1. Arithmatic Operators : Python Arithmetic operators are used to perform basic mathematical operations like addition, subtraction, multiplication, and division. In Python 3.x the result of division is a floating-point while in Python 2.x division of 2 integers was an integer. To obtain an integer result in Python 3.x floored (// integer) is used.

it is used for basic math problems.

- variable with values

```
a = 5
b = 3
c = 2.4
d = 3.1
e = True
f = False
```

- addition/ sum

```
a = 5
b = 3
```

```
c = 2.4
d = 3.1

sum1 = a + b
sum2 = c + d
print(f"sum of {a} and {b} :",sum1)
print(f"sum of {c} and {d} :",sum2)
```

```
sum of 5 and 3 : 8
sum of 2.4 and 3.1 : 5.5
```

- subtraction

```
a = 5
b = 3
c = 2.4
d = 3.1

sub1 = a - b
sub2 = c - d
print(f"sub of {a} and {b} :",sub1)
print(f"sub of {c} and {d} :",sub2)
```

```
sub of 5 and 3 : 2
sub of 2.4 and 3.1 : -0.7000000000000002
```

- multiplication

```
a = 5
b = 3
c = 2.4
d = 3.1

mul1 = a * b
mul2 = c * d
print(f"mul of {a} and {b} :",mul1)
print(f"mul of {c} and {d} :",mul2)
```

```
mul of 5 and 3 : 15
mul of 2.4 and 3.1 : 7.4399999999999995
```

- division

There are two types of division operators:

1. Float division
2. Floor division

Float division: The quotient returned by this operator is always a float number, no matter if two numbers are integers. It is just a normal math division.

Example: The code performs division operations and prints the results. It demonstrates that both integer and floating-point divisions return accurate results. For example, '10/2' results in '5.0', and '-10/2' results in '-5.0'.

```
print(5/5)
print(10/2)
print(-10/2)
print(20.0/2)
```

```
1.0
5.0
-5.0
10.0
```

```
a = 5
b = 3
c = 2.4
d = 3.1

div1 = b / a
div2 = d / c
print(f"{b} divided by {a} :",div1)
print(f"{d}divided by {c} :",div2)
```

```
3 divided by 5 : 0.6
3.1divided by 2.4 : 1.2916666666666667
```

Integer division(Floor division): The quotient returned by this operator is dependent on the argument being passed. If any of the numbers is float, it returns output in float. It is also known as Floor division because, if any number is negative, then the output will be floored.

In other words, Floor division is a normal division operation except that it returns the largest possible integer. This integer is either less than or equal to the normal division result. Floor function is mathematically denoted by this [] symbol.

Example: The code demonstrates integer (floor) division operations using the '//' operator. It provides results as follows: '10//3' equals '3', '-5//2' equals '-3', '5.0//2' equals '2.0', and '-5.0//2' equals '-3.0'. Integer division returns the largest integer less than or equal to the division result.

```
print(10//3)
print (-5//2)
print (5.0//2)
print (-5.0//2)
```

```
3
-3
2.0
-3.0
```

```
a = 5
b = 3
```

```
c = 2.4
d = 3.1

div1 = b // a
div2 = d // c
print(f"floor division of {b} and {a} :",div1)
print(f"floor division of {d} and {c} :",div2)
```

```
floor division of 3 and 5 : 0
floor division of 3.1 and 2.4 : 1.0
```

- exponatial/ power

```
a = 5
b = 3
c = 2.4
d = 3.1

ex1 = a ** b
ex2 = c ** d
print("a to the power b :",ex1)
print("c to the power d :",ex2)
```

```
a to the power b : 125
c to the power d : 15.088805115741808
```

- reminder/modulus

```
a = 5
b = 3
c = 2.4
d = 3.1

rem1 = a % b
rem2 = c % d
print(f"mod of {a} and {b} :",rem1)
print(f"mod of {c} and {d} :",rem2)
```

```
mod of 5 and 3 : 2
mod of 2.4 and 3.1 : 2.4
```

2. Assignment Operators : Python Assignment operators are used to assign values to the variables. who hold '=' and used to assign value to exponant(variable).

```
x = 4
print("x =", x)

x += 3  #x = x + 3
print("(x += 3)=",x)
```

```
x -= 3  #x = x - 3
print("(x -= 3)=",x)

x *= 3  #x = x * 3
print("(x *= 3)=",x)

x /= 3  #x = x / 3
print("(x /= 3)=",x)

x %= 3  #x = x % 3
print("(x %= 3)=",x)

x /= 3  #x = x // 3
print("(x /= 3)=",x)

x **= 3  #x = x ** 3
print("(x **= 3)=",x)
```

```
x = 4
(x += 3)= 7
(x -= 3)= 4
(x *= 3)= 12
(x /= 3)= 4.0
(x %= 3)= 1.0
(x /= 3)= 0.3333333333333333
(x **= 3)= 0.037037037037037035
```

3. Comparison/relational operators: Comparison operators are used to compare two values. Comparison operators compare values and return true or false. They are also known as relational operators.

```
x = 5
y = 8

print("(x == y) = ",x == y)
print("(x != y) = ",x != y)
print("(x < y) = ",x < y)
print("(x <= y) = ",x <= y)
print("(x > y) = ",x > y)
print("(x >= y) = ",x >= y)
```

```
(x == y) =  False
(x != y) =  True
(x < y) =  True
(x <= y) =  True
(x > y) =  False
(x >= y) =  False
```

4. Logical operator: Returns True if both statements are true rather than false.

```
bool1 = True
bool2 = False

print("value of 'bool1 and bool2' is : ",bool1 and bool2)
print("value of 'bool1 or bool2' is : ",bool1 or bool2)
print("value of 'not bool2' is : ",not bool2)
```

```
value of 'bool1 and bool2' is :  False
value of 'bool1 or bool2' is :  True
value of 'not bool2' is :  True
```

5. Identity Operators: identity operators are the operators who used to compare the object that they are equal to or not.

```
x = 10
y = 15
z = 10

print("x = 10, y = 15, z = 10")
print("x is y : ", x is y)
print("x is not y : ", x is not y)
print("x is z : ", x is z)
```

```
x = 10, y = 15, z = 10
x is y :  False
x is not y :  True
x is z :  True
```

6. membership operators: it is used to test that specific value in give sequence or not.

```
x = [1, 2, 3, 4, 5]

print("1 in x = ",1 in x)
print("1 not in x = ",1 not in x)
```

```
1 in x =  True
1 not in x =  False
```

7. Bitwise operator: Bitwise operators are used to compare (binary) numbers.

Note: Python bitwise operators work only on integers.

- Bitwise AND (&) operator: Returns 1 if both the bits are 1 else 0.

```
print(6 & 3)
```

```
2
```

- Bitwise OR(|) operator: Returns 1 if either of the bit is 1 else 0.

```
print(6 | 3)
```

```
7
```

- Bitwise Not(~) operator: inverts individual bits

```
print(~3)

-4
```

- Bitwise XOR(^) operator

```
print(10 ^ 4)

14
```

- Bitwise right shift(>>) operator: The left operand’s value is moved toward right by the number of bits specified by the right operand.

```
print(8 >> 2)

2
```

- Bitwise left shift(<<) operator:The left operand’s value is moved toward left by the number of bits specified by the right operand.

```
print(3 << 2)

12
```

Operator Precedence

Operator precedence describes the order in which operations are performed. (read online)

operator and operator preceding in w3school and geeks for geeks portal match

Operator Precedence and Associativity in Python

operators have different levels of precedence, which determine the order in which they are evaluated. When multiple operators are present in an expression, the ones with higher precedence are evaluated first. In the case of operators with the same precedence, their associativity comes into play, determining the order of evaluation.

- Precedence of Python Operators: This is used in an expression with more than one operator with different precedence to determine which operation to perform first.

EXAMPLE:

- Precedence of '+' & '*'

```
expr = 10 + 20 * 30
print(expr)

610
```

EXPLAINATION: 10 + 20 * 30 is calculated as 10 + (20 * 30) and not as (10 + 20) * 30

- Precedence of Logical Operators in Python: In the given code, the 'if' block is executed even if the age is 0. Because the precedence of logical 'and' is greater than the logical 'or'. An expression is a collection of numbers, variables, operations, and built-in or user-defined function calls. The Python interpreter can evaluate a valid expression.
- Precedence of 'or' & 'and'

```
name = "Alex"
age = 0

if name == "Alex" or name == "John" and age >= 2:
      print("Hello! Welcome.")
else:
      print("Good Bye!!")

Hello! Welcome.
```

Hence, To run the 'else' block we can use parenthesis( ) as their precedence is highest among all the operators.

```
name = "Alex"
age = 0

if (name == "Alex" or name == "John") and age >= 2:
      print("Hello! Welcome.")
else:
      print("Good Bye!!")

Good Bye!!
```

- Associativity of Python Operators: If an expression contains two or more operators with the same precedence then Operator Associativity is used to determine. It can either be Left to Right or from Right to Left.

EXAMPLE: In this code, '' *and '/' have the same precedence and their associativity is Left to Right, so the expression "100 / 10* 10" is treated as "(100 / 10) * 10".

```
print(100 / 10 * 10)
print(5 - 2 + 3)
print(5 - (2 + 3))
print(2 ** 3 ** 2)
```

```
100.0
6
0
512
```

Operators Precedence and Associativity in Python: Operators Precedence and Associativity are two main characteristics of operators that determine the evaluation order of sub-expressions in the absence of brackets.

EXAMPLE:

100 + 200 / 10 - 3 * 10

```python
expression = 100 + 200 / 10 - 3 * 10
print(expression)
```

```
90.0
```

- Non-associative Operators: In Python, most operators have associativity, which means they are evaluated from left to right or right to left when they have the same precedence. However, there are a few operators that are non-associative, meaning they cannot be chained together.

NOTE:The following table shows the precedence of Python operators. It's in reverse order (the upper operator holds higher precedence than the lower operator).

Type Converrsion and Type Casting:

The two terms type casting and the type conversion are used in a program to convert one data type to another data type. The conversion of data type is possible only by the compiler when they are compatible with each other. Let's discuss the difference between type casting and type conversion in any programming language.

1. type casting:

What is a type casting? When a data type is converted into another data type by a programmer or user while writing a program code of any programming language, the mechanism is known as type casting. The programmer manually uses it to convert one data type into another. It is used if we want to change the target data type to another data type. Remember that the destination data type must be smaller than the source data type. Hence it is also called a narrowing conversion.

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)

- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals
- Integers:

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
#q = int('1.5')     # unable to do

print(x,"type of x : ",type(x))
print(y,"type of y : ",type(y))
print(z,"type of z : ",type(z))
#print(q,"type of q : ",type(q))

1 type of x :  <class 'int'>
2 type of y :  <class 'int'>
3 type of z :  <class 'int'>
```

- Floats:

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2

print(x,type(x))
print(y,type(y))
print(z,type(z))
print(w,type(w))

1.0 <class 'float'>
2.8 <class 'float'>
3.0 <class 'float'>
4.2 <class 'float'>
```

- Strings:

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'

print(x,type(x))
print(y,type(y))
print(z,type(z))

s1 <class 'str'>
2 <class 'str'>
3.0 <class 'str'>
```

2. type conversion:

What is a type conversion?

type conversion is a technique to convert data type from one to another in possible ways. to see the data type we use type() function and to covernt data type in typer converstion we use int(), float() and complex() functions.

You can convert from one type to another with the int(), float(), and complex() methods:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

# before conversion
print("x : ",type(x))
print("y : ",type(y))
print("z : ",type(z))

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

```
x :  <class 'int'>
y :  <class 'float'>
z :  <class 'complex'>
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

Difference betweenType Casting &Type Conversion:

Input() function :

it's used to take user input. By default, it returns the user input in form of a string.

```
name = input('enter your name : ')
add = input("enter your address : ")

print("my name is ",name," and ","i live in ",add)    #default type
print(f"my name is {name} and i live in {add}")      # using f-string

enter your name :  raj
enter your address :  indore

my name is  raj  and  i live in  indore
my name is raj and i live in indore
```

NOTE: input function take input as 'string' by default but with type conversion we can take input as any data type i.e. int and float.

```
a = input("enter any number: ")
print(a)
print(type(a))

enter any number:  12.4

12.4
<class 'str'>

a = float(input("enter any number: "))
print(a)
print(type(a))

enter any number:  12

12.0
<class 'float'>

a = int(input("enter any number: "))
print(a)
print(type(a))

enter any number:  12

12
<class 'int'>

a = float(input("enter no. 1 :"))
b = float(input("enter no. 2 :"))
c = a + b
sum = 'sum of %f and %f is'%(a,b)
print(sum,c)

enter no. 1 : 12.5
enter no. 2 : 13.7

sum of 12.500000 and 13.700000 is 26.2
```

Strings (data type) :

sequence of character is known as string. there are various operation and functions, we can applied on strings.

```
str1 = "python programming"

print(str1)
print(type(str1))
```

```
python programming
<class 'str'>
```

```
greet = 'good morning !!!, '
name = input("enter name : ")

print(greet+name)

msg = greet+name
print(msg)
```

```
enter name :  xxxx

good morning !!!, xxxx
good morning !!!, xxxx
```

- String indexing:

In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character, and so on.

While accessing an index out of the range will cause an IndexError. Only Integers are allowed to be passed as an index, float or other types that will cause a TypeError.

1. positive indexing : The first character has an index of 0 , the character at position x has an index x - 1 , and the last character in the string has an index of string_length - 1 . If the index is greater than string_length - 1 , Python returns the error IndexError: string index out of range.

```
name = "Rajendra Kumar Randa"
print("index of 0 :", name[0])
print("index of 2 :", name[2])
print("index of 18 :", name[18])
```

```
index of 0 : R
index of 2 : j
index of 18 : d
```

ERROR : name = "Rajendra Kumar Randa" print("index of 23 :", name[23])

IndexError Traceback (most recent call last) Cell In[11], line 2 1 name = "Rajendra Kumar Randa" ----> 2 print("index of 23 :", name[23])

IndexError: string index out of range

2. nagative index: Negative indexing retrieves elements from the end by providing negative numbers as sequence indexes.

it goes to string = [-n, ... , -4, -3, -2, -1]

```
name = "Rajendra Kumar Randa"

print("index of 2 :", name[-2])
print("index of 2 :", name[-14])
```

```
index of 2 : d
index of 2 : r
```

- String Slicing : In Python, the String Slicing method is used to access a range of characters in the String. Slicing in a String is done by using a Slicing operator, i.e., a colon (:). One thing to keep in mind while using this method is that the string returned after slicing includes the character at the start index but not the character at the last index.

SYNTAX : str_name(start_index:end_index:skip_increment_indexing_number)

```
name = "Rajendra Kumar Randa"

print(name)
print(type(name))
print(len(name))

print(name[0])
print(name[3])
print(name[4]) #not work for out of index
print(name[:5])
print(name[:4])
print(name[0:])
# print(name[1:3]) #[1:2]
c = name[-4:-2] #nagative index
print(c)
print(name[-4:-2])
d = name[1::5] #ski p indexing value
d = name[0:6:1] #string[start_index:end_index:skip_index]
print(d)
```

```
Rajendra Kumar Randa
<class 'str'>
20
R
e
n
```

```
Rajen
Raje
Rajendra Kumar Randa
an
an
Rajend
```

```
l = [0, 1, 2, 3, 4, 5, 6, 7, 8]
m = l[0:9:2]
print(m)
```

```
[0, 2, 4, 6, 8]
```

Functions on Strings :

```
story = '''In “The Necklace” by Guy de Maupassant, the main character, 

Mathilde, has always dreamed of being an aristocrat but lives in 
poverty. Embarrassed about her lack of fine possessions, she borrows a 

necklace from a wealthy friend but loses it. The story is known for 
its
subversive and influential twist ending.'''
```

```
story
```

```
'In “The Necklace” by Guy de Maupassant, the main character, \
nMathilde, has always dreamed of being an aristocrat but lives in \
npoverty. Embarrassed about her lack of fine possessions, she borrows 
a \nnecklace from a wealthy friend but loses it. The story is known 
for its \nsubversive and influential twist ending.'
```

```
print(story)
```

```
In “The Necklace” by Guy de Maupassant, the main character, 
Mathilde, has always dreamed of being an aristocrat but lives in 
poverty. Embarrassed about her lack of fine possessions, she borrows a 

necklace from a wealthy friend but loses it. The story is known for 
its
subversive and influential twist ending.
```

```
print("length : ",len(story))
```

```
length :  312
```

```
story = "A quick brown fox jumps  over the lazy dog . dog are smart 
pet."

print(story)
print(len(story))
print(story.endswith("dog"))
print(story.endswith("."))
print(story.count("v"))
print(story.count("brown"))
print(story.count(" "))
print(story.find("fox"))
print(story.find(" "))
print(story.replace("dog", "cat"))

A quick brown fox jumps  over the lazy dog . dog are smart pet.
63
False
True
1
1
14
14
1
A quick brown fox jumps  over the lazy cat . cat are smart pet.
```

space sequence character:

they are special characters who used to make output better. some of them are :

- \n : new line
- \t : a tab
- \ : useing
- : use same quote as start or end inside the strings and etc.

EXAMPLES:

```
new_story = "this is a ball.\nthis color is blue.\nin maths,\n\tif we
divide 2 into 2 (2/2) is 1\n and this is harry\'s\tball.\\'"

print(new_story)

this is a ball.
this color is blue.
in maths,
     if we divide 2 into 2 (2/2) is 1
 and this is harry's  ball.\'
```

- Quiz.format giving letter usinf space sequence characters. letter = "Dear <|NAME|>, greeting from company 'ABC'. I'm happy to tell about your selection. you're selected. have a great day! date: <|DATE|>". NOTE: "NAME" and "DATE" are given by users.

```
letter = '''Dear <|NAME|> ,
greeting from company 'ABC'. I'm happy to tell about your selection.
you're selected.\n
have a great day!
```

```
\t\t\t\t\t\tdate: <|DATE|>'''

name = input("enter your name: ")
date = input("enter date: ")

letter = letter.replace("<|NAME|>",name)
letter = letter.replace("<|DATE|>",date)

print("********** greeting ************")
print(letter)
```

```
enter your name:  raj
enter date:  12/12/24

********** greeting ************
Dear raj ,
greeting from company 'ABC'. I'm happy to tell about your selection.
you're selected.

have a great day!
                                 date: 12/12/24
```

QUIZ : find double space and exchange it to single?

1. str1 = "this is my ball."
2. str2 = "this is my pen."
3. str3 = "this is my book."

```
str1 = "this is my  ball."
str2 = "this is my pen."
str3 = "this  is my  book."
doublespace1 = str1.find("  ")
doublespace2 = str2.find("  ")
print(doublespace1)
print(doublespace2)

str3 = str3.replace("  "," ")
print(str3)
```

```
10
-1
this is my book.
```

QUIZ : format the given latter? ltr = "dear harry, this python course is nice!thanks!"

```
formatted_letter = "dear harry, \n\tthis python course is nice!\n\t\t\t\t\t\tthanks!"
print(formatted_letter)
```

```
dear harry,
     this python course is nice!
                                      thanks!
```

List (data type) :

Lists are used to store multiple items in a single variable. it's changable/ mutable and indexed/ordered, it allows duplicates.

- empty list: we represent empty list or list with no elements via "[ ]" sign or list().

```
l = []
print(l)
print(len(l))
print(type(l))
```

```
[]
0
<class 'list'>
```

```
l = list()
print(l)
print(len(l))
print(type(l))
```

```
[]
0
<class 'list'>
```

- insert function: It is used to insert an element in any list.

SYNTAX: list_name.insert(index,value)

where, index is the position we want to insert the element and value is the value of element we want to insert.

```
l = []
l.insert(0,2)  # list_name[index, value]
print(l)
print(len(l))
print(type(l))
```

```
[2]
1
<class 'list'>
```

```
num = [1, 16, 56, 10, 14, 1, 0]
print(num)
```

```
num.insert(0, 2.3)       # insert(index no, value)
print("insert '2.3' on index[0] : ",num)

[1, 16, 56, 10, 14, 1, 0]
insert '2.3' on index[0] :  [2.3, 1, 16, 56, 10, 14, 1, 0]
```

- basic function of list:

```
a = [1, 2.3, 'ram', True, 0, 0, 0]

print("list : ",a)
print("type : ",type(a))
print("length : ",len(a))

list :  [1, 2.3, 'ram', True, 0, 0, 0]
type :  <class 'list'>
length :  7
```

- indexing in list: it works same as work on string elements. +ve index: it works like: [0,1,2,.......,n-2,n-1] -ve index: it works like: [n,n-1,.......,-3,-2,-1]

```
a = [1, 2.3, 'ram', True, 0, 0, 0]

print("list : ",a)
print("index of [0] : ",a[0])
print("index of [2] : ",a[2])
print("type of element of list on index[1] : ",type(a[1]))
print("type of element of list on index[2] : ",type(a[2]))

list :  [1, 2.3, 'ram', True, 0, 0, 0]
type :  <class 'list'>
length :  7
index of [0] :  1
index of [2] :  ram
type of element of list on index[1] :  <class 'float'>
type of element of list on index[2] :  <class 'str'>

friends = ["ram","syam","mohan","sohan","meena","sita", 4, 500]

print(" list of friends = ",friends)
print("friends[3] =  ",friends[3])
print("friends[-3] =  ",friends[-3])

 list of friends =  ['ram', 'syam', 'mohan', 'sohan', 'meena', 'sita', 
4, 500]
friends[3] =   sohan
friends[-3] =   sita
```

- update function: it is used to update the value. list_name[index] = new_value

```
a = [1, 2.3, 'ram', True, 0, 0, 0]

print("before update item on index[6] list will be : ",a)
a[6] = 9
print("after update item on index[6] list will be : ",a)

before update item on index[6] list will be :  [1, 2.3, 'ram', True,
0, 0, 0]
after update item on index[6] list will be :  [1, 2.3, 'ram', True, 0,
0, 9]

friends = ["ram","syam","mohan","sohan","meena","sita", 4, 500]

print(" list of friends = ",friends)
friends[1] = "juhi"
print("after update item on index[1] friends list will be : ",friends)

 list of friends =  ['ram', 'syam', 'mohan', 'sohan', 'meena', 'sita',
4, 500]
after update item on index[1] friends list will be :  ['ram', 'juhi',
'mohan', 'sohan', 'meena', 'sita', 4, 500]

friends = ["ram","syam","mohan","sohan","meena","sita", 4, 500]

print(" list of friends = ",friends)
friends[-1] = "kajal"
print("after update item on index[-1] friends list will be :
",friends)

 list of friends =  ['ram', 'syam', 'mohan', 'sohan', 'meena', 'sita',
4, 500]
after update item on index[-1] friends list will be :  ['ram', 'syam',
'mohan', 'sohan', 'meena', 'sita', 4, 'kajal']
```

- slicing in list: it means print or choose specific part of list. SYNTAX: list_name[starting_index:ending_index:skipping_index_numbers]

NOTE: you can skip anyone of part of index, if you skip any part than then default values of starting_index is '0', ending_index is 'n' and skipping_index_numbers is 1

```
friends = ["ram","syam","mohan","sohan","meena","sita", 4, 500]

print(" list of friends = ",friends)
print("slice with index friends[:4] = ",friends[:4]) #slice with index
print("slice with index friends[-4:] = ",friends[-4:]) #slice with
index

 list of friends =  ['ram', 'syam', 'mohan', 'sohan', 'meena', 'sita',
4, 500]
slice with index friends[:4] =  ['ram', 'syam', 'mohan', 'sohan']
slice with index friends[-4:] =  ['meena', 'sita', 4, 500]
```

NOTE: list work on all function of string as string.

```
num = [1, 16, 56, 10, 0, 14, 0]

print("num[:3] = ",num[:3])     # num[0:3]
print("num[3:] = ",num[3:])     # num[3:last element]
print("num[::2] = ",num[::2])    #num[0:n-1:2nd element skipped]

num[:3] =  [1, 16, 56]
num[3:] =  [10, 0, 14, 0]
num[::2] =  [1, 56, 0, 0]
```

- sort function: used to sort the elements of list. SYNTAX: list_name.sort()

```
num = [1, 16, 56, 10, 0, 14, 0]
print("list : ",num)
num.sort()
print("sorted list : ",num)

list :  [1, 16, 56, 10, 0, 14, 0]
sorted list :  [0, 0, 1, 10, 14, 16, 56]
```

- reverse function: used to reverse the elements of list. SYNTAX: list_name.reverse()

```
num = [1, 16, 56, 10, 0, 14, 0]

print("list : ",num)
num.reverse()
print("reversed list : ",num)

list :  [1, 16, 56, 10, 0, 14, 0]
reversed list :  [0, 14, 0, 10, 56, 16, 1]
```

- append function: this function add value after the last element. SYNTAX: list_name.append(value)

```
num = [1, 16, 56, 10, 14, 1, 0]
print(num)

num.append(50)
print("append 50 : ",num)

[1, 16, 56, 10, 14, 1, 0]
append 50 :  [1, 16, 56, 10, 14, 1, 0, 50]
```

- pop function: it is used to remove the elements using index or last added item. SYNTAX: type 1-list_name.pop(index), type 2-list_name.pop() type 1,remove the element exists in given index and type 2, remove last added item in list.

type 1-list_name.pop(index), type 1,remove the element exists in given index in list.

```
num = [1, 16, 56, 10, 14, 1, 0]
print(num)
num.pop(1)      # pop(index)
print("pop '1' : ",num)
```

```
[1, 16, 56, 10, 14, 1, 0]
pop '1' :  [1, 56, 10, 14, 1, 0]
```

type 2-list_name.pop() type 2, remove last added item in list.

```
num = [1, 16, 56, 10]
print(num)
# insert some elements in the last index(append)
num.append(10)
num.append(14)
num.append(1)
num.append(0)
num.insert(1,13)
print(num)

num.pop()
print("pop : ",num)  #remove last added item
```

```
[1, 16, 56, 10]
[1, 13, 16, 56, 10, 10, 14, 1, 0]
pop :  [1, 13, 16, 56, 10, 10, 14, 1]
```

- remove function: it removes the element from list. SYNATX: list_name(value)

```
num = [1, 16, 56, 10, 14, 1, 0]
print(num)

num.remove(0)   # remove(value)
print("remove '0' : ",num)
```

```
[1, 16, 56, 10, 14, 1, 0]
remove '0' :  [1, 16, 56, 10, 14, 1]
```

- del function: it is used to delete element and also delete whole list from source. SYNTAX: type 1-list_name.del(index), type 2-list_name.del()

type 1-list_name.pop(index)

```
num = [1, 16, 56, 10, 14, 1, 0]
print(num)

del num[1]      # delete item permanently from source list on index 1
print("del index[1] : ",num)
```

```
[1, 16, 56, 10, 14, 1, 0]
del index[1] :  [1, 56, 10, 14, 1, 0]
```

type 2-list_name.pop()

- delete list permanently from source

diference between del and clear function in list:

```

```

- extend function: it is used to merge two data types.
- list with list

```
x = ['a','b','c']
y = ['p','q','z']
x.extend(y)
print(x)

['a', 'b', 'c', 'p', 'q', 'z']
['a', 'b', 'c', 1, 2, 3]
```

- list with tuple

```
p = ['a','b','c']
q = (1,2,3)
p.extend(q)
print(p)

['a', 'b', 'c', 1, 2, 3]
```

- loops in list:
- check wether given string is in list or not.

```
l = ['ram', 'sohan', 'mohan']
x = input("enter name :")
if x in l:
    print("yes,it is...")
else:
    print("no, it is not...")

enter name : ram

yes,it is...
```

- for loop in list:
- print item one by one all elements

```
x = [0, 10, 20, 30, 40, 50, 60, 70, 8, 90, 100]

for item in x:
    print(item)

0
10
20
30
40
50
60
70
8
90
100
```

- print all elements in list

```
x = [0, 10, 20, 30, 40, 50, 60, 70, 8, 90, 100]

for item in range(len(x)):
    print(x[item])

0
10
20
30
40
50
60
70
8
90
100
```

- print element via index using range() and len()

```
x = [0, 10, 20, 30, 40, 50, 60, 70, 8, 90, 100]

for item in range(len(x)):
    print(x[4])

40
40
40
40
40
40
40
```

```
40
40
40
40
```

- print all element using while loop

```
x = [0, 10, 20, 30, 40, 50, 60, 70, 8, 90, 100]
i = 0
while i < len(x):
    print(x[i])
    i = i + 1

0
10
20
30
40
50
60
70
8
90
100
```

- for look in one-line

```
x = [0, 10, 20, 30, 40, 50, 60, 70, 8, 90, 100]

[print(item) for item in x]

0
10
20
30
40
50
60
70
8
90
100

[None, None, None, None, None, None, None, None, None, None, None]
```

Tuple (data type) :

it is collection of all type of data type. it is imutable(unchangable), allowed to hold duplicate values and also indexed/ordered.

- empty tuple

```
empty_tuple = ()
print(empty_tuple)
print(type(empty_tuple))
print(len(empty_tuple))
```

```
()
<class 'tuple'>
0
```

```
empty_tuple = tuple()
print(empty_tuple)
print(type(empty_tuple))
print(len(empty_tuple))
```

```
()
<class 'tuple'>
0
```

- tuple with single value

```
t = (3)   # wrong way
print(t)
print(type(t))

t = (2,)  # right way
print(t)
print(type(t))
```

```
3
<class 'int'>
(2,)
<class 'tuple'>
```

- some basic functions in tuple

```
t = (0, 12, 2.5, 24, 50, 2, 2, 2)
print(t)
print(type(t))
print(type(t[2]))
```

- count function : it returns number of repetition of any specific value. SYNTAX: tuple_name.count(value)

```
t = (0, 12, 2.5, 24, 50, 2, 2, 2)
print(t)
print("count '2' : ", t.count(2))
```

```
(0, 12, 2.5, 24, 50, 2, 2, 2)
count '2' :  3
```

- index function : it returns index number of given value. SYNTAX: tuple_name.index(value)

```
t = (0, 12, 2.5, 24, 50, 2, 2, 2)
print(t)
print("index(50) : ", t.index(50))   # return index no by value index(value)
```

```
(0, 12, 2.5, 24, 50, 2, 2, 2)
index(50) :  4
```

- how to update any item in tuple?

steps to update tuples...

1. first, convert tuple into list
2. update item of list
3. convert list into tuple in the end

```
# given tuple
x = ('apple','banana','cherry')
print(x)
print(type(x))

# convert tuple into list and update item
y = list(x)
y[1] = 'kiwi'

print(y)
print(type(y))

# convert list into tuple
x = tuple(y)
print(x)
print(type(x))
```

```
('apple', 'banana', 'cherry')
<class 'tuple'>
['apple', 'kiwi', 'cherry']
<class 'list'>
('apple', 'kiwi', 'cherry')
<class 'tuple'>
```

NOTE: by this way we can perform all operation on tuples.

- loop with tuple:
- for loop in tuple

```
a = (1,3,5,6,7,8)
for i in a:
    print(i)
```

```
1
3
5
6
7
8
```

- for loop with len() in tuple

```python
a = (1,3,5,6,7,8)
for i in range(len(a)):
    print(a[i])
```

```
1
3
5
6
7
8
```

- while loop in tuple

```python
ab = (1,3,4,5,6)
i = 0
while i < len(ab):
    print(ab[i])
    i = i + 1
```

```
1
3
4
5
6
```

math with tuple:

- Join two tuples

```python
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)
```

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

- Multiply the tuple by any number

```python
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2
print(mytuple)
```

```
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

Using Asterisk*

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list.

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

```
apple
banana
['cherry', 'strawberry', 'raspberry']
```

NOTE: Assign the rest of the values as a list in variable who hold asteric sign named "red"

Quiz: some questions about list and tuples.

Quiz: create a list with user define name of 5 fruits.

```
f1 = input("enter fruit no 1 : ")
f2 = input("enter fruit no 2 : ")
f3 = input("enter fruit no 3 : ")
f4 = input("enter fruit no 4 : ")
f5 = input("enter fruit no 5 : ")

l =[f1, f2, f3, f4, f5]

print("list of give fruits are : ", l)
print(type(l))
print(len(l))
```

Quiz: create a collection of marks of 5 subject in increment order.

```
m1  = int(input("enter marks of student no. 1 : "))
m2  = int(input("enter marks of student no. 2 : "))
m3  = int(input("enter marks of student no. 3 : "))
m4  = int(input("enter marks of student no. 4 : "))
m5  = int(input("enter marks of student no. 5 : "))

l = [m1, m2, m3, m4, m5]
l.sort()
print("marks of students are in sorted form : ", l)
```

Quiz: prove that tuple is immutable?

QUIZ: WAP to calculate sum of all element in given list.mylist = [12, 23, 34, 45]

- way 1 : sum of element of list without using loop

```
mylist = [12, 23, 34, 45]

sum = mylist[0] + mylist[1] + mylist[2] + mylist[3]
print("sum : ",sum)

sum :  114
```

*way 2 : sum of element of list without using loop

```
mylist = [12, 23, 34, 45]

sum = 0
for x in range(0, len(mylist)):
    sum = sum + mylist[x]
print("sum = ",sum)

sum =  114
```

QUIZ: WAP to count zeroes in given tuple.

```
t  = ( 0, 21, 1, 0, 4, 6, 23, 0, 0 )
print("total no of zeroes : ",t.count(0))

total no of zeroes :  4
```

QUIZ: Using the tuple() method to make a tuple.

- note the double round-brackets

```
thistuple = tuple(("apple", "banana", "cherry"))
print(thistuple)

('apple', 'banana', 'cherry')
```

ERROR:if you just use tuple() because tuple() with single paranthesis take only one args.

thistuple = tuple("apple", "banana", "cherry") print(thistuple)

TypeError Traceback (most recent call last) Cell In[9], line 1 ----> 1 thistuple = tuple("apple", "banana", "cherry") 2 print(thistuple)

TypeError: tuple expected at most 1 argument, got 3

QUIZ: WAP to find out that given item is exist in tuple or not.

```
t = (1,2,3,4,56)
print(t)
x = int()

if x in t:
    print("true")
else:
    print("false")
```

```
true
```

Dictionary (data type) :

its collection of 'key-value' pairs. it is indexed, unordered, mutable(changable), and hold non-duplicate keys. if two keys would be same than key will hold new updated value and if values are same it doesn't matter.

- empty dictionary

```
empty_dict = {}
print(empty_dict)
print(type(empty_dict))
```

```
{}
<class 'dict'>
```

```
empty_dict = dict()
print(empty_dict)
print(type(empty_dict))
```

```
{}
<class 'dict'>
```

- simple dictionary

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

```
{'a': 1, 'b': 2, 'c': 3}
<class 'dict'>
```

```
d2 = dict(x = 1, y = True, z = 'raj')
print(d2)
print(type(d2))
```

```
{'x': 1, 'y': True, 'z': 'raj'}
<class 'dict'>
```

- Let's see a complex dictionary

```
myDict = {
    "red"  : 'one type of color',
    "fast" : 'in a quick manner',
    "harry": 'a programmer',
    "l"    : [1, 2, 3, 4, 5],
    "t"    : (10, 20, 30, 40, 50),
    "D"    : {
        "nu": [11, 12, 13, 14],
        "tu": (110, 120, 130, 140),
        "di": {
            "x" : [7, 8, 9],
            "y" : (77, 88, 99),
            "d2":{"m": [9, 99, 999],
                  "n": [8, 88, 888],
                  "o":(5,55,555),
                 }
             }
    },
  }

# print dictionary
print("Dictionary is : ",myDict)

# type of element N dictionary
print("type of Dictionary is : ",type(myDict))
print("type of 'o' is : ",type(myDict['D']['di']['d2']['o']))


# print elements
print("value of 'red' is : ",myDict['red'])
print("value of 'fast' is : ",myDict['fast'])
print("value of 'harry' is : ",myDict['harry'])
print("value of 'l' is : ",myDict['l'])
print("value of 't' is : ",myDict['t'])
print("value of 'D' is : ",myDict['D'])

# print inner elements
print("value of 'nu' is : ",myDict['D']['nu'])
print("value of 'di' is : ",myDict['D']['di'])

# print inner value
print("value of 'x' is : ",myDict['D']['di']['x'])
print("value of 'd2' is : ",myDict['D']['di']['d2'])
print("value of 'o' is : ",myDict['D']['di']['d2']['o'])

print("keys : ",myDict.keys())
print("type of values : ",type(myDict.values()))
print("values : ",myDict.values())
print("type of keys : ",type(myDict.keys()))
```

```
# item() if you want to see key-value pair
print("item : ",myDict.items())

# convert key to list of keys:
print("keys : ",list(myDict.keys()))

# add value
myDict['marks'] = ['a','b','c']
myDict['D']['di']['hero'] = 'save the life'
print(myDict.items())
```

```
Dictionary is :  {'red': 'one type of color', 'fast': 'in a quick
manner', 'harry': 'a programmer', 'l': [1, 2, 3, 4, 5], 't': (10, 20,
30, 40, 50), 'D': {'nu': [11, 12, 13, 14], 'tu': (110, 120, 130, 140),
'di': {'x': [7, 8, 9], 'y': (77, 88, 99), 'd2': {'m': [9, 99, 999],
'n': [8, 88, 888], 'o': (5, 55, 555)}}}}
type of Dictionary is :  <class 'dict'>
type of 'o' is :  <class 'tuple'>
value of 'red' is :  one type of color
value of 'fast' is :  in a quick manner
value of 'harry' is :  a programmer
value of 'l' is :  [1, 2, 3, 4, 5]
value of 't' is :  (10, 20, 30, 40, 50)
value of 'D' is :  {'nu': [11, 12, 13, 14], 'tu': (110, 120, 130,
140), 'di': {'x': [7, 8, 9], 'y': (77, 88, 99), 'd2': {'m': [9, 99,
999], 'n': [8, 88, 888], 'o': (5, 55, 555)}}}
value of 'nu' is :  [11, 12, 13, 14]
value of 'di' is :  {'x': [7, 8, 9], 'y': (77, 88, 99), 'd2': {'m':
[9, 99, 999], 'n': [8, 88, 888], 'o': (5, 55, 555)}}
value of 'x' is :  [7, 8, 9]
value of 'd2' is :  {'m': [9, 99, 999], 'n': [8, 88, 888], 'o': (5,
55, 555)}
value of 'o' is :  (5, 55, 555)
keys :  dict_keys(['red', 'fast', 'harry', 'l', 't', 'D'])
type of values :  <class 'dict_values'>
values :  dict_values(['one type of color', 'in a quick manner', 'a
programmer', [1, 2, 3, 4, 5], (10, 20, 30, 40, 50), {'nu': [11, 12,
13, 14], 'tu': (110, 120, 130, 140), 'di': {'x': [7, 8, 9], 'y': (77,
88, 99), 'd2': {'m': [9, 99, 999], 'n': [8, 88, 888], 'o': (5, 55,
555)}}}])
type of keys :  <class 'dict_keys'>
item :  dict_items([('red', 'one type of color'), ('fast', 'in a quick
manner'), ('harry', 'a programmer'), ('l', [1, 2, 3, 4, 5]), ('t',
(10, 20, 30, 40, 50)), ('D', {'nu': [11, 12, 13, 14], 'tu': (110, 120,
130, 140), 'di': {'x': [7, 8, 9], 'y': (77, 88, 99), 'd2': {'m': [9,
99, 999], 'n': [8, 88, 888], 'o': (5, 55, 555)}}})])
keys :  ['red', 'fast', 'harry', 'l', 't', 'D']
dict_items([('red', 'one type of color'), ('fast', 'in a quick
manner'), ('harry', 'a programmer'), ('l', [1, 2, 3, 4, 5]), ('t',
(10, 20, 30, 40, 50)), ('D', {'nu': [11, 12, 13, 14], 'tu': (110, 120,
130, 140), 'di': {'x': [7, 8, 9], 'y': (77, 88, 99), 'd2': {'m': [9,
99, 999], 'n': [8, 88, 888], 'o': (5, 55, 555)}, 'hero': 'save the
life'}}), ('marks', ['a', 'b', 'c'])])

newDict = {
    'a': 'this is a',
    'b': 'this is b',
    'c': 'this is c',
}
print(newDict)
print(newDict.items())
print(newDict.keys())
print(newDict.values())

newDict['d'] = 'this is d'
print('after update : ',newDict)

print(newDict['d'])
print(newDict.get('d'))

{'a': 'this is a', 'b': 'this is b', 'c': 'this is c'}
dict_items([('a', 'this is a'), ('b', 'this is b'), ('c', 'this is
c')])
dict_keys(['a', 'b', 'c'])
dict_values(['this is a', 'this is b', 'this is c'])
after update :  {'a': 'this is a', 'b': 'this is b', 'c': 'this is c',
'd': 'this is d'}
this is d
this is d
```

Quiz: Difference between " get() " and traditional "[ ]" method.

- dictionary with if

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")

Yes, 'model' is one of the keys in the thisdict dictionary
```

- chane the value in dictionary using "[ ]" traditional method

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

```
print(thisdict)
# Change the value:
thisdict["year"] = 2018
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

- chane the value in dictionary using "update()" method

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

print(thisdict)
# update dictionary
thisdict.update({'year':2020})
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

- chane the value in dictionary using "[ ]" traditional method

```
# Change the value:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

print(thisdict)
# add pair
thisdict['engin'] = '900CC'
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'engin': '900CC'}
```

- chane the value in dictionary using "update()" method

```
# Change the value:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

```python
print(thisdict)
# add pair
thisdict.update({'color': 'blue'})
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'blue'}
```

Quiz: Difference between " update() " and traditional "[ ]" method.

Ans. The main difference between the update() method and the traditional method of adding key-value pairs to a dictionary in Python is that the update() method merges two dictionaries into one, while the traditional method overwrites the existing dictionary with the new key-value pairs. Using update method we can add new pair but with traditional method we can't.

```python
d1 = {1: 'a', 2: 'b'}
d2 = {3: 'c', 4: 'd'}

print("d1 = ",d1)
print("d2 = ",d2)

d1.update(d2)
print("after update via d1.update(d2) : ")

print("d1 = ",d1)
print("d2 = ",d2)
```

```
d1 =  {1: 'a', 2: 'b'}
d2 =  {3: 'c', 4: 'd'}
after update via d1.update(d2) :
d1 =  {1: 'a', 2: 'b', 3: 'c', 4: 'd'}
d2 =  {3: 'c', 4: 'd'}
```

```python
d1 = {1: 'a', 2: 'b'}
d2 = {3: 'c', 4: 'd'}

print("d1 = ",d1)
print("d2 = ",d2)
d2.update(d1)
print("after update via d2.update(d1) : ")

print("d1 = ",d1)
print("d2 = ",d2)
```

```
d1 =  {1: 'a', 2: 'b'}
d2 =  {3: 'c', 4: 'd'}
after update via d2.update(d1) :
d1 =  {1: 'a', 2: 'b'}
d2 =  {3: 'c', 4: 'd', 1: 'a', 2: 'b'}
```

- update method can also update more than one pairs with creating dict objects

```
d1 = {1: 'a', 2: 'b'}

print("d1 = ",d1)
d1.update([('a', 1), ('b', 2)])
print("d1 = ",d1)
```

```
d1 =  {1: 'a', 2: 'b'}
d1 =  {1: 'a', 2: 'b', 'a': 1, 'b': 2}
```

```
d1 = {1: 'a', 2: 'b'}
d2 = {3: 'c', 4: 'd'}

print("d1 = ",d1)
print("d2 = ",d2)

d1[1] = 'x'
d2[3] = 'p'

print("after change value:")

print("d1 = ",d1)
print("d2 = ",d2)
```

```
d1 =  {1: 'a', 2: 'b'}
d2 =  {3: 'c', 4: 'd'}
after change value:
d1 =  {1: 'x', 2: 'b'}
d2 =  {3: 'p', 4: 'd'}
```

- delete pair in dictionary

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  'engin': '900CC'
}

print(thisdict)
# delete pair
del thisdict['engin']
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'engin': '900CC'}
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

- loops in dictionary
- print all key names

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
 }

# print all key names
for x in thisdict:
    print(x)

brand
model
year
```

- print all keys

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
 }

# print all keys
for x in thisdict.keys():
    print(x)

brand
model
year
```

- print all values

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
 }

# print all values
for x in thisdict:
    print(thisdict[x])
```

- print all values

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
 }

# print all values
for x in thisdict.values():
    print(x)
```

```
Ford
Mustang
1964
```

- print pairs of dict

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
 }

# print pairs of dict
for x,y in thisdict.items():
    print(x,y)
```

- print pairs of dict in better way

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
 }

# print pairs of dict in better way
for x,y in thisdict.items():
    print(x," : ",y)

brand  :  Ford
model  :  Mustang
year  :  1964
```

- copy the dictionary from one to another object
- using tradional method

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
 }

# make a copy of dict using dict()
mydict = dict(thisdict)
print(mydict)

{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

- using copy method

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
 }

# make a copy of dict using co[y]
mydict = thisdict.copy()
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

- nexted dictionary

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}

print(myfamily["child2"]["name"])
```

```
Tobias
```

NOTE : Python has a set of built-in methods that you can use on dictionaries.

Set (data type) :

it is non-pair, immutable/non-changable, non.indexed, unordered and not hold duplicate values. you can't change value but u can add or delete values.

- empty set

```
g = set()
print(g)
print(type(g))
```

```
set()
<class 'set'>
```

- add elements in empty set

```
g = set()
print(g)

g.add(10)
g.add(20)
g.add(30)
print(g)
```

```
set()
{10, 20, 30}
```

```
s = {10, 20, 30}
print("s : ",s)
print(type(s))
s.add(40)
print("s : ",s)
print(len(s))
```

```
s :  {10, 20, 30}
<class 'set'>
s :  {40, 10, 20, 30}
4
```

- delete elements using remove method

```
a = {11, 21, 31, 42, 51, True, 12.5, 3+9j}
print(a)
a.remove(21)
print("remove 21 :",a)
a.pop()  # remove random item
print(a)
```

- delete elements using pop method

```
a = {11, 21, 31, 42, 51, True, 12.5, 3+9j}
print(a)
a.pop()  # remove random item
print(a)
```

```
{True, 42, 11, 12.5, 51, 21, (3+9j), 31}
{42, 11, 12.5, 51, 21, (3+9j), 31}
```

```
b = set()
b.add(10)
b.add(20)
b.add(30)
b.add(35)
print(b)
print(type(b))
```

```
{10, 35, 20, 30}
<class 'set'>
```

- set in basic math

1. union :

```
# union
s1 = {1, 2, 3, 4, 5, 7, 9, 11}
s2 = {1, 2, 4, 6, 8, 10, 3, 5}

x = s1.union(s2)

print('s1 = ',s1)
print('s2 = ',s2)

print("union of 's1' and 's2' is :",x)

s1 =  {1, 2, 3, 4, 5, 7, 9, 11}
s2 =  {1, 2, 3, 4, 5, 6, 8, 10}
union of 's1' and 's2' is : {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
```

2. intersectoin :

```
# intersection
s1 = {1, 2, 3, 4, 5, 7, 9, 11}
s2 = {1, 2, 4, 6, 8, 10, 3, 5}

y = s1.intersection(s2)

print('s1 = ',s1)
print('s2 = ',s2)

print("intersection of 's1' and 's2' is :",y)

s1 =  {1, 2, 3, 4, 5, 7, 9, 11}
s2 =  {1, 2, 3, 4, 5, 6, 8, 10}
intersection of 's1' and 's2' is : {1, 2, 3, 4, 5}
```

3. symmetric_difference_update :

```
# union and intersection
s1 = {1, 2, 3, 4, 5, 7, 9, 11}
s2 = {1, 2, 4, 6, 8, 10, 3, 5}

z = s1.symmetric_difference_update(s2)

print('s1 = ',s1)
print('s2 = ',s2)
print("all element who are not in both set s1 and s2 : ",z)

s1 =  {6, 7, 8, 9, 10, 11}
s2 =  {1, 2, 3, 4, 5, 6, 8, 10}
all element who are not in both set s1 and s2 :  None
```

- print more than one set in single time

```
s1 = set()
s2 = {'a', True, 2.5, 12}
s3 = set(('char', False, 12.65, 13))
print(s1,s2,s3)

set() {True, 2.5, 'a', 12} {False, 13, 'char', 12.65}
```

- set with loops
- for in sets

```
myset = {'raj', 'mohan', 'sita', 'gita'}
for x in myset:
    print(x)

gita
sita
raj
mohan
```

- check value present or not

```
print('raj' in myset)

True
```

- update function in set

```
a1 = {'a', 'b', 'c'}
a2 = {1, 2, 3}
a1.update(a2)
print(a1)
a2.update(a1)
print(a2)

{'c', 1, 'b', 2, 3, 'a'}
{1, 2, 3, 'c', 'b', 'a'}
```

- remove function

```
o = {'a', 12, 'd', 45, 's'}

print("set : ",o)
```

```
o.remove(12)
print('after removing 12 : ',o)

set :  {'d', 12, 45, 'a', 's'}
after removing 12 :  {'d', 45, 'a', 's'}
```

- discard function

```
o = {'a', 12, 'd', 45, 's'}

print("set : ",o)
o.discard('a')
print('after discarding "a" : ',o)

set :  {'d', 12, 45, 'a', 's'}
after discarding "a" :  {'d', 12, 45, 's'}
```

- delete function

del method in set : The del keyword in Python can be used to delete an object. In Python, everything is an object, so the del keyword can also be used to delete variables, lists, or parts of a list, etc.

One way to use the del keyword is to delete an element from a set. To do this, you can pass the element you want to delete to the del keyword. For example, the following code will delete the element 1 from the set my_set:

```
x = {1, 3, 5, 7, 9}
print(x)
del x

{1, 3, 5, 7, 9}
```

Quiz : Difference between discard and remove function?

Ans.: There are two methods to remove an item from a set in Python: discard() method: The discard() method removes the specified item from the set if it is present. It does not raise an error if the item is not present in the set. remove() method: The remove() method removes the specified item from the set if it is present. It raises an error if the item is not present in the set.t?

Let's see is by code

NOTE:As you can see, the discard() method did not raise an error when the item 6 was not present in the set, while the remove() method did.

Which method you should use depends on your needs. If you are sure that the item you want to remove is present in the set, then you can use the remove() method. However, if you are not sure, then you should use the discard() method to avoid raising an error. How do you delete items from a set?

NOTE : Python has a set of built-in methods that you can use on sets.

Ques.: create empty dict and allow your 4 friend to enter their fav_lang and use keys as their names, assume name will be unique

```
f1 = input('enter favorite language of geeta : ')
f2 = input('enter favorite language of meeta : ')
f3 = input('enter favorite language of seeta : ')
f4 = input('enter favorite language of neeta : ')
dict = {}
dict['geeta'] = f1
dict['meeta'] = f2
dict['seeta'] = f3
dict['neeta'] = f4
print(dict.items())
```

```
enter favorite language of geeta :  eng
enter favorite language of meeta :  hin
enter favorite language of seeta :  guj
enter favorite language of neeta :  marathi
```

```
dict_items([('geeta', 'eng'), ('meeta', 'hin'), ('seeta', 'guj'),
('neeta', 'marathi')])
```

Ques.: can you change the value inside a list which is contained in sets s = {8, 7, 12, [1, 2]}

Ans.: no, coz set is immutable so list can never containned in set and another thing if it'll be tuple than you can't change coz it's non-changable means hashable

Indentation:

the spaces at the beginning of a code line. Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important. Python uses indentation to indicate a block of code.it is white space before statements to make them a single unit or block example: a = 22 if a > 9 : print("greater") else: print("lesser")

pass statement:

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

pass: it means null statement(stands for do nothing with previous statement)

```
a = 33
b = 200

if b > a:
    pass

i = 4
if i > 3 :
    pass
```

```
while i > 6 :
    pass
def run(player):
    pass
print("that's it...")

that's it...
```

- we can use pass statement to skip writing control statement or code inside of function to complete it later

```
def dummy(drug):
    pass

dummy('a')
print("usefull to create empty function using 'pass' statement ")

usefull to create empty function using 'pass' statement
```

Control statements:

```
        it works like a ladder when 1st condition will go true it'll
go for execute the code under it rather than go for next condition or
code.
```

it's work ladder if 1st will true than it'll never go for next. if, elif and else also optional and work anywhere if-elif-else ladder.

python support usual conditions from math:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

Type of control statement in python:

1. if
2. nested if
3. elif
4. if-else
5. if-elif-else

1. if : it works only and only when if condition will be true than next statement will execute.

SYNTAX : if(condition): code/statements

- execute only print() and only condition will true

```
a = 33
b = 200
if b > a :
    print("b is greater than a")

b is greater than a
```

- execute but no output when condition isn't true

```
a = 33
b = 200
if b < a :
    print("b is greater than a")
```

- one life with if:

```
a, b = 15, 12
if a > b : print("a is greater than b")

a is greater than b
```

2. Nested if : we can use multiple if inside each other with condition and also inside itself as per our need. innner if only work if outer if executes.

SYNTAX : if(condition): code/statements if(condition): code/statements ....

```
age = 19

if age >= 18:
    print("you are eligible to vote")
    if age >= 21:
        print("you can drive a car")

you are eligible to vote

age = 26

if age >= 18:
    print("you are eligible to vote")
    if age >= 21:
        print("you can drive a car")

you are eligible to vote
you can drive a car

age = 15

if age >= 18:
    print("you are eligible to vote")
    if age >= 21:
        print("you can drive a car")

age = int(input("enter age :"))

if age >= 18:
    print("you are eligible to vote")
    if age >= 21:
        print("you can drive a car")

enter age : 78

you are eligible to vote
you can drive a car
```

3. elif : It works when previous conditions were not true, then try this condition.

SYNTAX : if(condition): code/statements elif(condition): code/statements ....

```
a = 33
b = 33

if a > b :
    print("a is greater than b...")
elif a == b :
    print("a and b other are equals")

a and b other are equals

a = 33
b = 32

if a > b :
    print("a is greater than b...")
elif a == b :
    print("a and b other are equals")

a is greater than b...

a = 12

if(a==7):
  print("yes")
elif(a>56):
  print("yes or no") #no output
```

4. else : it works whenever another all conditions will be false. it can work with all statements.

SYNTAX : loop/control statement(condition): code/statements .... else: code/statemrnts

```
a = 22
if a > 9 :
    print("greater")
else:
    print("lesser")
```

```
greater
```

```
a = 2
if a > 9 :
    print("greater")
else:
    print("lesser")
```

```
lesser
```

```
age = int(input("enter age: "))
if age >= 18 :
    print("yes")
else:
    print("no")
```

```
enter age:  15
```

```
no
```

- u can work if your age is : age greater than 33 and also age less than 55

```
age = int(input("enter age: "))
if(age>33 and age<55):
    print("you're ",age,",you can work....")
else:
    print("you're ",age,",you can't work...")
```

```
enter age:  35
```

```
you're  35 ,you can work....
```

- one line with if-else:

```
a = 2
b = 330
print("a is greater than b") if a > b else print("b is greater than 
a")
```

```
b is greater than a
```

- nested if-else :

```
x = 40

if x > 10 :
    print("x is above 10 ")
    if x > 20 :
        print("and also above 20")
    else:
        print("and not above 20")
else:
    print("less then 10")

x is above 10
and also above 20
```

5. if-elif-else : it work like ladder of conditions you can use all keywords togather in code also anywhere else in code where you need to execute statements with conditions.

```
a = 200
b = 33
if b > a :
    print("b is greater than a")
elif a == b:
    print("b and a both are equals")
else:
    print("b is lesser than a")

b is lesser than a
```

- one line with ternary operator or conditional expressions:

```
a = 330
b = 330
print("A") if a > b else print("=") if b == a else print("B")

=
```

- 'and' in if statements:

```
a = 200
b = 33
c = 500
if a > b and c > a :
    print("bothe conditions are true...")
else:
    print("both conditions are not true...")

bothe conditions are true...
```

- 'or' in if statements:

```
a = 200
b = 33
c = 500
```

```
if a > b or c < a :
    print("anyone of them, condition should be true...")
else:
    print("both conditions are not true...")

anyone of them, condition should be true...
```

- 'not' in if statement:

```
a = 10
b = 5
if not a < b :
    print("b is not greater than a")
else:
    print("it is  true b is greater than a")

b is not greater than a
```

- 'is' if statement:

```
a = None
if a is None:
    print(True)
elif a < 0 or a > 0 :
    print(False)
else:
    print("a is zero")
```

- 'in' if statement:

```
a = [1, 3, 5, 7, 9]
x = 7

if x in a:
    print(True)
else:
    print(False)

True
```

- a complex example

```
a = 12
if(a==7):
    print("yes")
elif(a>56):
    print("yes or no")
else:
    print("i'm optional")
```

Quiz: a spam comment is defined as a text containing keywords [make a lot of money, buy now,suscribe this,click this,] WAP to detect it.

```
print("make a lot of money,buy now,suscribe this,click this")
comment = input("to check your comment spam or not\nenter your comment :\n")
if("make a lot of money" in comment or "buy now" in comment or "suscribe this" in comment or "click this" in comment):
    print("your comment is spam..")
else:
    print("your comment is not spam..")
```

```
make a lot of money,buy now,suscribe this,click this

to check your comment spam or not
enter your comment :
 click this

your comment is spam..
```

- in other way

```
keywords = ["make a lot of money,buy now", "suscribe this", "click this"]
print("keywords = ",keywords)

comment = input("to check your comment spam or not\nenter your comment :\n")



if comment in keywords :
    print("your comment is spam..")
else:
    print("your comment is not spam..")
```

```
keywords =  ['make a lot of money,buy now', 'suscribe this', 'click this']

to check your comment spam or not
enter your comment :
 aaaa

your comment is not spam..
```

Quiz : WAP given username hold 10char or not

```
username = input("enter your username : ")
total_char = len(username)

if total_char >= 10 :
    print("valid username")
else:
    print("invalid username")

enter your username :  asdegdfgfhejg

valid username
```

- in other way

```
username = input("enter your username : ")

if len(username) >= 10 :
    print("valid username")
else:
    print("invalid username")

enter your username :  gsdfgsdfghsfrthsfgh

valid username
```

Quiz: WAP to check ,whether name in list or not. ["raj","mohan","sohan","sita","gita"]

```
names = ["raj","mohan","sohan","sita","gita"]
name = input("enter name to check wether name is in list or not : ")

if name in names:
    print("name is exist in list...")
else:
    print("name isn't exist in list...")

enter name to check wether name is in list or not :  raj

name is exist in list...
```

Quiz : wap to findout any string/name is exixt in post or not.

```


post = input("enter post : ")
name = input("enter name to search : ")
if name in post:
    print("name is exist in list...")
else:
    print("name isn't exist in list...")
```

LOOP :

Looping means repeating something over and over until a particular condition is satisfied. It is repeatation of block of statement.

Python has two primitive loop commands:

- while loops
- for loops

1. While : With the while loop we can execute a set of statements as long as a condition is true.

SYNTAX : code...........

While(condition): code/statements ....

code...........

- print yes 10 time

```
i = 1
while i < 11 :
    print("yes ",i)
    i = i + 1
print("Done...!")
```

```
yes  1
yes  2
yes  3
yes  4
yes  5
yes  6
yes  7
yes  8
yes  9
yes  10
Done...!
```

- print number series from 1 - 1

```
print("numbers from 1 to 15 : ")

i = 1
while i <= 15 :
    print(i)
    i += 1

print("that's it")
```

```
numbers from 1 to 15 :
1
2
```

```
3
4
5
6
7
8
9
10
11
12
13
14
15
that's it
```

```
print("numbers from 1 to 15 : ")

i = 1
while i < 16 :
    print(i)
    i += 1

print("that's it")
```

```
numbers from 1 to 15 :
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
that's it
```

- print string multiple times

```
i = 1
while i < 6 :
    print(i, ". I'm rajendra")
    i = i + 1
```

```
1 . I'm rajendra
2 . I'm rajendra
3 . I'm rajendra
4 . I'm rajendra
5 . I'm rajendra
```

- print element from collection / list

```
fruit = ['apple', 'banana', 'papaya', 'graps', 'mango', 'cherry']

i = 0
while i < len(fruit) :
    print(fruit[i])
    i = i + 1
```

```
apple
banana
papaya
graps
mango
cherry
```

Extra : while with break, continue and pass.

break : when break condition is true in loop then after successful executing break statement loop stop printing values.

```
i = 1
while i < 9 :
    print(i)
    if i == 5 :
        break
    i = i + 1
```

```
1
2
3
4
5
```

continue: when condition for continue is true than that value skipped and againg loop will start the printing values.

```
i = 0
while i < 6 :
    i += 1
    if i == 3 :
        continue
    print(i)
```

```
1
2
4
5
6
```

"else" works everywhere same as else work with if control statements.

else : when loop condition is false then else condition execute.

```
i = int(input("enter any number : "))
while i < 6 :
    print(i)
    i += 1
else:
    print(" i is greater than 6")
```

```
enter any number :  8

 i is greater than 6
```

2. For loop : For used to iterate through a sequesnce data types like list, tuple, string or iterables with operators (is,in,range)

SYNTAX : code........... .... for iterator in/is dataframes(list,tuple,set): code/statements..... ....

code...........

- print all elemets of sequence.

```
fruits = ['banana', 'mango', 'apple', 'watermalon', 'graps']
for items in fruits:
    print(items)
```

```
banana
mango
apple
watermalon
graps
```

- print all elements with each elements of another element of sequence.

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
    for y in fruits:
        print(x, y)
```

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

- for with range: range() function is used to generate sequence of number.

it's print until 'range' default starting values of range is 0 if there is single value and if there is 2 then first is starting index and end with (index-1).

- print range 0 to 7

```
for i in range(8):
    print(i)

0
1
2
3
4
5
6
7
```

- print number between range of 1 - 10

```
for i in range(1,11):
    print(i)

1
2
3
4
5
6
7
8
9
10
```

- range(1,10,2) = 1-10 with spacing 2

```
for i in range(1,11,2):
    print(i)
```

```
1
3
5
7
9
```

- else in for : else will execute when for loop will exit with condition.

```
for i in range(11):
    print(i)

else:   #when condition will be false
    print("this is inside of 'for'")

0
1
2
3
4
5
6
7
8
9
10
this is inside of 'for'
```

- break with for loop :
- if loop will execute with condition, then break will stop generating output
- else will execute when for loop will exit with condition
- not exit with break, else will run with suceess termination of for

```
for i in range(11):
    print(i)
    if i == 5 :
        break
else:
    print("this is inside of 'for'")      # 0 1 2 3 4 5

0
1
2
3
4
5
```

```
for i in range(8):
    print(i)
    if(i==5):
        break
else:
    print("this is inside of 'for'")  # 0 1 2 3 this is inside of
'for'

0
1
2
3
4
5
```

- continue with for loop : if condition will be true than continue statement skip generating that all input creating under the condition when it is true then start printing the ouput.

```
for i in range(11):
    if i == 5 or i == 7 :
        continue
    print(i)

0
1
2
3
4
6
8
9
10
```

- print multiplication table using loop.

way 1 :

```
num = int(input("enter number to print table : "))
i = 1
for i in range(1,11):
    print(i * num)

enter number to print table :  2

2
4
6
8
10
12
```

```
14
16
18
20
```

way 2 :

```
num = int(input("enter number to print table : "))
i = 1
for i in range(1,11):
    print(num, " X " ,i, " = ",i*num)

enter number to print table :  2

2  X  1  =  2
2  X  2  =  4
2  X  3  =  6
2  X  4  =  8
2  X  5  =  10
2  X  6  =  12
2  X  7  =  14
2  X  8  =  16
2  X  9  =  18
2  X  10  =  20
```

way 3 :

```
num = int(input("enter number to print table : "))
i = 1
for i in range(1,11):
    print(f"{num} X {i} = {i*num}")

enter number to print table :  2

2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20
```

NOTE : try to study str() and fstrings too f"string_char {values,express,calculation}string_char {values,express,calculation}string_char {values,express,calculation}" via online sources or in string chapter.

Quiz : WAP to greet those all whose name start with 's'.

names = ['ram', 'mohan', 'sohan', 'sita', 'suresh', 'lakhan', 'sachin', 'harry']

```
names = ['ram', 'mohan', 'sohan', 'sita', 'suresh', 'lakhan',
'sachin', 'harry']
print("list : ",names,"\ngreeting for all whose names starts with
's' :")

for name in names:
    if name.startswith("s"):
        print("Good morning...!, ",name)
```

```
list :  ['ram', 'mohan', 'sohan', 'sita', 'suresh', 'lakhan',
'sachin', 'harry']
greeting for all whose names starts with 's' :
Good morning...!,  sohan
Good morning...!,  sita
Good morning...!,  suresh
Good morning...!,  sachin
```

Quiz : WAP to check weather number is prime or not.

```
num = int(input("enter any number to check wether number is prime or
not : "))
prime = True
for i in range(2,num):
    if num % i == 0 and num != 2 :
        prime = False
        break

if prime:
    print(num," is prime number...")
else:
    print(num," is not prime number...")
```

```
enter any number to check wether number is prime or not :  7

7  is prime number...
```

Quiz : WAP to findout factorial.

```
num = int(input("enter any number to get factorial : "))
fact = 1
for i in range(1,num+1):
    fact = fact *i
print(f"factorial of {num} is : {fact}")
```

```
enter any number to get factorial :  5

factorial of 5 is : 120
```

Quiz : WAP to print first 'n' natual number using while loop.

```
num = int(input("enter range to get sum of first natural number : "))
sum = 0
i = 1
print("natural no. are : ")
while i < num+1 :
    sum = sum + i
    n = i
    print(n)
    i += 1
print(f"sum of first {i-1} natural no. is : {sum}")
```

```
enter range to get sum of first natural number :  5

natural no. are : 
1
2
3
4
5
sum of first 5 natural no. is : 15
```

Quiz : WAP to print first 'n' natual number using for loop.

```
num = int(input("enter range to get sum of first natural number : "))
sum = 0
i = 1
print("natural no. are : ")
for i in range(i,num+1) :
    sum = sum + i
    n = i
    print(n)
    i += 1
print(f"sum of first {i-1} natural no. is : {sum}")
```

```
enter range to get sum of first natural number :  5

natural no. are : 
1
2
3
4
5
sum of first 5 natural no. is : 15
```

Quiz : WAP to write Reverse Multiplication Table using loop.

```
num = int(input("enter number to get table : "))
limit = 10
```

```python
i = 0
for i in range(limit,i,-1):
    print(f"{num} X {i} = {num*i}")
```

```
enter number to get table :  2
```

```
2 X 10 = 20
2 X 9 = 18
2 X 8 = 16
2 X 7 = 14
2 X 6 = 12
2 X 5 = 10
2 X 4 = 8
2 X 3 = 6
2 X 2 = 4
2 X 1 = 2
```

Let's try patterns using control statements and loops :

1. print tringle :

```python
i = 3
for i in range(3) :
    print(" " * (3-i-1),end="")
    print("*" * (2*i+1),end="")
    print(" " * (3-i-1))
```

```
  *
 ***
*****
```

```python
n = int(input("enter range : "))
i = n
for i in range(n):
    print(" " * (n-i-1),end = "")
    print("*" * (2*i+1),end = "")
    print(" " * (n-i-1))
```

```
enter range :  3
```

```
  *
 ***
*****
```

2. right side tringle :

```python
for i in range(4):
    print("*" * (i+1))
```

```
*
**
***
****

n = int(input("enter range : "))
for i in range(n) :
    print("*" * (i + 1))

enter range :  3

*
**
***
```

3. left side tringle :

```
for i in range(5) :
    print("*" * (5-i))

*****
****
***
**
*

n = int(input("enter range : "))
for i in range(n):
    print("*" * (n-i))

enter range :  5

*****
****
***
**
*
```

- some more patterns :

Function & Recursion:

function:

group of statement or specific task you can use it and we can stop repeatation to write code and use it many times.

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

when a code gets bigger in size and complexity of it grows,it gets difficult to track specific statement. so, a function can be reused by the programmer in a given program any number of times.

SYNTAX : def function_name(parameters): ---> function defination block_of_statements; ---> function_body

function(parameter) ---> funtion calling

1. in simple way

```
marks1 = [45, 78, 86, 77]
marks2 = [75, 89, 92, 70]

p1 = ((sum(marks1) / 400) * 100)
p2 = ((sum(marks2) / 400) * 100)

print(p1)
print(p2)
```

```
71.5
81.5
```

1. using function

```
def percent(marks):                                   # function definition
    return (sum(marks) / 400) * 100                   # function body

marks1 = [45, 78, 86, 77]
marks2 = [75, 89, 92, 70]

p1 = percent(marks1)      # function calling
p2 = percent(marks2)      # function calling

print(p1)
print(p2)
```

```
71.5
81.5
```

- some more

```
def demo():
    print('you are welcome')
def demo1(name):
    print('have a good day !!',name)

demo()
demo1('ram')
```

```
you are welcome
have a good day !! ram
```

types of functions :

function can return results or print directly but most of them, there are two type of functions :

1. built-in : already define in python's library
2. user-defined : defined by users

- built-in function :

```
mark = [10, 20, 30, 40]
print(sum(mark))    # here sum()  is built-in function

100
```

- user defined function :

```
def greet(name):
    print('good morning !!!,', name)    # here greet is user define
function
greet('raj')

good morning !!!, raj
```

- let's see them in deep by some more examples :

```
def add(x,y,z):
    return x + y + z

a = add(45,85,89)
print(a)

219

def add(x,y,z):
    sum =  x + y + z
    return sum

a = add(45,85,89)
print(a)

219

def gret(name):
    gr = "hello " + name
    return gr

gret('Mr. R')

'hello Mr. R'

def add(x,y):
    return x + y
```

```
def sub(x,y):
    return x - y

a = add(45,15)
b = sub(26,11)
print("addition : ",a,"\nsubtraction : ",b)
```

```
addition :  60
subtraction :  15
```

```
def add(x,y):
    add = x + y
    return add

def sub(x,y):
    sub = x - y
    return sub

a = add(45,15)
b = sub(26,11)
print("addition : ",a,"\nsubtraction : ",b)
```

```
addition :  60
subtraction :  15
```

- The terms parameter and argument can be used for the same thing: information that are passed into a function. From a function's perspective: A parameter is the variable listed inside the parentheses in the function definition. An argument is the value that are sent to the function when it is called.

There are two type of arguments in function:

1. Formal Arguments : the args used in function's definition. 2. Actual Arguments : the args used in function's calling. Let's understand it by following example.

```
def add(a,b):     # formal args : a,b
    return a + b

add(2,3)     # Actual args : 2,3
```

```
5
```

there are four types of argument in python functions as per use.

- number of arguments/ required/ positional argument
- keyword arguments
- default arguments
- variable arguments

1. number of arguments/ required argument/ positional argument : there are any number of args can have in function if you put there specific number of args then that would be necessary to give same number of args as input of values at function's calling.

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

it follows the position order.

the no. of args and order of args in the function call must be the same as that in the function definition.

you can't call any function without required args.

Example : we create a function expects 2 arguments. so, we only can gets 2 arguments, it means not more than 2 and aslo less than 2.

SYNTAX : def function_name(args1,args2): ------------> function definition function body.... ....

function_name(parameter1, parameter2) ----------> function calling with required args

```python
def fullName(x,y):
    print(f"my full name is {x} - {y}.")

fullName("raj", "kumar")
```

```
my full name is raj - kumar.
```

```python
def order(a,b,c):
    print("1 = ",a)
    print("2 = ",b)
    print("3 = ",c)

print("way 1 :")
order(1,2,3)
print("way 2 :")
order(3,2,1)   # see the order of values
```

```
way 1 :
1 =  1
2 =  2
3 =  3
way 2 :
1 =  3
2 =  2
3 =  1
```

- if we don't give same number of args as that function expect than

2. Keyword Argument : allow us to change the positional orgers of args. python interpreter uses the keywords provided to match the values with parameters.

SYNTAX : def fun_name(agrs, args): -------> function definition function body..... ....

fun_name(args1 = value1, args1 = value2) -------> function calling with using formal args as keywords.

```
def intro(name,age):
    print(f"my name is {name} with {age} year's old...")

intro(age = 23, name = "raj")

my name is raj with 23 year's old...
```

3. default : we can have a value as default argument in functions, if we have specify name = 'stranger' in line.

for it, it write args with their default values and when we don't call function with right values it use default value given in function's definition.

Example : def function_name(args = values):

```
<b>SYNTAX : </b>
<pre>
def fun_name(agrs1, args2 = "default value"):     -------> function 
definition
    function body.....
    ..................

fun_name(value1, value2)     -------> function calling with using 
formal args as keywords.
</pre>

def greeting(name = 'stranger'):
    gr = "Good Morning !!!, " + name
    return gr

a = greeting()
b = greeting("raj")
print(a)
print(b)

Good Morning !!!, stranger
Good Morning !!!, raj
```

NOTE : it must be used as last argument in function's definition.

3. variable number arguments : to add an orbitary args than specified during the function definition. Special Symbols Used for passing arguments, there are two types of keywords used in arguments. these are -

1. *args (Non-Keyword Arguments)
2. **kwargs (Keyword Arguments)

NOTE : "We use the "wildcard" or " " *notation like this* – args OR **kwargs – as our function's argument when we have doubts about the number of arguments we should pass in a function."

SYNTAX : def function_name(args1,args2,..., *args):

def function_name(args1,args2,..., *kwargs):

SYNTAX : if you have *args & **kwargs arguments. def function_name(args1,args2,..., *args, **kwargs):

- *args : If you do not know how many arguments that will be passed into your function, add a* before the parameter name in the function definition.

  This way the function will receive a tuple of arguments, and can access the items accordingly:

- If the number of arguments is unknown, add a * before the parameter name

```
def myfun(*args):
    print(args)
    print(type(args))

myfun()
myfun(1)
myfun(1,2,3,4,5)

()
<class 'tuple'>
(1,)
<class 'tuple'>
(1, 2, 3, 4, 5)
<class 'tuple'>
```

- sum of numbers

```
def add(*args):
    sum = 0
    for i in args:
        sum = sum + i
    print(sum)

add()
add(1,3)
add(2,4,6)

0
4
12
```

```
def add(a, *args):
    print("a : ",a)
    sum = 0
    for i in args:
        sum = sum + i
    print("sum = ",sum)

add(1,3)
add(2,4,6)
```

```
a :  1
sum =  3
a :  2
sum =  10
```

- If the number of arguments is unknown, add an asteric (*) before the parameter name:

```
def myfun(*name):
    result = f"my full name is {name[0]}, {name[1]} {name[2]}"
    return result

x = myfun('rajendra', 'kumar','randa')
print(x)
```

```
my full name is rajendra, kumar randa
```

NOTE : The phrase Keyword Arguments are often shortened to kwargs in Python documentations.

- **kwargs : If you do not know how many keyword arguments that will be passed into your function, add two asterisk:** before the parameter name in the function definition. This way the function will receive a dictionary of arguments, and can access the items accordingly.

If the number of keyword arguments is unknown, add a double ** before the parameter name.

it takes key-value pairs like dictionary in argument and we can create dictionary and pass it as ** with variable name.

```
def myfun(**kwargs):
    print(kwargs)
    print(type(kwargs))
    print(kwargs['a'])

myfun(a = 3, b = 5)
```

```
{'a': 3, 'b': 5}
<class 'dict'>
3
```

```
def myfun(**kwargs):
    print("a = ", kwargs['a'])
    print(kwargs.items())

myfun(a = 3, b = 5)
```

```
a =  3
dict_items([('a', 3), ('b', 5)])
```

```
def myfun(**kwargs):
    for keys, values in kwargs.items():
        print("key : " ,keys,"value : ",values)

myfun(a = 3, b = 5)
myfun(a = 2, b = 4, c = 6)
```

```
key :  a value :  3
key :  b value :  5
key :  a value :  2
key :  b value :  4
key :  c value :  6
```

```
def myFun(**kwargs):
    for key, value in kwargs.items():
        print(f"{key} = {value}")

# Driver code
myFun(first='Geeks', mid='for', last='Geeks')
```

```
first = Geeks
mid = for
last = Geeks
```

```
def my_function(**kid):
  print("His last name is ", kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

```
His last name is  Refsnes
```

```
def fun1(**kwargs):
    for key,value in kwargs.items():
        print(key,":",value)

dict = {'a': 'this is a','b': 'this is b','c': 'this is c','d': 'this
is d',}
fun1(**dict)
```

```
a : this is a
b : this is b
c : this is c
d : this is d
```

Ques. : difference between *args and **kwargs is that *args is works for collection of values and **kwargs for dictionaries.

by default use this argument in the last like (......**kwargs) example of** kwargs with simple variable,

```
def fun1(s,**kwargs):
    print(s)
    for key,value in kwargs.items():
        print(key,":",value)

str = 'this is string'
dict = {'a': 'this is a','b': 'this is b','c': 'this is c','d': 'this
is d',}
fun1(str,**dict)
```

```
this is string
a : this is a
b : this is b
c : this is c
d : this is d
```

let's see simple, *args and **kwargs type argument togather, order to use arguments : default we use it in this order as : simple, default, *args, **kwargs

main thing in that, if you don't pass them as argument in the time of function calling they'll never give errors

```
def show(x,y,*a,**c):
    print(x)
    print(y)
    print(a)
    print(c)


p = 'this is python'
q = 'it is so simple'
l1 = [1, 2, 3, 4, 5]
l2 = [9, 8 , 7, 6]
dict1 = {0: 'this is 0','x': 'this is x','y': 'this is y','z': 'this
is z',}
dict2 = {0: 'this is 0','x': 'this is x','y': 'this is y','z': 'this
is z',}
show(p,q,l1,l2,dict1,dict2)

# only one * and ** parameter allowed
```

```
this is python
it is so simple
([1, 2, 3, 4, 5], [9, 8, 7, 6], {0: 'this is 0', 'x': 'this is x',
'y': 'this is y', 'z': 'this is z'}, {0: 'this is 0', 'x': 'this is
x', 'y': 'this is y', 'z': 'this is z'})
{}
```

```
def myfun(a,b =10,*t,**d):
    print("a -> ",a)
    print("b -> ",b)
    print("*t -> ",t)
    print("**d -> ", d.items())

x = (10, 20, 30)
y = {'ONE':"this is one",'TWO':"this is two"}
myfun(12,*x ,**y )
```

```
a ->  12
b ->  10
*t ->  (20, 30)
**d ->  dict_items([('ONE', 'this is one'), ('TWO', 'this is two')])
```

4. variable argument : using *args and **kwargs to call a function.

SYNTAX : def fun_name(args1,args2,args3): function_body...... ....

fun_name(*args) fun_name(**kwargs)

```
def myfun(a,b,c,d):
    print(a,b,c,d)

t = (1,3,5,7)
myfun(*t)
```

```
1 3 5 7
```

```
def myfun(a,b,c,d):
    print(a,b,c,d)

d = {'a': "this is A", 'b':"this is B",'c':"this is C",'d': "this is
D"}
myfun(**d)
```

```
this is A this is B this is C this is D
```

Recursion:

the function called himself is known as recursion.

SYNTAX : def function_name(args1, args2, ...): ----> function definition statement 1; ----> function body function_name(args1, args2, ...) ----> function calling itself:recursion

function_name(args1, args2, ...) ----> function calling

- let's see that via example :

```
# factorial : 4! = 4 * 3 * 2 * 1 = 42
# n! = n * (n-1) * .......3 * 2 * 1 || fact(n) = n * fact(n-1)
def fact(n):
    if n == 1 or n == 0:       # coz fact of 1 or 0 is always 1
        return 1
    else:
        return (n*fact(n-1))

x = fact(5)
print(x)

120
```

NOTE : be careful with using recursion working to ensure that the function doesn't call infinitly himself. recursion is sometimes the most direct way to code an algorithm.

lambda/ Anonyoums function :

this function create using 'lanbda' keyword. A lambda function is a small anonymous function. it takes any number of arguments, but can only have one expression. it is also known as one-line function.

SYNTAX : lambda arguments : expression

```
# lambda function:
a = lambda num : num +1; print(a(5))
square = lambda x : x*x
sum = lambda x, y, z : x+y+z
x = 3
y = 2
print(square(x)) #9
print(sum(x,y,1)) #6

6
9
6
```

Ques. : Why Use Lambda Functions? Ans. : The power of lambda is better shown when you use them as an anonymous function inside another function. Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number. why we use: if you wwant to create any function with one expression or need to return function as so you can use it that time and also can use it anywhere.

- Add 10 to any given any number, and return the result.

```
x = lambda a : a + 10
print(x(5))

15
```

- Multiply argument a with argument b and return the result:

```
x = lambda a,b : a * b
x(2,4)

8
```

- Summarize argument a, b, and c and return the result:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))

13
```

Array in python :

Arrays are used to store multiple values in one single variable. it is the collection of dis-similar data elements in columns or in rows.

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

SYNTAX : array_name = [element1, element2, element3, ..., elementn]

NOTE : Python does not have built-in support for Arrays, but Python Lists can be used instead.

NOTE : This page shows you how to use LISTS as ARRAYS, however, to work with arrays in Python you will have to import a library, like the "NumPy library".

- create an array

```
ar = []
print(ar)
print(type(ar))

[]
<class 'list'>

cars = ['ford', 'volvo', 'BMW']

print(cars)
print(type(cars))

['ford', 'volvo', 'BMW']
<class 'list'>
```

- access the elements

```
cars = ['ford', 'volvo', 'BMW']

x = cars[0]

print(x)
```

```
ford
```

- change the value

```
cars = ['ford', 'volvo', 'BMW']

cars[0] = 'Toyota'

print(cars)

['Toyota', 'volvo', 'BMW']
```

- length of an array

```
cars = ['ford', 'volvo', 'BMW']

print(len(cars))

3
```

NOTE : The length of an array is always one more than the highest array index.

- loop in array

```
cars = ['ford', 'volvo', 'BMW']

for car in cars:
    print(car)

ford
volvo
BMW
```

- add element in array

```
cars = ['ford', 'volvo', 'BMW']

cars. Append("honda")
print(cars)

['ford', 'volvo', 'BMW', 'honda']
```

- remove elements in array

1. pop() : you can use pop() to remove any element of array.

```
cars = ['ford', 'volvo', 'BMW']

cars.pop(1)

print(cars)
```

```
['ford', 'BMW']
```

2. remove() : You can also use the remove() method to remove an element from the array.

```
cars = ['ford', 'volvo', 'BMW']

cars.remove('volvo')

print(cars)

['ford', 'BMW']
```

NOTE : The list's remove() method only removes the first occurrence of the specified value.

Ques.: what is the difference between pop() and remove(). difference : pop() used index to remove elements and remove() used values to remove any elements.

Array Methods

Python has a set of built-in methods that you can use on lists/arrays NOTE : you read them all in list chapter.. Method Description append()Adds an element at the end of the list clear()Removes all the elements from the list copy()Returns a copy of the list count()Returns the number of elements with the specified value extend()Add the elements of a list (or any iterable), to the end of the current list index()Returns the index of the first element with the specified value insert()Adds an element at the specified position pop()Removes the element at the specified position remove()Removes the first item with the specified value reverse()Reverses the order of the list sort()Sorts the list

Python Object Oriented Programming : In Python, object-oriented Programming (OOP's) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

Object Oriented Programming is used to solving a problem by creating an object is one of the most approch in programming. this is called Object Oriented Programming.this concept is focus on re-use the code. .

Naming Convention :

there are two naming conventions mostly used in programming languages. we also should use it for naming variable, functions and classes or even for objects.

- camel case : rajendraKumarRanda
- pascal case : RajendraKumarRanda

OOPs Concepts in Python

- Class
- Objects
- Polymorphism

- Encapsulation
- Inheritance
- Data Abstraction

Python Class :

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods. class is a blueprint for creating object, class not store memory it is just a instance.

NOTE : class class_name: # it is written in pascal case

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator. Eg.: Myclass.Myattribute.

SYNTAX : class class_name: statement 1; statement 2; statement 3; .... ; statement n;

```
class myclass:
    name = 'raj'
    age = 23
    place = 'INDIA'
```

python Object :

The object is an entity that has a state and behavior associated with it. It may be any real-world object like a mouse, keyboard, chair, table, pen, etc. Integers, strings, floating-point numbers, even arrays, and dictionaries, are all objects. More specifically, any single integer or any single string is an object. The number 12 is an object, the string "Hello, world" is an object. it is an instantiation of a class,when class is defined,memory is allocated only after object instantiation object of a class can be invoke the method available to it without revealing the implementation details to the usion.

SYNTAX : object_name = class_name() object_name.properties = value

```
student = myclass()

print(student.name)
print(student.age)
print(student.place)

raj
23
INDIA
```

Let's see the classes and their objects

```
# create class
class myclass:
    x = 5

# create object
number = myclass()
print(number.x)
```

```
5
```

- Let's see more examples

```
class RailwayForm:
    formType = "Railway-Form"
    def printData(self):
        print(f"name : {self.name}")
        print(f"train : {self.train}")

# create object
rajForm = RailwayForm()

# insert value for your object
rajForm.name = "raj kumar"
rajForm.train = "xyz express"

# print value of function inside of class for your object
rajForm.printData()

# print class variable
print(RailwayForm.formType)
```

```
name : raj kumar
train : xyz express
Railway-Form
```

elements called in Opps in above example :

- class name : RailwayForm
- class variable : formType
- instance variable : name, train
- object name : rajForm

```
class EmployeeData:
    Company = " VS Code Company "
    def employee(self):
        print(f"Employee Details of '{self.name}' are here : ")
        print(f"NAME : {self.name}")
        print(f"SALARY : {self.salary} ")
        print(f"PLACE : {self.place}")
        print("***********************************")
```

```
# object 1
# creating object of class
emp1 = EmployeeData()

# insert values of class for object 1
emp1.name = "Rajendra"
emp1.salary = "20,000/-"
emp1.place = "INDORE"

# print data for object 1
print(emp1.Company)   # print class variable for object 1
emp1.employee()      # print value using function inside of class for
object 2

# object 2
# creating object of class
emp2 = EmployeeData()

# insert values of class for object 2
emp2.name = "Mohan"
emp2.salary = "15,000/-"
emp2.place = "BHOPAL"

# print data for object 2
print(emp2.Company)   # print class variable for object 1
emp2.employee()       # print value using function inside of class for
object 2
```

```
 VS Code Company
Employee Details of 'Rajendra' are here :
NAME : Rajendra
SALARY : 20,000/-
PLACE : INDORE
**********************************
 VS Code Company
Employee Details of 'Mohan' are here :
NAME : Mohan
SALARY : 15,000/-
PLACE : BHOPAL
**********************************
```

- change the value of class variable

1. for all objects.

```
class EmployeeData:
    Company = " VS Code Company "
    def employee(self):
        print(f"Employee Details of '{self.name}' are here : ")
        print(f"NAME : {self.name}")
        print(f"SALARY : {self.salary} ")
        print(f"PLACE : {self.place}")
        print("**********************************")

# creating objects
emp1 = EmployeeData()
emp2 = EmployeeData()

# it'll change value of variable for every object
EmployeeData.Company = "Google"


# print class variable
print(emp1.Company)
print(emp2.Company)

Google
Google
```

1. for specific object

```
class EmployeeData:
    Company = " VS Code Company "
    def employee(self):
        print(f"Employee Details of '{self.name}' are here : ")
        print(f"NAME : {self.name}")
        print(f"SALARY : {self.salary} ")
        print(f"PLACE : {self.place}")
        print("**********************************")

# creating objects
emp1 = EmployeeData()
emp2 = EmployeeData()

# change value of class variable for every object 1
emp1.Company = "Google"

# change value of class variable for every object 1
emp2.Company = "Facebook"

# print class variable
print(emp1.Company)    # for object 1
print(emp2.Company)    # for object 2

Google
Facebook
```

- example change value using class_name

```
class Employee:
    company = "Google"
    salary = 100
```

```
raj = Employee()
ram = Employee()

raj.salary = 500

print(raj.company)
print(raj.salary)

print(ram.company)
print(ram.salary)

Employee.company = "Yahoo!"

print(raj.company)
print(ram.company)

print(raj.salary)
print(ram.salary)

ram.company = 'youtube'
print(raj.company)
print(ram.company)
```

```
Google
500
Google
100
Yahoo!
Yahoo!
500
100
Yahoo!
youtube
```

- add/ create instance variable

```
class Employee:
    company = "Google"
    salary = 100

raj = Employee()
ram = Employee()

# create instance variable
ram.address = 'INDORE'

print(raj.company)
print(raj.salary)

print(ram.company)
print(ram.salary)
print(ram.address)
```

```
Google
100
Google
100
INDORE
```

- class attribute: attribute defined inside the class
- instance attribute: an attribute that belongs to instance(object) assuming the class from previous example. syntax: raj.salary = 500 #adding instance attribute instance attribute take preference over class attribute during assignment & retrival

SELF: it refers instance of class it is automatically passed with function call from an object. NOTE : after remove 'self ' you'll get error like that ".getSalary()" takes 0 positional arguments but 1 was given.

```
class Employee:
  company = "Google"
  def getSalary(self):
    print("sal is 100K")

harry = Employee()
harry.getSalary() #it means 'employee.getSalary(harry)' <--just for
understanding
```

```
sal is 100K
```

```
class Employee:
  company = "Google"
  def getSalary(self):
    print(f"company : {self.company} \nsalary is {self.salary}")

harry = Employee()

harry.company = 'ABC company'
harry.salary = 100000

harry.getSalary()
```

```
company : ABC company
salary is 100000
```

static method :

sometimes, we need to define a function who don't use/need self parameter so for that we can define them as static method.

```python
class Employee:
  company = "Google"
  def getData(self,sign):
    print(f"company: {self.company} and salary will be {self.salary} \
n{sign}")
  @staticmethod   #it's a decorator to mark greet as a static method
  def greet():
    print("have a good day...")
  @staticmethod   #it's a decorator to mark greet as a static method
  def time():
    print("now, it's 9am")

harry = Employee()

harry.salary = 1000
harry.getData("thanks!")

Employee.greet()
harry.greet()  # same as employee.greet()
harry.time()
```

```
company: Google and salary will be 1000
thanks!
have a good day...
have a good day...
now, it's 9am
```

constructor:

A constructor is a special method of a class or structure in object-oriented programming that initializes a newly created object of that type. Whenever an object is created, the constructor is called automatically.

'__init__': it is a special method which is first run as soon as the object created, it is known as constructor, it takes self args and also take further args.

```python
class Employee:
  company = "Google"
  def __init__(self):
    print("employee is created!")

harry = Employee()  # employee is created!
```

```
employee is created!
```

```python
class Employee:
  company = 'facebook'
  def __init__(self, name, salary, subunit):
    self.name = name
    self.salary = salary
    self.subunit = subunit
    print("employee created!")

  def getDetails(self):
    print(f"name :{self.name}")
    print(f"salary :{self.salary}")
    print(f"subunit :{self.subunit}")

  def getData(self,signature):
    print(f"company is {self.company} and salary will be {self.salary}
\n{signature}")

  @staticmethod
  def greet():
    print("have a good day !!!")

  @staticmethod
  def time():
    print("time : 9am")

raj = Employee("raj",1000, "Youtube")
# raj = Employee()   #error:missing 3 args
raj.getDetails()

raj.getData('Mr. R')

raj.greet()
raj.time()

employee created!
name :raj
salary :1000
subunit :Youtube
company is facebook and salary will be 1000
Mr. R
have a good day !!!
time : 9am
```

NOTE : you dont need to write manually like raj,name = 'raj'. in above example.

Ques. : create a class programmer for storing data of few programmer working on ms.

```
class Programmer:
    def __init__(self,name,product):
        self.name = name
        self.product = product
    def getInfo(self):
        print(f"programmers details: \n name : {self.name} \n
product : {self.product}\n")
```

```
obj1 = Programmer("ram", "skype")
obj2 = Programmer("sita", "git")

obj1.getInfo()
obj2.getInfo()
```

```
programmers details:
 name : ram
 product : skype

programmers details:
 name : sita
 product : git
```

Ques : write a class calculator capable of finding square, cube and squareroot of a number.

```
class Calculator:
    def __init__(self,num):
        self.num = num
    def square(self):
        print(f"square of {self.num} is {self.num**2}")
    def cube(self):
        print(f"cube of {self.num} is {self.num**3}")
    def squareRoot(self):
        print(f"square root of {self.num} is {self.num**0.5}")

a = Calculator(9)

a.square()
a.cube()
a.squareRoot()
```

```
square of 9 is 81
cube of 9 is 729
square root of 9 is 3.0
```

Ques : create a class with a class attribute a ,create an object from it and set directly using object a = 0, does this change class attribute?

```
class Sample:
    a = 10

obj = Sample()

obj.a = 20

print(obj.a)
print(Sample.a)
```

```
20
10
```

REASON : no it'll not change, coz object create instance of class attribute for self.

Ques : add a static method in ques 2 to greet the user with hello [ques 2: write a class calculator capable of finding square, cube and squareroot of a number.]

```python
class Calculator:
    def __init__(self,num):
        self.num = num
    def square(self):
        print(f"square of {self.num} is {self.num**2}")
    def cube(self):
        print(f"cube of {self.num} is {self.num**3}")
    def squareRoot(self):
        print(f"square root of {self.num} is {self.num**0.5}")
    @staticmethod
    def greet():
        print("thanks for using it...!")

a = Calculator(2)

a.square()
a.cube()
a.squareRoot()
a.greet()
```

```
square of 2 is 4
cube of 2 is 8
square root of 2 is 1.4142135623730951
thanks for using it...!
```

Ques. : wriet a class train which has method to book a ticket and eget status(no. of seat) and get fare information of train running under indian railways.

```python
class Train:
  def __init__(self, name, fare, seats):
    self.name = name
    self.seats = seats
    self.fare = fare
    print("have a good day..")
  def info(self):
    print(f"train:{self.name} \nticket fare: {self.fare} \ntotal seat:
{self.seats}")

enterCity = Train("XYZ-express", 90, 300)
enterCity.info()
```

```
have a good day..
train:XYZ-express
ticket fare: 90
total seat:300
```

Ques : can you change self parameter inside a class to something else (say 'sir'),try changing self to "slf","harry" and see the efect.

```
class Sample:
  def __init__(sf,name):   # you can use any keyword in the place of 
'self' like 'sf'
    sf.name = name

ob = Sample("harry")
print(ob.name)
```

```
harry
```

just for fun

```
class Train:
  def __init__(self, name, fare, seats):
    self.name = name
    self.seats = seats
    self.fare = fare
  def status(self):
    print(f"train name:{self.name}\n ticket fare: {self.fare} \n total 
seat:{self.seats}")
  def bookTicket(self):
    if(self.seats>0):
      print(f"ticket booked!! your sit no. is {self.seats}")
      self.seats = self.seats-1
    else:
      print("sorry all ticket are already booked!!!")

enterCity = Train("XYZ-express", 10, 5)
enterCity.status()
enterCity.bookTicket()
enterCity.bookTicket()
enterCity.bookTicket()
enterCity.bookTicket()
enterCity.status()
```

```
train name:XYZ-express
 ticket fare: 10
 total seat:5
ticket booked!! your sit no. is 5
ticket booked!! your sit no. is 4
ticket booked!! your sit no. is 3
ticket booked!! your sit no. is 2
train name:XYZ-express
 ticket fare: 10
 total seat:1
```

Quick Revision : class & object

```
class Student:
    school = 'Python e-Learning Academy'
    def GetData(self):
        print("Student Details :\nNAME : {self.name} \n CLASS :
{self.std}\n")

obj1 = Student()
obj2 = Student()

obj1.name = "raju"
obj1.std = '12th'

obj2.name = "ram"
obj2.std = '10th'

print(obj1.school)
obj1.GetData()

print(obj2.school)
obj2.GetData()

print(Student.school)

Python e-Learning Academy
Student Details :
NAME : {self.name}
 CLASS : {self.std}

Python e-Learning Academy
Student Details :
NAME : {self.name}
 CLASS : {self.std}

Python e-Learning Academy

class Student:
    school = 'Python e-Learning Academy'
    def GetData(self):
        print("Student Details :\nNAME : {self.name} \n CLASS :
{self.std}\n")

obj1 = Student()
obj2 = Student()
```

```python
print(Student.school)
print(obj1.school)
print(obj2.school)

Student.school = "just python"

print(Student.school)
print(obj1.school)
print(obj2.school)

obj2.school = "java T point"

print(Student.school)
print(obj1.school)
print(obj2.school)
```

```
Python e-Learning Academy
Python e-Learning Academy
Python e-Learning Academy
just python
just python
just python
just python
just python
java T point
```

```python
class Mypython:
    lang = 'high-level pro'
    def intro(self):
        print(f"inventor : {self.inv} \n place : {self.place}")
    @staticmethod
    def geet():
        print("this is great programming language used in all world-
wide areas...")

obj1 = Mypython()

obj1.inv = "guido ven russum"
obj1.place = "neitherland USA"

obj1.intro()
obj1.geet()

Mypython.geet()
```

```
inventor : guido ven russum
 place : neitherland USA
this is great programming language used in all world-wide areas...
this is great programming language used in all world-wide areas...
```

```
class Emp:
    def __init__(self,name,position,salary):
        self.n = name
        self.pos = position
        self.sal = salary
        print("\nCOMPANY = XYZ institute pvt LTD\
n********************************")
    def Data(self,place):
        print(f"NAME : {self.n} \nPOSITION : {self.pos} \nSALARY :
{self.sal} \nADDRESS : {place}")
    @staticmethod
    def greet():
        print("---------------------------")
        print("thanks for visiting us...")
        print("---------------------------\n")

e1 = Emp("raj", "manager", "50,000/-")
e1.Data("INDORE")
e1.greet()

e2 = Emp("rahul", "developer", "65,000/-")
e2.Data("BHOPAL")
e2.greet()
```

```

COMPANY = XYZ institute pvt LTD
********************************
NAME : raj
POSITION : manager
SALARY : 50,000/-
ADDRESS : INDORE
---------------------------
thanks for visiting us...
---------------------------


COMPANY = XYZ institute pvt LTD
********************************
NAME : rahul
POSITION : developer
SALARY : 65,000/-
ADDRESS : BHOPAL
---------------------------
thanks for visiting us...
---------------------------
```

INHERITANCE:

creating a new class using an existing or old class with adding or modifying new features.

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

- Parent class : It is the class being inherited from, also called "base class".
- Child class : It is the class that inherits from another class, also called "derived class".

Create a Parent Class : Any class can be a parent class, so the syntax is the same as creating any other class.

SYNTAX :

- create class : class class_name: def **init**(self, var_1, var_2, ..., var_n): self.variable_1 = var_1 self.variable_2 = var_2 self.variable_3 = var_3 .... self.variable_n = var_n

```
  def function_name(self):
      print(self.variable_1, self.variable_2, ........,
self.variable_n)
```

- create and execute the object

  object_1 = class_name(value_1, value_2, value_2, ..., value_n)
  object_1.function_name()

Create a Child Class : a class who holds name of his parenet class as arguments to inherit their features

SYNTAX :

- create child class class child_class_name(parent_class_name): statements_1 statements_1 .... statements_n

```
# parent class
class Employee:
    company = "Python-Pro"
    def showData(self):
        print(f"this an employee of {self.company}...")

# child class
class Programmer(Employee):
    lang = 'python'
    def getLanguage(self):
        print(f"programming skills : {self.lang}")

e = Employee()
p = Programmer()

e.showData()
p.showData()
p.getLanguage()
```

```
this an employee of Python-Pro...
this an employee of Python-Pro...
programming skills : python
```

Error : * parent class class Employee: company = "Python-Pro" def showData(self): print(f"this an employee of {self.company}...")

- child class class Programmer(Employee): lang = 'python' def getLanguage(self): print(f"programming skills : {self.lang}")
- creating object e = Employee() p = Programmer()
- try to access child class feature by parent class object e.getLanguage() --- AttributeError Traceback (most recent call last) Cell In[5], line 19 17 p.showData() 18 p.getLanguage() ---> 19 e.getLanguage()

AttributeError : 'Employee' object has no attribute 'getLanguage'

- Over-riding

```
class Employee:
  company = "Google"
  def showDetails(self):
    print(f"this is an employee..")

class Programmer(Employee):
  language = 'python'
  company = 'Yahoo!'          #overriding
  def getlanguage(self):
    print(f"language is {self.language}")
  def showDetails(self):
    print("i am programmer...")   #overriding

e = Employee()
p = Programmer()

e.showDetails()
p.showDetails()

print(e.company)
print(p.company)

# e.getlanguage() #AttributeError: 'Employee' object has no attribute
'getlanguage'
p.getlanguage()
```

```
this is an employee..
i am programmer...
Google
```

```
Yahoo!
language is python
```

Inheritance Types :

There are 5 different types of inheritance in Python.They are: 1. Single Inheritance : a child class inherits from only one parent class. 2. Multiple Inheritance : a child class inherits from multiple parent classes. 3. Multilevel Inheritance : a child class inherits from its parent class, which is inheriting from its parent class. 4. Hierarchical Inheritance : more than one child class are created from a single parent class. 5. Hybrid Inheritance : combines more than one form of inheritance.

1. single inheritance:The inheritance in which a single derived class is inherited from a single base class is known as the Single Inheritance.

```
class
```

2. multiple inheritance:

```

```

3. multilevel inheritance:

```

```

- super()
- class()
- decorator
- operator overloading

define adv python

exception handling :

file handling:

we learn here to handle text files and folders, connect with databases and work with command mode.

we will see how to create, detele and rename any txt and excel file and also edit content written inside of files.

- files I/O : YOU CAN INTERACT WITH FILES(read,write) VIA PY-CODE theory:

FILE: collection of data to store data in computer on server or external storages.

1. VOLATILE-> file stored in ram like if you are playing games,
2. NON VOLATILE: data stored in HDD or pen-drive

TYPES OF TXT FILE:

1. TEXT FILE(.txt, .c)
2. BINARY FILE(.jpg, .dot)

open(): it is build-in func., used to opening a file,it takes 2 parameter (file name, mode) i.e.open("sample.txt","r")

USED FILE Formate : "data1.txt"

MODE: there are various mode of open() function used in text files.

SYNTAX :

1. f = open("txt file name with path")
2. with open("txt file name with path") as object_name:

close():

- it is used to close the text file.
- kindly always close the file in the end after using file.

SYNTAX : object_name.close()

NOTE :

- 'r' is default mode in open().
- you no need to use close() if you are handling txt files using "with" keyword coz after working it'll automatically closed.

- ## read txt file :

To open the file, use the built-in function named open() . The open() function returns a file object, which has a read() method for reading the content of the file.

There two mode used to read the files :

r : read mode to read the file rt : read mode to read the txt file

you can read content using read() and in opent() you can file name or file name with path and extention and mode of open()

SYNTAX : declared_variable = object_name.close()

- there is also 2 ways to read the files.

1. default way ````:

```
f = open('poem.txt')
data = f.read()
print(data)
f.close()

Twinkle, Twinkle, Little Star
BY JANE TAYLOR
.......................................
Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
```

```
Like a diamond in the sky.

When the blazing sun is gone,
When he nothing shines upon,
Then you show your little light,
Twinkle, twinkle, all the night.

Then the traveler in the dark
Thanks you for your tiny spark,
How could he see where to go,
If you did not twinkle so?

In the dark blue sky you keep,
Often through my curtains peep
For you never shut your eye,
Till the sun is in the sky.

As your bright and tiny spark
Lights the traveler in the dark,
Though I know not what you are,
Twinkle, twinkle, little star.

Source: The Golden Book of Poetry (1947)
```

1. via using path :

```
f = open('C://Users//rajen//Downloads//0. data science//poem.txt','r')
data = f.read()
print(data)
f.close()

Twinkle, Twinkle, Little Star
BY JANE TAYLOR
.......................................
Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.

When the blazing sun is gone,
When he nothing shines upon,
Then you show your little light,
Twinkle, twinkle, all the night.

Then the traveler in the dark
Thanks you for your tiny spark,
How could he see where to go,
If you did not twinkle so?

In the dark blue sky you keep,
Often through my curtains peep
For you never shut your eye,
Till the sun is in the sky.

As your bright and tiny spark
Lights the traveler in the dark,
Though I know not what you are,
Twinkle, twinkle, little star.

Source: The Golden Book of Poetry (1947)
```

1. via using 'with' keyword :

```
with open('poem.txt') as f:
    data = f.read()
    print(data)

Twinkle, Twinkle, Little Star
BY JANE TAYLOR
.......................................
Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.

When the blazing sun is gone,
When he nothing shines upon,
Then you show your little light,
Twinkle, twinkle, all the night.

Then the traveler in the dark
Thanks you for your tiny spark,
How could he see where to go,
If you did not twinkle so?

In the dark blue sky you keep,
Often through my curtains peep
For you never shut your eye,
Till the sun is in the sky.

As your bright and tiny spark
Lights the traveler in the dark,
Though I know not what you are,
Twinkle, twinkle, little star.

Source: The Golden Book of Poetry (1947)
```

- using loop

```
f = open("dummy.txt")
data = f.read()
```

```
for x in data:
    print(data)
    f.close()
```

```
this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4

this is line 1
this is line 1
this is line 1
this is line 4
```

```python
f = open('dummy.txt')
data = f.read()
for x in data:
    print(x)
    f.close()
```

```
t
h
i
s

i
s

l
i
n
e

1


t
h
i
s

i
s

l
i
n
e

1


t
h
i
s

i
s
```

```
l
i
n
e

1


t
h
i
s

i
s

l
i
n
e

4
```

- readline() : read the file line by line

```
f = open("dummy.txt", "r")
print(f.readline())
f.close()
```

```
this is line 1
```

```
f = open("dummy.txt", "r")
print(f.readline())
print(f.readline())
print(f.readline())
print(f.readline())
print(f.readline())
print(f.readline())
print(f.readline())
print(f.readline())
print(f.readline())
print(f.readline())
print(f.readline())
print(f.readline())
f.close()
```

```
this is line 1

this is line 1

this is line 1

this is line 4
```

```
data = open("dummy.txt")
print(data.readline())
x = data.readline()
print("1. ",x)
y = data.readline()
print("2. ",y)
z = data.readline()
print("3. ",z)
data.close()
```

```
this is line 1

1.  this is line 2

2.  this is line 3

3.  this is line 4
```

- normal code and without mention 'r' in open()

```
f = open('demo1.txt','r')
f1 = open('demo2.txt')
data1 = f.read()
data2 = f1.read()
print(data1)
print(data2)
f.close()
f1.close()
```

```
this is demo 1

this is demo 2
```

- write the txt file

it's use to write in any text file but it is replace the data from new one to old or already written data.

there are mostly 3 mode used to write in the text file w : write mode to write the file wt : write mode to write the txt file a : append mode to write after content the file

- first write in existing txt file than check it by read() funtion/ reading it.

1. write into txt file

```
f = open("new.txt", 'w')
data = f.write("you are welcome")
f.close()

f = open("new.txt", 'r')
data = f.read()
print(data)
f.close()

you are welcome
```

- if you use write() function again then it'll rewrite content

```
f = open("new.txt","w")
data = f.write("This is text file where we'll write using python file
handling write() function. ")
f.close()
f = open("new.txt","r")
data = f.read()
print(data)
f.close()

This is text file where we'll write using python file handling write()
function.
```

- in new.txt file already written 'you are welcome' and after using write() it replaced with 'this is text file where we'll write using python file handling write() function.'
- if in text file is already written then we must use 'a' append mode then our new content will write after already existing content, but in write mathod write() replace content.

```
f = open('new.txt','a')
data = f.write('This is content of second times.')
f.close()
f = open('new.txt', 'r')
data = f.read()
print(data)
f.close()

This is text file where we'll write using python file handling write()
function. This is content of second times.
```

- let's write in exmpty text file and than try to read it

```
with open('demo.txt','w') as f:
    f.write('this is style 2.')

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

```
this is style 2.
```

- file handling with "with" keyword.
- create txt file (way 1)

```
f = open("with.txt")
data = f.read()
f.close()
```

- create txt file (way 2)

```
with open("with.txt", 'r') as f:
    data = f.read()
```

- write into empty txt file (way 1)

```
f = open("with.txt", 'w')
data = f.write("hey welcome this is empty text file to write")
f.close()
```

- write into empty txt file (way 2)

```
with open("with.txt", 'w') as f:
    data = f.write("hey welcome this is empty text file to write")
```

- read the file (way 1)

```
f = open("with.txt")
data = f.read()
print(data)
f.close()
```

```
hey welcome this is empty text file to write
```

- read the file (way 2)

```
with open("with.txt") as f:
    data = f.read()
    print(data)
```

```
hey welcome this is empty text file to write
```

- write data in already written file (way 1)

```python
f = open('with.txt','a')
data = f.write('this li line 2, ')
data = f.write('this is line 3')
data = f.close

f = open("with.txt","r")
data = f.read()
print(data)
f.close()
```

```
hey welcome this is empty text file to write
```

- write data in already written file (way 2)

```python
with open('with.txt','a') as f:
    data = f.write('this li line 4,')
    data = f.write('this is line 5')
    data = f.close

with open("with.txt","r") as f:
    data = f.read()
    print(data)
```

```
hey welcome this is empty text file to writethis li line 2, this is
line 3this li line 4, this is line 5this li line 4,this is line 5
```

Ques. : WAP to find out that given string exist in text file or not.

- find any data or information in txt file

```python
word = input("enter word to check weather is exist in poem 'the
star' : ")
with open('poem.txt','r') as f:
    data = f.read()
    if word in data:
        print("word matched...")
    else:
        print("word doesn't exist...")
```

```
enter word to check weather is exist in poem 'the star' :  twinkle

word matched...
```

Ques. : WAP to generate table of 2-20 and save it in different-different txt files. NOTE : please check files in folder for making sure.

```python
for i in range(2,21):
    with open(f"table of {i}.txt", 'w') as f:
        for j in range(1,11):
            f.write(f"{i} X {j} = {i*j}")
            if j!=10:
                f.write("\n")
```

Ques. : WAP to generate table of 2-20 and save it in different-different txt files and save this text file in specific folder. NOTE : please check files in folder for making sure.

```
for i in range(2,21):
    with open(f"E:\\1. Data Science\\0. Self Learning\\1.1. 
PROGRAMMING WITH PYTHON\\\\NOTEBOOKS\\table txt file\\
table_of_{i}.txt","w") as f:
        for j in range(1,11):
            # f.write(f"{i}X{j}={i*j}\n") #way 1 but it'll add new 
line in line no.11
            f.write(f"{i} X {j} = {i*j}")
            if j!=10:
                f.write('\n')
```

Ques. : WAP to replace any word or string in txt file everywhere is that words written and how many times. NOTE : create a txt file names 'sample.txt' with write specific string in multiple time for using in code with following paragraph- "this is john. john is lazy and dumb person. he don't complete work on time coz he is lazy. lazy people are not good fit for team work. he losts his job many time because he is so lazy. no one like lazy people."

```
with open("sample.txt") as f:
    data = f.read()
    data = data.replace("lazy", "l***")
    with open("sample1.txt", 'w') as f:
        data = f.write(data)

with open("sample1.txt") as f:
    data = f.read()
    print(data)
```

```
this is john.
john is l*** and dumb person.
he don't complete work on time coz he is l***.
l*** people are not good fit for team work.
he losts his job many time because he is so l***.
no one like l*** people.
```

Ques. : WAP to replace/ hide words exists in list from txt file. OR WAP to hide words from collection in given txt file.

here is collection of words : ("Damn, dumb, Hell, fool") hide them using "*******"

hide.txt file papagraph :

Well, damn, that was a dumb move! I can't believe I fell for that fool's trick. Now, I'm stuck in this hell of a situation. It's a reminder not to underestimate people, even if they seem like complete fools. Sometimes, appearances can be deceiving, and I've landed myself in quite a predicament.

Hell, I'll have to find a way out of this mess and not let my guard down again.Well, damn, that was a dumb move! I can't believe I fell for such a foolish trick. Now, I find myself in a situation that feels like hell. It's a stark reminder that sometimes impulsive decisions can lead to regrettable outcomes. Next time, I'll be more cautious and avoid acting like a fool.

```
words =  ["Damn", "dumb", "Hell", "fool"]
with open("hide.txt", 'r') as f:
    data = f.read()

for word in words:
    data = data.replace(word, "*******")
    with open("hide.txt", 'w') as f:
        f.write(data)

with open("hide.txt", 'r') as f:
    data = f.read()
    print(data)
```

```
Well, damn, that was a ******* move! I can't believe I fell for that
*******'s trick. Now, I'm stuck in this hell of a situation. It's a
reminder not to underestimate people, even if they seem like complete
*******s. Sometimes, appearances can be deceiving, and I've landed
myself in quite a predicament. *******, I'll have to find a way out of
this mess and not let my guard down again.Well, damn, that was a
******* move! I can't believe I fell for such a *******ish trick. Now,
I find myself in a situation that feels like hell. It's a stark
reminder that sometimes impulsive decisions can lead to regrettable
outcomes. Next time, I'll be more cautious and avoid acting like a
*******.
```

Ques. : WAP to create and maintain a txt file named highscore.txt with following condition -

the game() in a program lets a user play a game and returns a score as an integer. you need to read the text file highscore.txt which is either blank or contains the previous highscore whenever game() break the highscore.

def game(x): return x

x = int(input("enter your score : ")) score = game(200)

with open('highscore.txt','r') as f: newscor = f.read()

if int(newscore) < score : with open('highscore.txt','w') as f: data = f.write('score') elif newscore == '': with open('highscore.txt','w') as f: data = f.write(score)

Quiz : WAP to mine a log file and findout whether it contains 'python' and make new code to find out line number.

```
with open("log.txt") as f:
    data = f.read().lower()
if "python" in data:
    print("words exists....")
else:
    print("words exists....")

words exists....
```

- now let's find out where that words exists in file

```
data = True
i = 1

with open("log.txt") as f:
    while data:
        data = f.readline()
        "python" in data.lower()
        print(data)
        print(f'\nstring present in file on line no,: {i}')
        i = i+1

03/22 08:51:06 INFO   :.....mailslot_create: creating mailslot for
RSVP


string present in file on line no,: 1
03/22 08:51:06 INFO   :....mailbox_register: mailbox allocated for
rsvp python has stopped working.....


string present in file on line no,: 2
03/22 08:51:06 INFO   :.....mailslot_create: creating mailslot for
RSVP via UDP


string present in file on line no,: 3
03/22 08:51:06 INFO   :....mailbox_register: mailbox allocated for
rsvp-udp


string present in file on line no,: 4
03/22 08:51:06 TRACE  :..entity_initialize: interface 127.0.0.1,
entity for rsvp allocated and initialized


string present in file on line no,: 5
03/22 08:51:06 INFO   :......mailslot_create: creating socket for
querying route


string present in file on line no,: 6
03/22 08:51:06 INFO   :.....mailbox_register: no mailbox necessary for
forward
```

```
string present in file on line no,: 7
03/22 08:51:06 INFO   :......mailslot_create: creating mailslot for
route engine - informational socket


string present in file on line no,: 8
03/22 08:51:06 TRACE  :......mailslot_create: ready to accept
informational socket connection


string present in file on line no,: 9
03/22 08:51:11 INFO   :.....mailbox_register: mailbox allocated for
route        python has stopped working.....


string present in file on line no,: 10
03/22 08:51:11 INFO   :.....mailslot_create: creating socket for
traffic control module


string present in file on line no,: 11
03/22 08:51:11 INFO   :....mailbox_register: no mailbox necessary for
traffic-control


string present in file on line no,: 12
03/22 08:51:11 INFO   :....mailslot_create: creating mailslot for RSVP
client API


string present in file on line no,: 13
03/22 08:51:11 INFO   :...mailbox_register: mailbox allocated for
rsvp-api


string present in file on line no,: 14
03/22 08:51:11 INFO   :...mailslot_create: creating mailslot for
terminate


string present in file on line no,: 15
03/22 08:51:11 INFO   :..mailbox_register: mailbox allocated for
terminate


string present in file on line no,: 16
03/22 08:51:11 INFO   :...mailslot_create: creating mailslot for dump
```

```
string present in file on line no,: 17
03/22 08:51:11 INFO   :..mailbox_register: mailbox allocated for dump


string present in file on line no,: 18
03/22 08:51:11 INFO   :...mailslot_create: creating mailslot for
(broken) pipe


string present in file on line no,: 19
03/22 08:51:11 INFO   :..mailbox_register: mailbox allocated for pipe


string present in file on line no,: 20
 07


string present in file on line no,: 21
03/22 08:51:11 INFO   :.main: rsvpd initialization complete


string present in file on line no,: 22
 08


string present in file on line no,: 23
03/22 08:52:50 INFO   :......rsvp_api_open: accepted a new connection
for rapi


string present in file on line no,: 24
03/22 08:52:50 INFO   :.......mailbox_register: mailbox allocated for
mailbox


string present in file on line no,: 25
03/22 08:52:50 TRACE  :......rsvp_event_mapSession:
Session=9.67.116.99:1047:6 does not exist


string present in file on line no,: 26
 09


string present in file on line no,: 27
03/22 08:52:50 EVENT  :.....api_reader: api request SESSION


string present in file on line no,: 28
 10
```

```
string present in file on line no,: 29
03/22 08:52:50 TRACE  :......rsvp_event_establishSession: local node
will send   python has stopped working.....


string present in file on line no,: 30
03/22 08:52:50 INFO   :........router_forward_getOI: Ioctl to get
route entry successful


string present in file on line no,: 31
03/22 08:52:50 TRACE  :........router_forward_getOI:         source
address:   9.67.116.98


string present in file on line no,: 32
03/22 08:52:50 TRACE  :........router_forward_getOI:         out inf:
9.67.116.98


string present in file on line no,: 33
03/22 08:52:50 TRACE  :........router_forward_getOI:         gateway:
0.0.0.0


string present in file on line no,: 34
03/22 08:52:50 TRACE  :........router_forward_getOI:         route
handle:   7f5251c8 python has stopped working...


string present in file on line no,: 35
 11


string present in file on line no,: 36
03/22 08:52:50 TRACE  :.......event_establishSessionSend: found
outgoing if=9.67.116.98 through python has stopped working.....


string present in file on line no,: 37
forward engine


string present in file on line no,: 38
03/22 08:52:50 TRACE  :......rsvp_event_mapSession:
Session=9.67.116.99:1047:6 exists


string present in file on line no,: 39
 12
```

```
string present in file on line no,: 40
03/22 08:52:50 EVENT  :.....api_reader: api request SENDER


string present in file on line no,: 41
 13


string present in file on line no,: 42
03/22 08:52:50 INFO   :.......init_policyAPI: papi_debug:  Entering


string present in file on line no,: 43



string present in file on line no,: 44
03/22 08:52:50 INFO   :.......init_policyAPI: papi_debug:  papiLogFunc
= 98681F0 papiUserValue = 0


string present in file on line no,: 45



string present in file on line no,: 46
03/22 08:52:50 INFO   :.......init_policyAPI: papi_debug:  Exiting


string present in file on line no,: 47



string present in file on line no,: 48
03/22 08:52:50 INFO   :.......init_policyAPI: APIInitialize:  Entering


string present in file on line no,: 49



string present in file on line no,: 50
03/22 08:52:50 INFO   :.......init_policyAPI: open_socket:  Entering


string present in file on line no,: 51



string present in file on line no,: 52
```

```
03/22 08:52:50 INFO   :.......init_policyAPI: open_socket:  Exiting


string present in file on line no,: 53



string present in file on line no,: 54
03/22 08:52:50 INFO   :.......init_policyAPI: APIInitialize:
ApiHandle = 98BDFB0,  connfd = 22


string present in file on line no,: 55



string present in file on line no,: 56
03/22 08:52:50 INFO   :.......init_policyAPI: APIInitialize:  Exiting


string present in file on line no,: 57



string present in file on line no,: 58
03/22 08:52:50 INFO   :.......init_policyAPI: RegisterWithPolicyAPI:
Entering   python has stopped working.....

string present in file on line no,: 59


string present in file on line no,: 60
```

copy text file:

Quiz : WAP to make copy file of text file. make a copy of poem.txt in a new text file named copy.txt

```python
with open('poem.txt') as f:
    data = f.read()
with open('copy.txt','w') as f:
    data = f.write(data)
```

compare text file data:

Quiz : WAP check any 2 or more txt files have same data or not. OR WAP check copied txt file copy 100% or not. OR WAP to check two file are identical or not (file's content matched or not)

```python
with open('copy.txt','r') as f:
    data1 = f.read()
```

```
with open('poem.txt','r') as f:
    data2 = f.read()
if data1 == data2:
    print('files are identical...')
else:
    print("files aren't identical...")
```

```
files are identical...
```

rename the text file :

Quiz : WAP to Rename text file.

```
import os

newname = "rename.txt"
oldname = "font.txt"

with open(oldname, 'r') as f:
    data = f.read()
with open(newname, 'w') as f:
    data = f.write(data)
os.remove(oldname)
```

NOTE : To delete any file or folder, you must import the OS module, and run its os.remove(args) function.

delete the file

```
import os
os.remove('delete.txt')
```

- check if file exist then delete if not then show msg.

```
import os
if os.path.exists('delete.txt'):
    os.remove('delete.txt')
    print("file deleted...")
else:
    print("file doesn't exist...")
```

```
file doesn't exist...
```

delete folder

Note: You can only remove empty folders.

```
import os
os.rmdir('delete')
```

- if you try to delete any directories(folder) who hold any files than you'll get following error.

upcoming new chapters:

- map(), filter() & reduce()
- f'string'
- list comprehensive
- enumenator
- global variable
- PIP
- virtual environment

use this youtube playlist in revision time to add skipped features

References

Official Portals :

1. Python
2. Playsound
3. OpenCV

Websites :

1. W3School
2. geeks for geeks
3. jupyter markdown

Youtube Playlists :

1. Code with Daneyal Lari
2. Code with Herry

tools :

- Convert jupyter noterbook to PDF