TUTOR JOES VIDEO TUTORIAL

1] GETTING INPUT IN PYTHON:

- In python if we get input,by default it will take it as a STRING type. So, if we want to get INTEGER, FLOAT or BOOLEAN data type, then we have to specifically give the particular data type before INPUT command.

ex:

```
main.py ×
1  a=int(input("enter a: "))
2  b=float(input("enter b: "))
3  c=a+b
4  print(c)
5  print(type(a))
6  print(type(b))
```

```
Console × Shell ×
enter a: 10
enter b: 10.5
20.5
<class 'int'>
<class 'float'>
>
```

- In python we can get the values of all variables in a SINGLE LINE. In this we use **split function[*.split()*]**.By default, It will take the space and show the values separately. If we want the split function to take (,) as the separated view, then we give [***.split(,)***].

Multi Line String:

- To declare Multi line string we have to give it inside triple coats(“”””…””””).

Getting Multi Line Input in Python:

- For this, We use LIST concept.
- We also use JOIN function(*.join*), to remove the square bracket or Comma in list.

```
main.py ×
1  para=[]
2  print("Enter a para: ")
3
4  while True:
5      line=input()
6      if line:
7          para.append(line)
8      else:
9          break
10 print(para)
11 output='\n'.join(para)
12
13 print(output)
```

```
Console × Shell ×
Enter a para:
mani is an engineer.
he is in hometown.
he loved his family.

['mani is an engineer.', 'he is in hometown.', 'he loved his
 family.']
mani is an engineer.
he is in hometown.
he loved his family.
>
```

2] TYPE CASTING:

3] STRING:

- ***Split lines***- it will split each line as separate list element.(*.splitlines()*).
- To split the line using **space** we use (*.split(“ “)*).
- ***STRIP FUNCTION*** - To remove the unwanted space while counting the length of the string.
- ***PARTITION*** Concept.
- ***STRING SLICING -*** It fully based on index values.

4] ARITHMETIC OPERATORS:

5] ASSIGNMENT OPERATORS:

=Assignment +=Addition -=Subtraction *=Multiplication /=Division %= Modulus **= Exponentiation //= Floor Division

6] COMPARISON OR RELATIONAL OPERATORS:

== Equal - check whether both side values are equal.*!= Not equal* - check both side values are not equal.*> greater than< less than>=greater than or equal to<= less than or equal to*

7] LOGICAL OPERATORS:

andorNot

```
a=25
print(a>=10 and a<=20)
print(a>=10 or a<=20)
print(not(a>=10 and a<=20))
```

```
False
True
True
>
```

8] IF STATEMENT:

9] LOOP CONCEPT:

10] CONTINUE STATEMENT:

https://www.tutorjoes.in/python_programming_tutorial/continue_using_while_loop_in_python

11] BREAK STATEMENT:

https://www.tutorjoes.in/python_programming_tutorial/break_using_while_loop_in_python

12] RANGE:

We can use Range using LIST and FOR LOOP also.
https://www.tutorjoes.in/python_programming_tutorial/range_in_python

13] FOR LOOP:

https://www.tutorjoes.in/python_programming_tutorial/for_loop_in_python

In for loop we use RANGE concept..

- Nested For Loop.

14] LIST:

Changeable.

LIST INBUILT FUNCTIONS:

.copy() - it will copy the list value to another list. ***.count()*** - It will count the occurrence of particular value in the list. ***.index()*** - it will print the value of the particular index using index number. ***len()*** - it will print the total number of values in the list.***max()*** - print maximum value in the list.***min()*** - print minimum value in the list..***pop()*** - remove elements using index value..***remove()*** - remove using the particular value..***append()*** - add new values to the list..***extend()*** - used to join two lists..***insert()*** - insert value at the particular index using index number.

LIST TYPE CONSTRUCTORS:

If we pass any value using list constructor it will automatically change it to List.**Eg:** print(list(range(5)))

OUTPUT: [0,1,2,3,4]

.sort().reverse()

15] TUPLE:

- Immutable.
- Surrounded by Round Bracket.
- Have index also.
- Have Negative index also.
- We can access the range also.
- ***We can't add or change any value inside the Tuple.For that, first we have to change the Tuple to List using List constructor and then we can add new value and again we can change the list to Tuple using Tuple constructor.***
- ***If we use single item inside the tuple, then we have to give that value with(,). Otherwise it will consider that value as Integer.***

EXAMPLE: ***a = (1,)***
print(type(a))
OUTPUT: <class 'tuple'>

TUPLE CONCADINATION:

Using '+' operator we can concadinate two tuples.

NESTED TUPLE:

```
a=(1,2,3,4)
b=(5,6,7,8)
c=(a,b)
print(c)
print(c[0])
print(c[1])
print(c[0][1])
```

```
((1, 2, 3, 4), (5, 6, 7, 8))
(1, 2, 3, 4)
(5, 6, 7, 8)
2
```

16] SET:

- Collection of Unordered and indexed datatype.
- Duplicate values not allowed.
- We use Curly brackets for Set *'{}'*.

.add().update().remove().discard().pop().clear().disjoint().joint()

17] DICTIONARY:

- We use Curly brackets for Dictionary.
- It consists of KEY and VALUES.
- In Dictionary we can access the values using their Key names.

Syntax to add new values to the dictionary:

Name of dictionary.update({'new key':'new value'})Print(Name of dictionary)

We can change the values also:

Name of dictionary['key']=new value

Pop function - it will completely remove the particular key and value.

We can use Multiple dictionary using Nested dictionary

```python
users={
  "user1": {
    "name":"mani",
    "age":23,
    "ismarried":"true"
  },

  "user2": {
    "name":"maddy",
    "age":22,
    "ismarried":"false"
  }
}
print(users)
```

```
{'user1': {'name': 'mani', 'age': 23, 'ismarried': 'true'
}, 'user2': {'name': 'maddy', 'age': 22, 'ismarried':
lse'}}
```

18] IDENTITY OPERATOR:

- Used to compare two objects for their Equality.

isis not

19] MEMBERSHIP OPERATOR:

- To Check whether the particular value we want is present inside the LIST/SET/TUPLE/DICTIONARY we use MEMBERSHIP FUNCTION.

```python
a=[1,2,3,4]
print(2 in a)
print(6 in a)
print(10 not in a)
```

```
True
False
True
```

20] FUNCTIONS IN PYTHON:

There are NINE types of functions are there:

1) No Return type Without argument function in python:

```python
def add():
    a = int(input("Enter a :"))
    b = int(input("Enter b :"))
    c = a+b
    print(c)
add()
```

```
Enter a :4
Enter b :6
10
```

2) No Return type With argument function:

main.py

```python
def add(a,b):
    c = a+b
    print(c)
add(10,20)
```

```
30
```

3) Return type Without argument function:

main.py

```python
def add():
    a = int(input("Enter a :"))
    b = int(input("Enter b :"))
    c = a+b
    return c
o = add()
print(o)
```

```
Enter a :3
Enter b :5
8
```

4) Return type With argument function:

```python
def add(a,b):
    c = a+b
    return c
o = add(10,6)
print(o)
```

```
16
```

5) Arbitrary Arguments Function in Python:

- We can pass ‘n’ number of values in the argument.
- For this we have to use ‘*’ symbol.

```python
def class_10(*students):
  print(students)
  for user in students:
    print(user)

class_10("Mani","Renu","Pavi")
```

```
('Mani', 'Renu', 'Pavi')
Mani
Renu
Pavi
```

6) Keyword Argument function in python:

- Here we use the variable name (in argument) as the key word to give the values.

```python
def message(name,age):
  print(name, "__age is__" ,age)

message(age=23,name="Mani") #here it will assingn the value to the appropriate argument eventhough we give the values by chhanging the arguments position in the print function
```

```
Mani __age is__ 23
```

7) Arbitrary Keyword Argument function in python:

- Here we have to use’**’.

```python
def biodata(**data):
  print(data)


biodata(NAME="MANI",AGE=23,GENDER="MALE")
```

```
{'NAME': 'MANI', 'AGE': 23, 'GENDER': 'MALE'}
```

```python
def biodata(**data):
  print(data)


biodata(NAME="MANI",AGE=23,GENDER="MALE")
```

```
{'NAME': 'MANI', 'AGE': 23, 'GENDER': 'MALE'}
```

8) Default parameter function in python:

- If user give value to the parameter, then it have to take that value. *If user doesn't give any value to the parameter , it have to take default value. For that we use this one.*

```python
def user(name,city="salem"):
  print(name,"is from",city)

user("Mani","Madurai")#here the user give the value to the parameter,.So, it will take that value.
user("Siva")#here the user doesn't give the value. So, it will take the default value.
```

```
Mani is from Madurai
Siva is from salem
```

9) Passing a LIST as an Argument in function python:

- Here we can pass the list as a argument to perform the function.

main.py

```python
def total(marks):
  return sum(marks)

print(total([78,59,90,98,88]))
```

total(marks)

Console

```
413
```

21] RECURSIVE FUNCTION:

- To complete the work if the function call itself, it's called as recursive function.

```python
def factorial(x):
  if x==1:
    return 1
  else:
    return(x*factorial(x-1))

print("FACTORIAL :",factorial(5))
```

```
FACTORIAL : 120
```

22] LAMBDA FUNCTION:

23] TRY BLOCK IN PYTHON:

Two types of errors are there.

- COMPILE TIME ERROR.
- RUN TIME ERROR.

Example: ***try:***

a=10/0

Except Exception as e:

Print(e)

Output: division by zero

24] TRY EXCEPT ELSE IN PYTHON:

```
try:
  a=10/25
except Exception as e:
  print(e) #if this program has ny exception means it will execute this exception part.
else:
  print("A value :",a) #if this program doesb't have exception means it will exceute this else part.
```

```
A value : 0.4
```

25] TRY EXCEPT ELSE FINALLY IN PYTHON:

```
try:
  a=10/0
except Exception as e:
  print(e) #if this program has ny exception means it will execute this exception part.
else:
  print("A value :",a) #if this program doesb't have exception means it will exceute this else part.
finally:
  print("Thank You") #Whether there is exception or no exception, whatever situation this finally block will run.
```

```
division by zero
Thank You
```

26| TYPES OF EXCETION IN PYTHON:

NAME ERROR EXCEPTION:

```
try:
    print(a)
except NameError as e:
    print("a is not defined") #here in try block we give print(a. but we didn't define "a" previously. so, it will give nameerror as "a" is not defined)
```

```
a is not defined
```

ZERO ERROR:

```
try:
    print(10/0)
except Exception as e:
    print("denominator can't be zero") #here if we divide anything by zero it will give zerodivisionerror.
```

```
denominator can't be zero
```

VALUE ERROR:

```
try:
    a=int(2.0) #here if we give float value, then using type conversion it will automatically change that float value to integer value. So no error will occur.
    print(a)
except Exception as e:
    print("denominator can't be zero")
```

```
2
```

```
try:
    a=int("mani")
except ValueError as e:
    print("please enter numbers only")  # here in try block we give String value. In type conversion String value cannot be converted into Integer value. So, In that case it will throw an error, that is called Value error.
```

```
please enter numbers only
```

INDEX ERROR:

```python
try:
    a=[5,6,7,8]
    print(a[0])
    print(a[9])
except IndexError as e:
    print("Invalid index, index is not present in the list")  #
```

```
5
Invalid index, index is not present in the list
>
```

FILE NOT FOUND EXCEPTION:

```python
try:
    f=open("test.txt")
except FileNotFoundError:
    print("File not found")
else:
    print(f.read)
```

```
File not found
>
```

27] HANDLING MULTIPLE EXCEPTIONS IN PYTHON:

```python
try:
    A=10/20
    print(A)
    B=[1,2,3,4]
    print(B[8])
except ZeroDivisionError:
    print("Denominator cannot be zero")
except IndexError:
    print("invalid index")
```

```
0.5
invalid index
>
```

PYTHON OOPS CONCEPT

CLASS AND OBJECT:

- To check whether the particular object is created for any particular class, We use:

Print(isinstance(object name.class name))

- To check the type of the object, We use:

Print(type(object name))

1] CLASS ATTRIBUTES:

```python
class Student():
    name = "Mani"
    age = 23

#GETATTRIBUTE METHOD
print(getattr(Student, 'name'))#by using this getattributemethod we can get the value of the attributes in tha class.
print(getattr(Student, 'age'))
print(getattr(Student, 'gender', 'No such attribute is found'))#here, gender attribute is not in the class. So, it will give ATTRIBUTE ERROR. So, we can give no such attribute is found.
print("____________________________________________________")
#DOT NOTATION
print(Student.name)
print(Student.age)
print("____________________________________________________")
#TO CHANGE ANY ATTRIBUTE VALUES IN THE CLASS WE USE SETATTRIBUTE METHOD
setattr(Student,'name','Maddy')
print(Student.name)
setattr(Student,'gender','Male')#WE CAN ADD NEW ATTRIBUTES TO THE CLASS BY SETATTRIBUTE METHOD.
```

```
Mani
23
No such attribute is found
____________________________________________________
Mani
23
____________________________________________________
Maddy
Male
```

```python
Student.city = 'Salem' #WE CAN ADD NEW ATTRIBUTES USING DOT NOTATION ALSO.
print(Student.city)

print(Student.__dict__)#WE CAN CHECK THE DICTIONARY FORMAT OF THE CLASS.

delattr(Student,"city") #WE CAN DELETE THE ATTRIBUTE IN THE CLASS USING THIS DELATTRIBUTE COMMAND.
print(Student.__dict__)
#WE CAN DELETE THE ATTRIBUTE USING DOT NOTATION ALSO.
del Student.gender
print(Student.__dict__)
```

```
--------------------------------------------------
Mani
23

--------------------------------------------------
Maddy
Male
Salem
{'__module__': '__main__', 'name': 'Maddy', 'age': 23, '_
_dict__': <attribute '__dict__' of 'Student' objects>, '_
_weakref__': <attribute '__weakref__' of 'Student' object
s>, '__doc__': None, 'gender': 'Male', 'city': 'Salem'}
{'__module__': '__main__', 'name': 'Maddy', 'age': 23, '_
_dict__': <attribute '__dict__' of 'Student' objects>, '_
_weakref__': <attribute '__weakref__' of 'Student' object
s>, '__doc__': None, 'gender': 'Male'}
{'__module__': '__main__', 'name': 'Maddy', 'age': 23, '_
_dict__': <attribute '__dict__' of 'Student' objects>, '_
_weakref__': <attribute '__weakref__' of 'Student' object
s>, '__doc__': None}
>
```

2] INSTANCE ATTRIBUTE IN PYTHON:

```python
class user:
  course = "java"

o = user() #THIS IS HOW WE HAVE TO CREATE OBJECT FOR THE CLASS.
#here, in object 'o' is called as the INSTANCE.
print(user.__dict__)
print(user.course)#PRINT CLASS ATTRIBUTE.

print(o.__dict__)#IT WILL PRINT THE NAMESPACE OF THE INSTANCE 'o'.
print(o.course)#this is called instance attribute. #USING THIS WE CAN PRINT THE COURSE VALUE USING THE INSTANCE. Here, first it will check within it's namespace for the course value.In the instance namespace(DICTIONARY) it's empty. So, after that it will check with its CLASS for the course value and then print the course value.
o.course = "c++" #WE CAN ADD CHANGE THE COURSE VALUE USING THIS COMMAND. BUT, IT WILL CHANGE THE COURSE VALUE ONLY IN THE INSTANCE 'o' DICTIONARY ALONE. IT CAN'T THE CHANGE THE COURSE VALUE OF THE CLASS ATTRIBUTE.
print(o.__dict__)

```

```
{'__module__': '__main__', 'course': 'java', '__dict__':
<attribute '__dict__' of 'user' objects>, '__weakref__':
<attribute '__weakref__' of 'user' objects>, '__doc__': N
one}
java
{}
java
{'course': 'c++'}
java
c++
>
```

```python
#OBJECT 2
o2 = user()
print(o2.course) #HERE AS WE SAW ALREADY, THIS NEW OBJECT'S NAMESPACE IS EMPTY. SO, IT WILL CJHECK WITH THE CLASS AND THEN PRINT THE COURSE VALUE OF THE CLASS ATTRIBUTE.

#WE CAN CHANGE THE CLASS ATTRIBUTE ALSO
user.course = "c++"
print(user.course)
```

```
{'__module__': '__main__', 'course': 'java', '__dict__':
<attribute '__dict__' of 'user' objects>, '__weakref__':
<attribute '__weakref__' of 'user' objects>, '__doc__': N
one}
java
{}
java
{'course': 'c++'}
java
c++
>
```

3] CLASS METHODS IN PYTHON:

```python
#CLASS METHOD

class Student:
    name = "Mani"
    age = 23

    def printall(): #Function for this class
        print("Name :",Student.name)
        print("Age :",Student.age)

Student.printall()#here, we call the class attributes using function name using DOT NOTATION.
print(Student.__dict__)# we can check the dictionary of the student class.

print(getattr(Student,'printall'))
getattr(Student,'printall')() #here also we can call the class attributes using GETATTRIBUTE METHOD. For that we have to additionally give '()' after printall.
```

```
Name : Mani
Age : 23
{'__module__': '__main__', 'name': 'Mani', 'age': 23, 'pr
intall': <function Student.printall at 0x7fd561247a60>, '
__dict__': <attribute '__dict__' of 'Student' objects>, '
__weakref__': <attribute '__weakref__' of 'Student' objec
ts>, '__doc__': None}
<function Student.printall at 0x7fd561247a60>
Name : Mani
Age : 23
>
```

4] INSTANCE METHOD:

```
#INSTANCE METHOD
class Student:
  name = "Mani"
  age = 23

  def printall(self): #HERE WE HAVE TO USE "SELF" KEY WORD FOR INSTANCE METHOD. BY THIS SELF KEYWORD WE CAN DIRECTLY CALL TE CLASS ATTRIBUTES USING THE "OBJECT".
    print("Name :",Student.name)
    print("Age :",Student.age)

o = Student()
o.printall()
```

```
Name : Mani
Age : 23
```

```
#INSTANCE METHOD
class Student:
  name = "Mani"
  age = 23

  def printall(self,gender): #HERE WE HAVE TO USE "SELF" KEY WORD FOR INSTANCE METHOD. BY THIS SELF KEYWORD WE CAN DIRECTLY CALL TE CLASS ATTRIBUTES USING THE "OBJECT".

#SELF - IT'S A DEFAULT ARGUMENT (I mean first parameter). So, it will automatically take thefirst aruguments through the object. In case if we need to give any nw attributes, then we have to give it additionally near to the self keyword. So, it will take the first attribute value by default and then for the new attribute we have to give the value in the object.
    print("Name :",Student.name)
    print("Age :",Student.age)
    print("Gender :",gender)

o = Student()
o.printall("Male")#here, we have to give the value for the newly added attribute. here, name and age age self attribute(first attribute). It aill automatically take the values fro the class Student.
```

```
Name : Mani
Age : 23
Gender : Male
```

5] CONSTRUCTOR: (__init__ Method):

```python
#__init__ method

class user:
  def __init__(self):
    print("call when new instance created")

o = user() #In constructor if we created the object it will automatically call the funtion program.
#we cann call it multiple times by creating secral objects.
o1 = user()
```

```
call when new instance created
call when new instance created
>
```

```python
#__init__ method

class user:
  def __init__(self,name): # if we need to add a new attribute 'name', then we have to give it here.
    print("call when new instance created")
    self.name = name #it will add the name we give newly in 'self.name'.

  def printall(self): #in previous function we add the new attribute "name". So, in this function "name" is assigned as the self(first parameter) like we did in previous concept "INSTANCE METHOD"
    print("Name :",self.name)

o = user("Mani") #So, while creating the constructor we have to pass the value for the name as the 'parameter'
o.printall()
print(o.__dict__)#here we check the dictionary of the object "o"
o1 = user("Maddy")
o1.printall()
print(o1.__dict__)#here we check the dictionary of the object "o1"
```

```
call when new instance created
Name : Mani
{'name': 'Mani'}
call when new instance created
Name : Maddy
{'name': 'Maddy'}
{'__module__': '__main__', '__init__': <function user.__init__ at 0x7f8c045c6a60>, 'printall': <function user.printall at 0x7f8c045c6af0>, '__dict__': <attribute '__dict__' of 'user' objects>, '__weakref__': <attribute '__weakref__' of 'user' objects>, '__doc__': None}
```



```python
print(user.__dict__) #if we check the dictionary of the CLASS "user", it will not contain the "name" value. Because, in this program we give the name using self keyword as a "INSTANCE ATTRIBUTE". So, the name value will be only present in their respective objects, not present in class.
```

```
call when new instance created
Name : Mani
{'name': 'Mani'}
call when new instance created
Name : Maddy
{'name': 'Maddy'}
{'__module__': '__main__', '__init__': <function user.__init__ at 0x7f8c045c6a60>, 'printall': <function user.printall at 0x7f8c045c6af0>, '__dict__': <attribute '__dict__' of 'user' objects>, '__weakref__': <attribute '__weakref__' of 'user' objects>, '__doc__': None}
```

6| PROPERTY DECORATOR:

```
1 class user:
2   def __init__(self,name,age):
3     self.name = name
4     self.age = age
5     self.msg = self.name + "is"  +  str(self.age) + "years old"#here we use"str
6  near age, because, while concadination we can't concadinate string and variable. So, we convert the integer age to String and then concadinate them.
7 o = user("Mani",23)
8 print(o.name)
9 print(o.age)
10 print(o.msg)
```

```
Mani
23
Maniis23years old
```

LINE 6: Here, for using property decorator, we have to command the self.msg in the previous program and then change it into the function and then we have to give that msg command in "return".

```
1 class user:
2   def __init__(self,name,age):
3     self.name = name
4     self.age = age
5     #self.msg = self.name + "is"  +  str(self.age) + "years old"      #here we use"str"near age, because, while concadination we can't concadinate string and variable. So, we convert the integer age to String and then concadinate them.
6   def msg(self):
7     return self.name + "is"  +  str(self.age) + "years old"
8
9 o = user("Mani",23)
10 print(o.name)
11 print(o.age)
12 print(o.msg)
13 o.age = 45 #here we give the new value to the age, but it can't change. Because, here in constructor the instance values are assinged when we called the constructor using object instance. So, while we give new values in runtime we can't call the constructor. Because, the instance can call the constructor only once.
14 print(o.msg)
15 #So if we want to the change the value instance and update the value without any error we use "PROPERTY DECORATOR".
```

```
Mani
23
<bound method user.msg of <__main__.user object at 0x7f31
2df2ffa0>>
<bound method user.msg of <__main__.user object at 0x7f31
2df2ffa0>>
```

LINE 12 and 14: After that while running we get the output like this in the previous one.(eg. <bound method user.msg…..) like that. It shows like this because we change the msg from attribute to function. So, we have to declare the msg as function in the object instance also (eg. msg()) in every place where the msg word is available.

```
1 class user:
2   def __init__(self,name,age):
3     self.name = name
4     self.age = age
5     #self.msg = self.name + "is"  +  str(self.age) + "years
  old"
6   def msg(self):
7     return self.name + "is"  +  str(self.age) + "years old"
8
9 o = user("Mani",23)
10 print(o.name)
11 print(o.age)
12 print(o.msg())
13 o.age = 45
14 print(o.msg())
```

```
Mani
23
Maniis23years old
Maniis45years old
```

LINE 6: If we work in a big project several team members are work with the same class (eg class user:). In that situation we can't change the "msg()" in every place of the object. So, in order to rectify that we use PROPERTY DECORATOR. Here, we have to give the property decorator before the "msg" function we created, after that no need to change the "msg" as "msg()" in object instance.

```
1 class user:
2   def __init__(self,name,age):
3     self.name = name
4     self.age = age
5     #self.msg = self.name + "is"  +  str(self.age) + "years
  old"
6   @property
7   def msg(self):
8     return self.name + "is"  +  str(self.age) + "years old"
9
10 o = user("Mani",23)
11 print(o.name)
12 print(o.age)
13 print(o.msg)
14 o.age = 45
15 print(o.msg)
```

```
Mani
23
Maniis23years old
Maniis45years old
```

7] PROPERTY DECORATOR GETTER SETTER IN PYTHON:

8] PROPERTY METHOD IN PYTHON:

9] CLASS METHOD DECORATOR:

```
main.py
class Student:    #HERE IN THIS PROGRAM IT WILL PRINT THE STUDENT'S NAME AND AGE WHILE WE CREATING SEVERAL OBJECTS,
#PRINT THE TOTAL NUMBER OF ADMISSIONS BASED ON THE NUMBER OF STUDENT DETAILS WE GET, FOR THAT WE HAVE TO USE CLASS
#METHOD.
    count = 0 #HERE WE DECLARE THE COUNT AS CLASS VARIABLE TO COUNT THE TOTAL NO. OF ADMISSION

    def __init__(self,name,age):
        self.name = name
        self.age = age
        Student.count += 1 #HERE, WHILE WE GIVE VALUE FOR THE STUDENT NAME AND AGE, THEN WHILE CALLING THE
#CONSTRUCTOR USING EACH OBJECT, IT WILL INCREMENT THE COUNT VALUE AND THEN GIVE THE TOTAL NO. OF ADMISSION VALUE.

    def printdetail(self):
        print("Name :",self.name, "Age :",self.age)

    @classmethod #HERE WE USE CLASS METHOD
    def total(cls): #HERE, WE CREATE THE FUNCTION USING CLASS VARIABLE AND THEN RETURN THE COUNT VALUE IN THE
#CLASS STUDENT.
        return cls.count

o = Student("Mani",23)
o.printdetail()
```

```
o = Student("Mani",23)
o.printdetail()

print("Total Admission :",Student.total()) #IT WILL PRINT THE TOTAL NO. OF ADMISSION BY CALLING THE TOTAL
#FUNCTION IN THE CLASS STUDENT.
```

```
input
Name : Mani Age : 23
Name : Mani Age : 23
Total Admission : 2
```


10] STATIC MEYHOD IN PYTHON:

```python
class Student():
  def __init__(self,name,age):
    self.name = name
    self.age = age
  def printdetails(self):
    print("Name :",self.name,"Age :",self.age)
  @staticmethod
  def welcome(): #In python, within the class we have to give self instance variable for all the methods we declared inside the class. Otherwise it will give an error.
#here, in this program we give student details and print the details and then we have one more function, that is "WELCOME" function, we have to display the Welcome message to all the students. So, it's a COMMON FUNCTION. For that we didn't give instance variable(self). In orader to rectify that error we have to give"@staticmethod" above the WELCOME FUNCTION.
    print("Welcome To Our College")
s = Student("Mani", 23)
s.printdetails()
s.welcome()
s = Student("MADDY", 25)
s.printdetails()
s.welcome()
```

```
Name : Mani Age : 23
Welcome To Our College
Name : MADDY Age : 25
Welcome To Our College
```

11] ABSTRACTION AND ENCAPSULATION IN PYTHON:

12] DATA ABSTRACTION - ***Refers to providing only essential information to the outside world and hiding their background details.***

13] ENCAPSULATIION - ***Wrapping code and data together into a single unit.***

14] SINGLE INHERITANCE:

- ***INHERITANCE - Is a process in which one object acquires all the properties and behaviors of its parent object automatically.***
- ***SINGLE INHERITANCE - the inheritance in which a derived class is inherited from only one base class.***

```python
class Nokia:
    company = "Nokia"
    website = "www.nokia.com"

    def contact_details(self):
        print("Address : Cherry road, Chennai.")

class Nokia1001(Nokia):#here we inherit the Nokia class (Single inheritance)
    def __init__(self):
        self.name = "Nokia 1001"
        self.year = 1998

    def product_details(self):
        print("Name : ",self.name)
        print("Year : ",self.year)
        print("Company : ",self.company)
        print("Website : ",self.website)

mobile = Nokia1001()
mobile.product_details()
mobile.contact_details()
```

Output:

```
Name :  Nokia 1001
Year :  1998
Company :  Nokia
Website :  www.nokia.com
Address : Cherry road, Chennai.
```

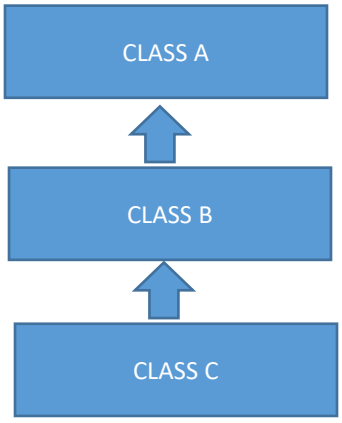
15| MULTIPLE INHERITANCE:

- Class inherits properties of more than one parent class.

```python
class Mother:
  def fishing(self):
    print("Fishing in rivers")
  def Cooking(self):
    print("Cooking Food")

class Father:
  def fishing(self):
    print("Fishing in lake")
  def Chess(self):
    print("Playing chess")

class Son(Mother,Father):#here we inherit the classes.
  def ride(self):
    print("Riding bike")

o = Son()
o.ride()
o.Cooking()
o.fishing()#here the fishing function is inherited from Mother , because the function is inherited based on the priority of the class we inherited. He we inherited Mother as the first class.
o.Chess()
```

```
Riding bike
Cooking Food
Fishing in rivers
Playing chess
```

16] MULTILEVEL INHERITANCE:



```python
class Grandfather:
  def own_house(self):
    print("grandpa having house")

class Father(Grandfather):#here Father inherit Grandfather class
  def own_bike(self):
    print("daddy having bike")

class Son(Father):#here Son inherit Father class
  def own_car(self):
    print("son having car")

o = Son()
o.own_car()
o.own_bike()
o.own_house()

o1 = Father()
o1.own_bike()
o1.own_house()

o2 = Grandfather()
o2.own_house()
```

```
son having car
daddy having bike
grandpa having house
daddy having bike
grandpa having house
grandpa having house
>
```

17] FUNCTION OVERRIDING:

```python
class Employee:
  def working_hrs(self):
    self.hrs = 50
  def print_hrs(self):
    print("Total Working Time :",self.hrs)

class Trainee(Employee):
  def working_hrs(self):
    self.hrs = 60

employee = Employee()
employee.working_hrs()
employee.print_hrs()

trainee = Trainee()
trainee.working_hrs()
trainee.print_hrs()#in trainee class we didn't have print_hrs function, But, it will check in it's inherited class(Employee class) for print_hrs function and then execute.
```

```
Total Working Time : 50
Total Working Time : 60
>
```

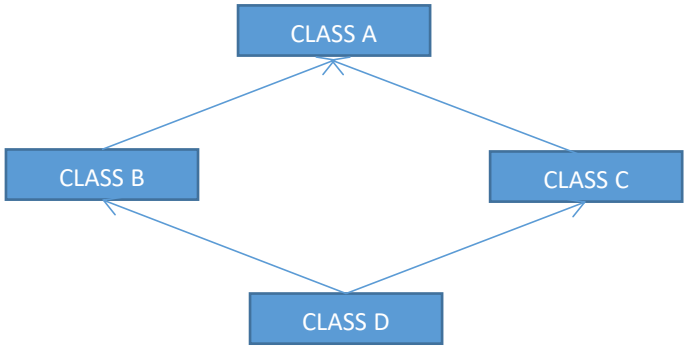
```python
class Employee:
  def working_hrs(self):
    self.hrs = 50
  def print_hrs(self):
    print("Total Working Time :",self.hrs)

class Trainee(Employee):
  def working_hrs(self):
    self.hrs = 60
  def reset_hrs(self):#once the trainee completed the training, then he will be promoted to employee. In that case, we have to change the working hrs for that trainee. So we use rest_hrs function. Here, we use "Super keyword" - it will access the super class working_hrs function and then rest the working_hrs for trainee.
    super().working_hrs()

employee = Employee()
employee.working_hrs()
employee.print_hrs()

trainee = Trainee()
trainee.working_hrs()
trainee.print_hrs()
```

```
Total Working Time : 50
Total Working Time : 60
Total Working Time : 50
```

```python
trainee = Trainee()
trainee.reset_hrs()
trainee.print_hrs()
#in trainee class we didn't have print_hrs function, But, it will check in it's inherited class(Employee class) for print_hrs function and then execute.
```

```
Total Working Time : 50
Total Working Time : 60
Total Working Time : 50
```

18| DIAMOND PROBLEM:



```python
class A:
  def display(self):
    print("I am in display of class A")
class B(A):
  def display(self):
    print("I am in display of class B")
class C(A):
  def display(self):
    print("I am in display of class C")
class D(B, C):
  def display(self):
    print("I am in display of class D")

o = D()
o.display()
#Here the concept is, B and C inherits class A, and class D inherits both classes B and C. Working procedure - once we created the object for the class and call the display function it will check with the particular class for that we created the object. If no display method is available in CLASS D means, then it will check eith its INHERITED CLASS(B,C - THIS ALSO BASED ON THE PRIORITY WE GIVE, IF WE GIVE B FIRST THEN IT WILL CHECK B FIRST). Like that it will work.
```

```
I am in display of class D
>
```

```python
class A:
  def display(self):
    print("I am in display of class A")
class B(A):
  def display(self):
    print("I am in display of class B")
class C(A):
  def display(self):
    print("I am in display of class C")
class D(C, B):
  pass
o = D()
o.display()
#Here the concept is, B and C inherits class A, and class D inherits both classes B and C. Working procedure - once we created the object for the class and call the display function it will check with the particular class for that we created the object. If no display method is available in CLASS D means, then it will check eith its INHERITED CLASS(B,C - THIS ALSO BASED ON THE PRIORITY WE GIVE, IF WE GIVE B FIRST THEN IT WILL CHECK B FIRST). Like that it will work.
```

```
I am in display of class C
>
```

19] OPERATOR OVERLOADING:

- IT COMES UNDER POLYMORPHISM.

20] ABSTRACT METHOD, ABSTRACT BASE CLASS IN PYTHON:

ABSTRACT CLASS:

- The class which is inherited from the BASE CLASS. All the functions inside the abstract class doesn't contain any definition, it only contains the function name.
- If we inherit the ABSTRACT CLASS with ANOTHER CLASS, then we have to define all the abstract methods that are present inside the abstract class.
- In the derived class we can give additional methods also, but importantly we have to define all the methods within the abstract class.

```python
from abc import ABC, abstractmethod #here we inherited the vlass ABC form the base class "abc".

class bank(ABC):#HERE bank is inherited from the class ABC. Here bank is the abstract class
  @abstractmethod#in abstract class we have to use this abstractmethod notation, because we didn't define any functions within the abstract class.
  def loan(self):
    pass
  @abstractmethod
  def credit(self):
    pass
  @abstractmethod
  def debit(self):
    pass

class HDFC(bank):
  def loan(self):
    print("HDFC will provide loan")
  def credit(self):
    print("HDFC will give 10% credit")
```

```
HDFC will provide loan
HDFC will give 10% credit
HDFC will provide debit
HDFC will provide card
>
```

```python
  def debit(self):
    print("HDFC will provide debit")
  def credit_card(self):#this one method is additionally we give other than the methods in the abstract class.
    print("HDFC will provide card")
o = HDFC()
o.loan()
o.credit()
o.debit()
o.credit_card()
```

```
HDFC will provide loan
HDFC will give 10% credit
HDFC will provide debit
HDFC will provide card
>
```

21] HOW TO OPEN AND READ A FILE IN PYTHON:

```python
try:
  file = open("C:\\Users\\smc\\Desktop\\PYTHON\\a.txt","r") #HERE "r" refers to read. If we want to write then we have to give "w".
  print(file.read())
except FileNotFoundError:
  print("Error : File not found")
else:
  file.close()

#In this program, if the file is present in that particular location, then it will print the content of that file. Otherwise, it will print as "FILE NOT FOUND".
```

```
Error : File not found
>
```

22] READLINE AND READLINES IN PYTHON:

In previous concept, we use ".read()" method. Here we use ".readline()",".readlines()".

- ***readline() - it will print line by line the content of the file.***
- ***readline(2) - here it will print the first two letters of the line in that file.it will vary based on the number we give.***
- ***readlines() - it will print all the lines in the LIST format.***

23] LOOP LINE BY LINE IN PYTHON FILE CONCEPT:

24] WRITE OR OVER WRITE TO AN EXISTING FILE IN PYTHON:

Here we use "w" to write the new content to the existing file. It will print the new line we give.

25] APPEND MODE FILE IN PYTHON:

In APPEND mode we use "a" to write the new content. In append if we give any new sentence, then it will add to the next line and then add the new content line by line.

26] DELETE A FILE IN PYTHON:

- ***To delete a file first we have to import the OS, to get the current system permission.***

- ***We have to check with exist method to know whether the file is present in the path. If the file is present, then we have to use "remove" function to remove the file.***
- ***If the file is not present, then the else part will execute as "File not Found".***
- ***If we want to delete the FOLDER, then we have to use ".rmdir("folder name")"***

27] SQLite Browser: