In [ ]:

```
'''

Class definition

Object definition

Reference definition


1 class multiple objects
1 object one or more multiple references

Syntax - class ClassName
                #code inside the class

'''
```

In [ ]:

```
'''

Class vs Object vs Instance

with 9 comments

In OO Programming, we often hear of terms like “Class”, “Object” and “Instance”;
but what actually is a Class / Object / Instance?

In short,

An object is a software bundle of related state and behavior.

A class is a blueprint or prototype from which objects are created.

An instance is a single and unique unit of a class.

Example,
we have a blueprint (class) represents student (object) with fields like name, a
ge, course (class member).
And we have 2 students here, Foo and Bob. So, Foo and Bob is 2 different instanc
es of the class
(Student class) that represent object (Student people).

Let me go into details…

Object
Real world objects shares 2 main characteristics, state and behavior.
Human have state (name, age) and behavior (running, sleeping).
Car have state (current speed, current gear) and behavior (applying brake, chang
ing gear).
Software objects are conceptually similar to real-world objects: they too consis
t of state and related behavior. An object stores its state in fields and expose
s its behavior through methods.

Class
Class is a “template” / “blueprint” that is used to create objects.
Basically, a class will consists of field, static field, method, static method a
nd constructor.
Field is used to hold the state of the class (eg: name of Student object).
Method is used to represent the behavior of the class (eg: how a Student object
 going to stand-up).
Constructor is used to create a new Instance of the Class.

Instance
An instance is a unique copy of a Class that representing an Object.
When a new instance of a class is created, the JVM will allocate a room of memor
y for that class instance.


'''
```

In [ ]:

```
# https://www.youtube.com/watch?v=hLC6GL_YBGM

# https://www.w3schools.com/python/python_classes.asp

# https://www.programiz.com/article/python-self-why

# To understand the meaning of classes we have to understand the __init__() func
tion, which is a constructor

# All classes have a function called __init__(), which is always executed when t
he class is being initiated.

# Use the __init__() function to assign values to object properties,
# or other operations that are necessary to do when the object is being created:

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

# Objects can also contain methods. Methods in objects are functions that belong
to the object.

# Note: The self parameter is a reference to the current instance of the class,
# and is used to access variables that belong to the class.

# It does not have to be named self , you can call it whatever you like,
# but it has to be the first parameter of any function in the class.
```

In [3]:

```
'''Please dont edit this example, As this example explains the self parameter cl
early'''

class PersonDetails:
    def __init__(self, ssid, name):    # here the self parameter is srini
        self.ssid1=ssid                # here the ssid1 is the instance variable
        self.name1=name                # here the name1 is the instance variable

    def tellDetails(self, vasu1):       # here the self parameter is vasu
        print("vasu1",vasu1)            # here the method parameter is vasu1 is
 a pass by parameter
        print("my ssid is:",self.ssid1) # here the ssid1 is the instance variabl
e
        print("my name is:",self.name1) # here the name1 is the instance variabl
e

p1=PersonDetails(12345,"Raj")           # p1 is the object of the class PersonDe
tails here

p2=PersonDetails(45678,"Ram")           # p2 is the object of the class PersonDe
tails here

print(p1.ssid1)                         # printing the variable ssid of the obje
ct p1
print(p1.name1)                         # printing the variable name of the obje
ct p1

print("------------")

print(p2.ssid1)                         # printing the variable ssid of the obje
ct p2
print(p2.name1)                         # printing the variable name of the obje
ct p2

print("--------------")

p1.tellDetails(10)                      # same as PersonDetails.tellDetails(p1,1
0)
p2.tellDetails(10)                      # same as PersonDetails.tellDetails(p2,1
0)

print("--------------")

PersonDetails.tellDetails(p1,10)        # same as p1.tellDetails(10)
PersonDetails.tellDetails(p2,10)        # same as p2.tellDetails(10)

print("--------------")


'''

p1.tellDetails(10) is same as PersonDetails.tellDetails(p1,10)

As per the above statement the self parameter is always required to be first par
amenter in the methods
of the class as internally the stmt " p1.tellDetails(10)" will be always convert
ed as
"PersonDetails.tellDetails(p1,10)" so to match the internal conversion we need t
o insert the self parameter
```

```
only through which we can identify the object reference p1.

So the syntax becomes as,

<Class Name>.<Method Name>(<Object name>, <argument for the method>)

which is same as,

<Object name>.<Method Name>(<arguement for the method>)

so hence the method definition should also be maintained as,

<Method Name>(<self parameter>, <arguement for the method>)

'''
```

```
12345
Raj
------------
45678
Ram
--------------
vasu1 10
my ssid is: 12345
my name is: Raj
vasu1 10
my ssid is: 45678
my name is: Ram
--------------
vasu1 10
my ssid is: 12345
my name is: Raj
vasu1 10
my ssid is: 45678
my name is: Ram
--------------
```

Out[3]:

```
'\n\np1.tellDetails(10) is same as PersonDetails.tellDetails(p1,10)
\n\nAs per the above statement the self parameter is always required
to be first paramenter in the methods \nof the class as internally t
he stmt " p1.tellDetails(10)" will be always converted as \n"PersonD
etails.tellDetails(p1,10)" so to match the internal conversion we ne
ed to insert the self parameter\nonly through which we can identify
the object reference p1.\n\nSo the syntax becomes as,\n\n<Class Name
>.<Method Name>(<Object name>, <argument for the method>)\n\nwhich i
s same as,\n\n<Object name>.<Method Name>(<arguement for the method
>)\n\nso hence the method definition should also be maintained as,\n
\n<Method Name>(<self parameter name>, <arguement for the method>)\n
\n'
```

In [3]:

```
# Example 1 - A Class with the __init__ constructor and a method, no class varia
ble

class PersonDetails:
    '''this class is for learning purpose and gives the details about the compan
y information'''

    def __init__(self, ssid, name):   # here the self parameter is self
        self.ssid=ssid                # here the ssid is the instance variable
        self.name=name                # here the name is the instance variable

    def tellDetails(self):            # here the self parameter is self
        print("my ssid is:",self.ssid)
        print("my name is:",self.name)

p1=PersonDetails(12345,"Raj")  #p1 is the object of the class PersonDetails here
p2=PersonDetails(45678,"Ram")  #p2 is the object of the class PersonDetails here

print(p1.ssid)  #printing the variable ssid of the object p1
print(p1.name)  #printing the variable name of the object p1

print("------------")

print(p2.ssid)  #printing the variable ssid of the object p2
print(p2.name)  #printing the variable name of the object p2
```

```
ssid: 12345
name: Raj
ssid: 45678
name: Ram
12345
Raj
------------
45678
Ram
```

In [85]:

```
# Example 2 - A Class without the __init__ constructor, but with a method and tw
o class variables

class PersonDetails:

    ssid=12345   # field 1, variable declared as member of class, this is a clas
s variable,
                 # can be used in all methods of the class
    name="Raj"   # field 2, variable declared as member of class, this is a clas
s variable,
                 # can be used in all methods of the class

    def tellDetails(self2):
        print("my ssid is:",self2.ssid)
        print("my name is:",self2.name)

p1=PersonDetails()  #p1 is the object of the class PersonDetails here

print(p1.ssid)  #printing the variable ssid of the object p1
print(p1.name)  #printing the variable name of the object p1

print("------------")

p1.tellDetails() #same as PersonDetails.tellDetails(p1)

print("------------")

PersonDetails.tellDetails(p1) #same as p1.tellDetails()
```

```
12345
Raj
------------
my ssid is: 12345
my name is: Raj
------------
my ssid is: 12345
my name is: Raj
```

In [110]:

```
# Example 3 - A Class without the __init__ constructor, but with a method and va
riable within it
# and no class variables

# Here val1 is a local variable inside the method tellDetails() which is defined
as a pass by parameter method

class PersonDetails:
    '''this class is for learning purpose and gives the details about the compan
y information'''

    def tellDetails(self,val1):            # here the self parameter is self
        print("my val1 is:",val1)

p1=PersonDetails()  #p1 is the object of the class PersonDetails here
p2=PersonDetails()  #p2 is the object of the class PersonDetails here

p1.tellDetails(10)

print("------------")

p2.tellDetails(20)
```

```
my val1 is: 10
------------
my val1 is: 20
```

In [112]:

```
# Example 4 - A Class without the __init__ constructor, but with a method within
it and no class variables

# Here val1 is a local variable inside the method tellDetails()

class PersonDetails:
    '''this class is for learning purpose and gives the details about the compan
y information'''

    def tellDetails(self):            # here the self parameter is self
        val1=10
        print("my val1 is:",val1)

p1=PersonDetails()  #p1 is the object of the class PersonDetails here
p2=PersonDetails()  #p2 is the object of the class PersonDetails here

p1.tellDetails()

print("------------")

p2.tellDetails()
```

```
my val1 is: 10
------------
my val1 is: 10
```

In [4]:

```
print(id(p1))
print(id(p2))
```

```
140193763641160
140193763642224
```

In [5]:

```
print(type(p1))
print(type(p2))
```

```
<class '__main__.PersonDetails'>
<class '__main__.PersonDetails'>
```

In [6]:

```
# Normal Example - where one constructor __init__ and a method tellCompanyInfo i
s made in the
# class CompanyDetails

class CompanyDetails:
    '''this class is for learning purpose and gives the details about the compan
y information'''

    def __init__(self,name, address, no_of_emp, no_of_dept, company_description,
no_of_product):
        self.name=name
        self.address=address
        self.no_of_emp=no_of_emp
        self.no_of_dept=no_of_dept
        self.company_description=company_description
        self.no_of_product=no_of_product

    def tellCompanyInfo(self):
        print("**********************")
        print("company name is",self.name)
        print("company address is",self.address)
        print("company no of employees are",self.no_of_emp)
        print("company no of departments are",self.no_of_dept)
        print("company description is",self.company_description)
        print("company no of products are",self.no_of_product)

c1=CompanyDetails("Infosys","E-City",2000,6,"Infosys Company",10) #c1 is the obj
ect of the class CompanyDetails here
c2=CompanyDetails("CGI","Whitefield",5000,8,"CGI Company",15) #c2 is the object
 of the class CompanyDetails here
c3=CompanyDetails("TCS","MG_Road",10000,10,"TCS Company",20) #c3 is the object o
f the class CompanyDetails here

c1.tellCompanyInfo()
c2.tellCompanyInfo()
c3.tellCompanyInfo()
```

```
**********************
company name is Infosys
company address is E-City
company no of employees are 2000
company no of departments are 6
company description is Infosys Company
company no of products are 10
**********************
company name is CGI
company address is Whitefield
company no of employees are 5000
company no of departments are 8
company description is CGI Company
company no of products are 15
**********************
company name is TCS
company address is MG_Road
company no of employees are 10000
company no of departments are 10
company description is TCS Company
company no of products are 20
```

In [7]:

```
# doc string

CompanyDetails.__doc__
```

Out[7]:

```
'this class is for learning purpose and gives the details about the
company information'
```

In [8]:

```
# Normal Example - where only one constructor __init__ is made in the class Demo

class Demo:
    def __init__(self):
        print("inside of class",id(self)) #both ids refer to same object


d=Demo()  #d is the object of the class Demo here
print("outside of class - d object",id(d)) #both ids refer to same object
print("*****************")
d1=Demo() #d1 is the object of the class Demo here
print("outside of class - d1 object",id(d1)) #both ids refer to same object
```

```
inside of class 140193754771584
outside of class - d object 140193754771584
*****************
inside of class 140193754771640
outside of class - d1 object 140193754771640
```

In [14]:

```
# Normal Example - where one constructor __init__ and a method tellCompanyInfo i
s made in the class
# CompanyDetails, also we have a local variable name inside the method tellCompa
nyInfo

# the first argument to the constructor(__init(self)) self must be there

# python virtual machine is responsible to provide the self for constructor and
 methods in the class

class CompanyDetails:
    '''this class is for learning purpose and gives the details about the compan
y information'''

    def __init__(self,name, address, no_of_emp, no_of_dept, company_description,
no_of_product):
        self.name=name
        self.address=address
        self.no_of_emp=no_of_emp
        self.no_of_dept=no_of_dept
        self.company_description=company_description
        self.no_of_product=no_of_product

    def tellCompanyInfo(self):
        print("*********************")

        '''
        in case if i use local variable and dont use the self variable then it t
akes local variable
        to differ local variable and current object variable we use self

        '''


        name="cisco"
        print("Local company name is:",name)

        print("company name is",self.name)
        print("company address is",self.address)
        print("company no of employees are",self.no_of_emp)
        print("company no of departments are",self.no_of_dept)
        print("company description is",self.company_description)
        print("company no of products are",self.no_of_product)

c1=CompanyDetails("Infosys","E-City",2000,6,"Infosys Company",10) #c1 is the obj
ect of the class CompanyDetails here
c2=CompanyDetails("CGI","Whitefield",5000,8,"CGI Company",15) #c2 is the object
 of the class CompanyDetails here
c3=CompanyDetails("TCS","MG_Road",10000,10,"TCS Company",20) #c3 is the object o
f the class CompanyDetails here

c1.tellCompanyInfo()
c2.tellCompanyInfo()
c3.tellCompanyInfo()

CompanyDetails.__doc__
```

```
**********************
Local company name is: cisco
company name is Infosys
company address is E-City
company no of employees are 2000
company no of departments are 6
company description is Infosys Company
company no of products are 10
**********************
Local company name is: cisco
company name is CGI
company address is Whitefield
company no of employees are 5000
company no of departments are 8
company description is CGI Company
company no of products are 15
**********************
Local company name is: cisco
company name is TCS
company address is MG_Road
company no of employees are 10000
company no of departments are 10
company description is TCS Company
company no of products are 20
```

Out[14]:

```
'this class is for learning purpose and gives the details about the
company information'
```

In [11]:

```
# Normal Example - where one constructor __init__ and a method tellCompanyInfo i
s made in the class
# CompanyDetails, also we have a local variable name inside the method tellCompa
nyInfo

# the self variable can differ by name, we can use any name instead of self

class CompanyDetails:
    '''this class is for learning purpose and gives the details about the compan
y information'''

    def __init__(self1, name, address, no_of_emp, no_of_dept, company_descriptio
n, no_of_product):
        self1.name=name
        self1.address=address
        self1.no_of_emp=no_of_emp
        self1.no_of_dept=no_of_dept
        self1.company_description=company_description
        self1.no_of_product=no_of_product

    def tellCompanyInfo(self2):
        print("***********************")

        '''
        in case if i use local variable and dont use the self variable then it t
akes local variable
        to differ local variable and current object variable we use self

        '''

        name="cisco"
        print("Local company name is:",name)

        print("company name is",self2.name)
        print("company address is",self2.address)
        print("company no of employees are",self2.no_of_emp)
        print("company no of departments are",self2.no_of_dept)
        print("company description is",self2.company_description)
        print("company no of products are",self2.no_of_product)

c1=CompanyDetails("Infosys","E-City",2000,6,"Infosys Company",10) #c1 is the obj
ect of the class CompanyDetails here
c2=CompanyDetails("CGI","Whitefield",5000,8,"CGI Company",15) #c2 is the object
 of the class CompanyDetails here
c3=CompanyDetails("TCS","MG_Road",10000,10,"TCS Company",20) #c3 is the object o
f the class CompanyDetails here

c1.tellCompanyInfo()
c2.tellCompanyInfo()
c3.tellCompanyInfo()

CompanyDetails.__doc__
```

```
**********************
Local company name is: cisco
company name is Infosys
company address is E-City
company no of employees are 2000
company no of departments are 6
company description is Infosys Company
company no of products are 10
**********************
Local company name is: cisco
company name is CGI
company address is Whitefield
company no of employees are 5000
company no of departments are 8
company description is CGI Company
company no of products are 15
**********************
Local company name is: cisco
company name is TCS
company address is MG_Road
company no of employees are 10000
company no of departments are 10
company description is TCS Company
company no of products are 20
```

In [10]:

```
'''
lets talk about __init__ now
Its a constructor and responsible declare and intialize the variable values

in java constructor is className()

ex- class Student -> Student() - constructor
ex- class Company -> Student() - constructor

But in python name always -> __init__()

whenever we create a object, constructor is executed automatically and execute o
nly once per object creation

every constructor must take atleast one argument as self and then any no of argu
ment it can take

if we dont provide constructor the python only provides 1 constructor like its s
pecial

__init(self)

'''
```

Out[10]:

```
'\nlets talk about __init__ now\nIts a constructor and responsible d
eclare and intialize the variable values\n\nin java constructor is c
lassName()\n\nex- class Student -> Student() - constructor\nex- clas
s Company -> Student() - constructor\n\nBut in python name always ->
__init__()\n\nwhenever we create a object, constructor is executed a
utomatically and execute only once per object creation\n\nevery cons
tructor must take atleast one argument as self and then any no of ar
gument it can take\n\nif we dont provide constructor the python only
provides 1 constructor like its special\n\n__init(self)\n\n'
```

In [19]:

```python
# if we dont give atleast 1 argument here??
class Demo:
    def __init__(self):
            print("construtor is getting execute here")

d1=Demo() #d1 is the object of the class Demo here
d2=Demo() #d2 is the object of the class Demo here
```

```
construtor is getting execute here
construtor is getting execute here
```

In [114]:

```
# Normal Example - where one constructor __init__ and a method tellCompanyInfo i
s made in the class
# CompanyDetails, here the tellCompanyInfo method is called multiple times

class CompanyDetails:
    '''this class is for learning purpose and gives the details about the compan
y information'''

    def __init__(self1,name, address, no_of_emp):
        self1.name=name
        self1.address=address
        self1.no_of_emp=no_of_emp

    def tellCompanyInfo(self2):
        print("company name is",self2.name)
        print("company address is",self2.address)
        print("company no of employees are",self2.no_of_emp)

c1=CompanyDetails("Infosys","E-City",2000) #c1 is the object of the class Compan
yDetails here

c1.tellCompanyInfo()
c1.tellCompanyInfo()
```

```
company name is Infosys
company address is E-City
company no of employees are 2000
company name is Infosys
company address is E-City
company no of employees are 2000
```

In [21]:

```
'''

Vaiables inside the class

1. Instance Varibles

2. Static Varibles

3. Local Varibles


Methods inside the class

1. Instance Methods

2. Static Methods

3. Local Methods


(Instance -> Object) variable


(Static -> Class level) variable


(Local -> Method level) variable

'''
```

Out[21]:

```
'\n\nVaiables inside the class\n\n1. Instance Varibles\n\n2. Static
Varibles\n\n3. Local Varibles\n\n\nMethods inside the class\n\n1. In
stance Methods\n\n2. Static Methods\n\n3. Local Methods\n\n\n(Instan
ce -> Object) variable\n\n\n(Static -> Class level) variable\n\n\n(L
ocal -> Method level) variable\n\n\n'
```

In [138]:

```
# Instance varible changes from time to time

# Normal Example - where one constructor __init__ and a method tellCompanyInfo i
s made in the class
# CompanyDetails, here an additional instance variable no_of_offices is added to
the object insatance c1

class CompanyDetails:
    '''this class is for learning purpose and gives the details about the compan
y information'''

    def __init__(self1, name, address, no_of_emp):
        self1.name=name
        self1.address=address
        self1.no_of_emp=no_of_emp

    def tellCompanyInfo(self2):
        print("company name is",self2.name)
        print("company address is",self2.address)
        print("company no of employees are",self2.no_of_emp)

# variable values are changing object by object so its instance variable
# variable which are declared using self function are instance varibles

c1=CompanyDetails("Infosys","E-City",2000) #c1 is the object of the class Compan
yDetails here
c1.tellCompanyInfo()
print("*******************")

c2=CompanyDetails("CGI","Whitefield",5000) #c2 is the object of the class Compan
yDetails here
c2.tellCompanyInfo()
print("*******************")

c3=CompanyDetails("TCS","MG_Road",10000) #c3 is the object of the class CompanyD
etails here
c3.tellCompanyInfo()

#printing instance variable

print("***********")
print("Instance varible are .......",c1.__dict__)  #printing the object instance
s in a dictionary format
print("***********")
print("Instance varible are .......",c2.__dict__)  #printing the object instance
s in a dictionary format
print("***********")
print("Instance varible are .......",c3.__dict__)  #printing the object instance
s in a dictionary format
```

```
company name is Infosys
company address is E-City
company no of employees are 2000
*******************
company name is CGI
company address is Whitefield
company no of employees are 5000
*******************
company name is TCS
company address is MG_Road
company no of employees are 10000
***********
Instance varible are ....... {'name': 'Infosys', 'address': 'E-Cit
y', 'no_of_emp': 2000}
***********
Instance varible are ....... {'name': 'CGI', 'address': 'Whitefiel
d', 'no_of_emp': 5000}
***********
Instance varible are ....... {'name': 'TCS', 'address': 'MG_Road',
'no_of_emp': 10000}
```

In [137]:

```
# Instance varible changes from time to time

# Normal Example - where one constructor __init__ and a method tellCompanyInfo i
s made in the class
# CompanyDetails, here an additional instance variable no_of_offices is added to
the object insatance c1

class CompanyDetails:
    '''this class is for learning purpose and gives the details about the compan
y information'''

    def __init__(self1, name, address, no_of_emp):
        self1.name=name
        self1.address=address
        self1.no_of_emp=no_of_emp

    def tellCompanyInfo(self2):
        print("company name is",self2.name)
        print("company address is",self2.address)
        print("company no of employees are",self2.no_of_emp)

# variable values are changing object by object so its instance variable
# variable which are declared using self function are instance varibles

c1=CompanyDetails("Infosys","E-City",2000) #c1 is the object of the class Compan
yDetails here
c1.tellCompanyInfo()
print("*********************")

c2=CompanyDetails("CGI","Whitefield",5000) #c2 is the object of the class Compan
yDetails here
c2.tellCompanyInfo()
print("*********************")

c3=CompanyDetails("TCS","MG_Road",10000) #c3 is the object of the class CompanyD
etails here
c3.tellCompanyInfo()

#printing instance variable

print("************")
print("Instance varible are .......",c1.__dict__)  #printing the object instance
s in a dictionary format
print("************")
print("Instance varible are .......",c2.__dict__)  #printing the object instance
s in a dictionary format
print("************")
print("Instance varible are .......",c3.__dict__)  #printing the object instance
s in a dictionary format

# from outside of class the with object reference so this variable becomes the i
nstance variable

c1.no_of_offices = 10  #an addtional variable for the same object instance c1

# now no of instance variables are 4(one more variable is added5r)

print("************")
print("Instance varible are .......",c1.__dict__) #printing the object instances
```

```
in a dictionary format
print("***********")
print("Instance varible are .......",c2.__dict__) #printing the object instances
in a dictionary format
print("***********")
print("Instance varible are .......",c3.__dict__) #printing the object instances
in a dictionary format
```

```
company name is Infosys
company address is E-City
company no of employees are 2000
*******************
company name is CGI
company address is Whitefield
company no of employees are 5000
*******************
company name is TCS
company address is MG_Road
company no of employees are 10000
***********
Instance varible are ....... {'name': 'Infosys', 'address': 'E-Cit
y', 'no_of_emp': 2000}
***********
Instance varible are ....... {'name': 'CGI', 'address': 'Whitefiel
d', 'no_of_emp': 5000}
***********
Instance varible are ....... {'name': 'TCS', 'address': 'MG_Road',
'no_of_emp': 10000}
***********
Instance varible are ....... {'name': 'Infosys', 'address': 'E-Cit
y', 'no_of_emp': 2000, 'no_of_offices': 10}
***********
Instance varible are ....... {'name': 'CGI', 'address': 'Whitefiel
d', 'no_of_emp': 5000}
***********
Instance varible are ....... {'name': 'TCS', 'address': 'MG_Road',
'no_of_emp': 10000}
```

In [136]:

```
# Class variable can be used by all objects

# Normal Example - where one constructor __init__ and a method tellCompanyInfo i
s made in the class
# CompanyDetails, here an additional instance variable no_of_offices is added to
the object insatance c1

class CompanyDetails:
    '''this class is for learning purpose and gives the details about the compan
y information'''

    no_of_products=100  #static variable, can be used in all methods of the clas
s, this is a class variable

    def __init__(self1,name, address, no_of_emp):
        self1.name=name
        self1.address=address
        self1.no_of_emp=no_of_emp

    def tellCompanyInfo(self2):
        print("company name is",self2.name)
        print("company address is",self2.address)
        print("company no of employees are",self2.no_of_emp)
        print("company product is",self2.no_of_products)

# variable values are changing object by object so its instance variable,

# variable which are declared using self function are instance varibles

c1=CompanyDetails("Infosys","E-City",2000)
c1.tellCompanyInfo()

print("*********************")

c2=CompanyDetails("CGI","Whitefield",5000)
c2.tellCompanyInfo()

print("*********************")

c3=CompanyDetails("TCS","MG_Road",10000)
c3.tellCompanyInfo()

#printing instance variable

print("***********")
print("Instance varible are .......",c1.__dict__) #printing the object instances
in a dictionary format
print("***********")
print("Instance varible are .......",c2.__dict__) #printing the object instances
in a dictionary format
print("***********")
print("Instance varible are .......",c3.__dict__) #printing the object instances
in a dictionary format
```

```
company name is Infosys
company address is E-City
company no of employees are 2000
company product is 100
*******************
company name is CGI
company address is Whitefield
company no of employees are 5000
company product is 100
*******************
company name is TCS
company address is MG_Road
company no of employees are 10000
company product is 100
***********
Instance varible are ....... {'name': 'Infosys', 'address': 'E-Cit
y', 'no_of_emp': 2000}
***********
Instance varible are ....... {'name': 'CGI', 'address': 'Whitefiel
d', 'no_of_emp': 5000}
***********
Instance varible are ....... {'name': 'TCS', 'address': 'MG_Road',
'no_of_emp': 10000}
```

In [135]:

```
# Local variable can ne used only within the function

class CompanyDetails:
    '''this class is for learning purpose and gives the details about the compan
y information'''

    def __init__(self1,name, address, no_of_emp):
        self1.name=name
        self1.address=address
        self1.no_of_emp=no_of_emp

    def tellCompanyInfo(self2):
        no_of_products=10       # local variable in the method tellCompanyInfo c
an be used only within this method
        print("company product is",no_of_products)
        print("company name is",self2.name)
        print("company address is",self2.address)
        print("company no of employees are",self2.no_of_emp)

# variable values are changing object by object so its instance variable,
# variable which are declared using self function are instance varibles

c1=CompanyDetails("Infosys","E-City",2000)
c1.tellCompanyInfo()
print("*******************")

c2=CompanyDetails("CGI","Whitefield",5000)
c2.tellCompanyInfo()
print("*******************")

c3=CompanyDetails("TCS","MG_Road",10000)
c3.tellCompanyInfo()

#printing instance variable

print("***********")
print("Instance variable are .......",c1.__dict__) #printing the object instance
s in a dictionary format
print("***********")
print("Instance variable are .......",c2.__dict__) #printing the object instance
s in a dictionary format
print("***********")
print("Instance variable are .......",c3.__dict__) #printing the object instance
s in a dictionary format
```

```
company product is 10
company name is Infosys
company address is E-City
company no of employees are 2000
*******************
company product is 10
company name is CGI
company address is Whitefield
company no of employees are 5000
*******************
company product is 10
company name is TCS
company address is MG_Road
company no of employees are 10000
***********
Instance variable are ....... {'name': 'Infosys', 'address': 'E-Cit
y', 'no_of_emp': 2000}
***********
Instance variable are ....... {'name': 'CGI', 'address': 'Whitefiel
d', 'no_of_emp': 5000}
***********
Instance variable are ....... {'name': 'TCS', 'address': 'MG_Road',
'no_of_emp': 10000}
```

In [9]:

```python
# Static variable can be used within the class function

class CompanyDetails:

    industryType="IT-Industry"  #static variable

    def __init__(self1, name, address, no_of_emp):

        self1.name=name
        self1.address=address
        self1.no_of_emp=no_of_emp

    def tellCompanyInfo(self2):

        print("company name is",self2.name)
        print("company address is",self2.address)
        print("company no of employees are",self2.no_of_emp)

    #@classmethod   # this is a decorator *******************
    def getIndustryType(cls1):
        print("Type of industry is:",cls1.industryType)

        # this method is using class level variableso its class level method
        # cls is alternate to self (in instance we use self and in class method)
        # self and cls are not predefined we can use any name

    def findSum(self, no1, no2):  #local method
        print("sum is:",(no1+no2))

def findAverage(no1, no2):       #local method
    print("Average is:",(no1+no2)/2)

c1=CompanyDetails("Infosys","E-City",2000)

c1.tellCompanyInfo()

CompanyDetails.getIndustryType(c1)

c1.getIndustryType()

print("*********************")

c1.findSum(10,10)    # method1 to call findSum

CompanyDetails.findSum(c1, 10, 10) # method2 to call findSum

findAverage(10, 10)
```

```
company name is Infosys
company address is E-City
company no of employees are 2000
Type of industry is: IT-Industry
Type of industry is: IT-Industry
*********************
sum is: 20
sum is: 20
Average is: 10.0
```

In [158]:

```python
# Local variable can ne used only within the local function

class CompanyDetails:

    def __init__(self1,name, address, no_of_emp):

        self1.name=name
        self1.address=address
        self1.no_of_emp=no_of_emp

    def tellCompanyInfo(self2):

        print("company name is",self2.name)
        print("company address is",self2.address)
        print("company no of employees are",self2.no_of_emp)

    @staticmethod   # this is a decorator *******************
    def findSum(no1, no2):  #local method - no self or cls keyword is required
        print("sum is:",(no1+no2))

def findAverage(no1, no2):       #local method
    print("Average is:",(no1+no2)/2)

c1=CompanyDetails("Infosys","E-City",2000)

c2=CompanyDetails("Infosys","E-City",2000)

c1.tellCompanyInfo()

print("*********************")

c2.tellCompanyInfo()

print("*********************")

c1.findSum(10,10)    # method1 to call findSum
CompanyDetails.findSum(13, 10) # method2 to call findSum

print("*********************")

c2.findSum(20,10)    # method1 to call findSum
CompanyDetails.findSum(14, 10) # method2 to call findSum

findAverage(10, 10)
```

```
company name is Infosys
company address is E-City
company no of employees are 2000
********************
company name is Infosys
company address is E-City
company no of employees are 2000
********************
sum is: 20
sum is: 23
********************
sum is: 30
sum is: 24
Average is: 10.0
```

In [160]:

```
# deleting the variables

class Demo:
    def __init__(self):
        self.x=10
        self.y=20
        self.z=30

    def delete(self):
        del self.x
        del self.y

d1=Demo()

print(d1.__dict__)
print(".........")
d1.delete()
print(d1.__dict__)

del d1.z
print(".........")
print(d1.__dict__)
```

```
{'x': 10, 'y': 20, 'z': 30}
.........
{'z': 30}
.........
{}
```

In [161]:

```python
# deleting the variables

class Demo:
    def __init__(self):
        self.x=10
        self.y=20
        self.z=30

d1=Demo()
d2=Demo()

del d1.x
del d2.y

print(d1.__dict__)
print(".........")
print(d2.__dict__)
```

```
{'y': 20, 'z': 30}
.........
{'x': 10, 'z': 30}
```

In [ ]:

```python
'''

Setter and Getter Functions (Alternative to Constructor)

def setvariableName(self, variableName):
    self.variableName=variableName

def getvariableName(self):
    return self.variableName

1.Inside the Constructor(__init__) we can not write validation but in getter fun
ction we can write the validation

'''
```

In [164]:

```
# Setter and Getter Way

class CompanyDetails:

    def setName(self, name):
        self.name=name

    def getName(self):
        return self.name

    def setAddress(self, address):
        self.address=address

    def getAddress(self):
        return self.address

    def setNo_of_Employees(self, no_of_emp):
        self.no_of_emp=no_of_emp

    def getNo_of_Employees(self):
        return self.no_of_emp

c2=CompanyDetails()

c2.setName("TCS")
c2.setAddress("E-City")
c2.setNo_of_Employees(2000)

print("company name is:",c2.getName())
print("company address is:",c2.getAddress())
print("company no of employees is:",c2.getNo_of_Employees())
```

```
company name is: TCS
company address is: E-City
company no of employees is: 2000
```

In [167]:

```
# Inheritance in Python

# extending the functionality from parent to child class and code re-usability

# if child class dont want or not satisfied with the parent implementation then
 in child also we can create
# the method here

# syntax -

# class childclassname(parentclass1, parentclass2....)

class PoliticalFamily:
    def givingProperty(self):
        print("Cash + gold + diamond + Farm House + Land + business")

    def kidMarriage(self):
        print("family is doing marriage of kid with Rani")

class FamilyKid(PoliticalFamily):
    pass  # Note: Use the pass keyword when you do not want to add any other pro
perties or
          #methods to the class.

kid=FamilyKid()

kid.givingProperty()
kid.kidMarriage()
```

```
Cash + gold + diamond + Farm House + Land + business
family is doing marriage of kid with Rani
```

In [168]:

```
# if child class is not satisfied with parent class method then child can overri
de that method in parent class

class PoliticalFamily:
    def givingProperty(self):
        print("Cash + gold + diamond + Farm House + Land + business")

    def kidMarriage(self):
        print("family is doing marriage of kid with Rani")

class FamilyKid(PoliticalFamily):
    def kidMarriage(self):
        print("Kid is doing marriage with Katrina.......")

kid=FamilyKid()

kid.givingProperty()
kid.kidMarriage()
```

```
Cash + gold + diamond + Farm House + Land + business
Kid is doing marriage with Katrina.......
```

In [187]:

```
# if child wants to call the parent class method also then we can use super()

class PoliticalFamily:
    def givingProperty(self):
        print("Cash + gold + diamond + Farm House + Land + business")

    def kidMarriage(self):
        print("family is doing marriage of kid with Rani")

class FamilyKid(PoliticalFamily):
    def kidMarriage(self):
        print("Kid is doing marriage with Katrina.......")
        super().kidMarriage()

kid=FamilyKid()

kid.givingProperty()
kid.kidMarriage()
```

```
Cash + gold + diamond + Farm House + Land + business
Kid is doing marriage with Katrina.......
family is doing marriage of kid with Rani
```

In [279]:

```
class Bird:

    def __init__(self):
        self.hungry = 1

    def eat(self):
        if self.hungry == 1:
            print('hungry....')
            self.hungry = 0
        else:
            print('No, thanks!')

class SongBird(Bird):

    def __init__(self):
        #super(SongBird, self).__init__()
        super().__init__()                  # same as super(SongBird, self).__
init__()
        self.sound = 'Squawk!'

    def sing(self):
        print(self.sound)

sb = SongBird()

sb.sing()
sb.eat()
sb.eat()
```

```
Squawk!
hungry....
No, thanks!
```

In [11]:

```python
class Person:

    def __init__(self, personName, personAge):
        self.name = personName
        self.age = personAge

    def showName(self):
        print(self.name)    # directly prints the data

    def showAge(self):
        print(self.age)

class Student:

    def __init__(self, studentId):
        self.studentId  = studentId

    def getId(self):
        return self.studentId  # should be used in a print statement to print th
e studentId

class Resident(Person, Student):

    def __init__(self, name, age, id):
        Person.__init__(self, name, age)
        Student.__init__(self, id)

resident1 = Resident('John', 30, '102')
resident1.showName()
resident1.showAge()
#resident1.getId()
print(resident1.getId())
```

```
John
30
102
```

In [287]:

```
# calling parent class members from child class - Normal Type

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

class Student(Person):
    def __init__(self, name, age, rollno, marks):
        self.name=name
        self.age=age
        self.rollno=rollno
        self.marks=marks

    def showStudent(self):
        print("Student Name :",self.name)
        print("Student Age :",self.age)
        print("Student RollNo :",self.rollno)
        print("Student Marks :",self.marks)

class Teacher(Person):
    def __init__(self, name, age, salary, subject):
        self.name=name
        self.age=age
        self.salary=salary
        self.subject=subject

    def showTeacher(self):
        print("Teacher Name :",self.name)
        print("Teacher Age :",self.age)
        print("Teacher Salary :",self.salary)
        print("Teacher Subject :",self.subject)

s1=Student("Raj",23, 110, 80)

s1.showStudent()

print("***************")

p1=Teacher("Ramesh", 40, 60000, "Python")

p1.showTeacher()

# here the name and age are by default will come to child class and code we have
to write multiple time

# if parent class have 100 properties then in every child class have to take 100
proper intialization is

# coming in child
```

```
Student Name : Raj
Student Age : 23
Student RollNo : 110
Student Marks : 80
***************
Teacher Name : Ramesh
Teacher Age : 40
Teacher Salary : 60000
Teacher Subject : Python
```

In [317]:

```
# calling class members of one class in another class - no inheritance involved
 here

class Person:

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

class Student():

    def __init__(self, name, age, rollno, marks):
        self.name=name
        self.age=age
        self.rollno=rollno
        self.marks=marks

    def showStudent(self):
        print("Student Name :",self.name)
        print("Student Age :",self.age)
        print("Student RollNo :",self.rollno)
        print("Student Marks :",self.marks)

class Teacher():

    def __init__(self, name, age, salary, subject):
        self.name=name
        self.age=age
        self.salary=salary
        self.subject=subject

    def showTeacher(self):
        print("Teacher Name :",self.name)
        print("Teacher Age :",self.age)
        print("Teacher Salary :",self.salary)
        print("Teacher Subject :",self.subject)

s1=Student("Raj", 23, 110, 80)    # Student.__init__(s1, "Raj", 23, 110, 80)

s1.showStudent()                  # Student.showStudent(s1)

print("***************")

p1=Teacher("Ramesh", 40, 60000, "Python") # Teacher.__init__(p1, "Raj", 23, 110,
80)

p1.showTeacher()                          # Teacher.showTeacher(p1)
```

```
Student Name : Raj
Student Age : 23
Student RollNo : 110
Student Marks : 80
***************
Teacher Name : Ramesh
Teacher Age : 40
Teacher Salary : 60000
Teacher Subject : Python
```

In [339]:

```
# calling parent class members from child class - Analysis example without __ini
t__ constructor

class Person:
    def getNameAge(self, name, age):
        self.name=name
        self.age=age

class Student(Person):

    def getRollnomarks(self, rollno, marks):
        self.rollno=rollno
        self.marks=marks

    def showStudent(self):
        print("Student Name :",self.name)
        print("Student Age :",self.age)
        print("Student RollNo :",self.rollno)
        print("Student Marks :",self.marks)

class Teacher(Person):

    def getsalarysubject(self, salary, subject):
        self.salary=salary
        self.subject=subject

    def showTeacher(self):
        print("Student Name :",self.name)
        print("Student Age :",self.age)
        print("Teacher Salary :",self.salary)
        print("Teacher Subject :",self.subject)

s11=Student()

Person.getNameAge(s11, "Raj", 23)    # Student.__init__(s1, "Raj", 23, 110, 80)

Student.getRollnomarks(s11, 110, 80)    # Student.__init__(s1, "Raj", 23, 110, 8
0)

s11.showStudent()                  # Student.showStudent(s1)

print("***************")

t11=Teacher()

Person.getNameAge(t11, "Ramesh", 40)    # Teacher.__init__(t1, "Ramesh", 40, 600
00, "Python")

Teacher.getsalarysubject(t11, 60000, "Python")  # Teacher.__init__(t1, "Ramesh",
40, 60000, "Python")

t11.showTeacher()                          # Teacher.showTeacher(t1)

# here the name and age are by default will come to child class and code we have
to write multiple time

# if parent class have 100 properties then in every child class have to take 100
proper intialization is
```

```python
# coming in child
```

```
Student Name : Raj
Student Age : 23
Student RollNo : 110
Student Marks : 80
***************
Student Name : Ramesh
Student Age : 40
Teacher Salary : 60000
Teacher Subject : Python
```

In [347]:

```
# calling parent class members from child class - Analysis example with __init__
constructor - Correct Example
# better understanding

class Person:
    def __init1__(self):
    #def __init__(self):
        self.name="Srinivasan"
        self.age=34
        print("Student Name :",self.name)
        print("Student Age :",self.age)

class Student(Person):

    def __init__(self):
        self.rollno=110
        self.marks=80

    def showStudent(self):
        print("Student RollNo :",self.rollno)
        print("Student Marks :",self.marks)

class Teacher(Person):

    def __init__(self):
        self.salary=60000
        self.subject="Python"

    def showTeacher(self):
        print("Teacher Salary :",self.salary)
        print("Teacher Subject :",self.subject)

s22=Student()

s22.__init1__()

s22.showStudent()                    # Student.showStudent(s1)

print("***************")

t22=Teacher()

t22.__init1__()

t22.showTeacher()                            # Teacher.showTeacher(t1)

# here the name and age are by default will come to child class and code we have
to write multiple time

# if parent class have 100 properties then in every child class have to take 100
proper intialization is

# coming in child
```

```
Student Name : Srinivasan
Student Age : 34
Student RollNo : 110
Student Marks : 80
***************
Student Name : Srinivasan
Student Age : 34
Teacher Salary : 60000
Teacher Subject : Python
```

In [288]:

```
# calling class members of one class in another class - no inheritance involved
 here

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

class Student():
    def __init__(self, name, age, rollno, marks):
        Person.__init__(self, name, age)      # Person.__init__(s1, "Raj", 23)
        self.rollno=rollno
        self.marks=marks

    def showStudent(self):
        print("Student Name :",self.name)
        print("Student Age :",self.age)
        print("Student RollNo :",self.rollno)
        print("Student Marks :",self.marks)

class Teacher():
    def __init__(self, name, age, salary, subject):
        Person.__init__(self, name, age)       # Person.__init__(p1, "Raj", 23,
 110, 80)
        self.salary=salary
        self.subject=subject

    def showTeacher(self):
        print("Teacher Name :",self.name)
        print("Teacher Age :",self.age)
        print("Teacher Salary :",self.salary)
        print("Teacher Subject :",self.subject)

s1=Student("Raj", 23, 110, 80)    # Student.__init__(s1, "Raj", 23, 110, 80)

s1.showStudent()                  # Student.showStudent(s1)

print("***************")

p1=Teacher("Ramesh", 40, 60000, "Python") # Teacher.__init__(p1, "Raj", 23, 110,
80)

p1.showTeacher()                          # Teacher.showTeacher(p1)
```

```
Student Name : Raj
Student Age : 23
Student RollNo : 110
Student Marks : 80
***************
Teacher Name : Ramesh
Teacher Age : 40
Teacher Salary : 60000
Teacher Subject : Python
```

In [ ]:

```
# calling class members of one class in another class - Type 2

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

class Student(Person):
    def __init__(self, name, age, rollno, marks):
        Person.__init__(self, name, age)      # Person.__init__(s1, "Raj", 23)
        self.rollno=rollno
        self.marks=marks

    def showStudent(self):
        print("Student Name :",self.name)
        print("Student Age :",self.age)
        print("Student RollNo :",self.rollno)
        print("Student Marks :",self.marks)

class Teacher(Person):
    def __init__(self, name, age, salary, subject):
        Person.__init__(self, name, age)       # Person.__init__(p1, "Raj", 23,
 110, 80)
        self.salary=salary
        self.subject=subject

    def showTeacher(self):
        print("Teacher Name :",self.name)
        print("Teacher Age :",self.age)
        print("Teacher Salary :",self.salary)
        print("Teacher Subject :",self.subject)

s1=Student("Raj", 23, 110, 80)    # Student.__init__(s1, "Raj", 23, 110, 80)

s1.showStudent()                  # Student.showStudent(s1)

print("***************")

p1=Teacher("Ramesh", 40, 60000, "Python") # Teacher.__init__(p1, "Raj", 23, 110,
80)

p1.showTeacher()                          # Teacher.showTeacher(p1)
```

In [282]:

```
# super() method - to call parent class members(__init__ alone) from child class
we use super() method

class Person:
    def __init__(self, name, age):
        print("Super Constructor is called")
        self.name=name
        self.age=age

class Student(Person):
    def __init__(self, name, age, rollno, marks):
        super().__init__(name, age)
        self.rollno=rollno
        self.marks=marks

    def showStudent(self):
        print("Student Name :",self.name)
        print("Student Age :",self.age)
        print("Student RollNo :",self.rollno)
        print("Student Marks :",self.marks)


class Teacher(Person):
    def __init__(self, name, age, salary, subject):
        super().__init__(name, age)
        self.salary=salary
        self.subject=subject

    def showTeacher(self):
        print("Teacher Name :",self.name)
        print("Teacher Age :",self.age)
        print("Teacher Salary :",self.salary)
        print("Teacher Subject :",self.subject)

s1=Student("Raj",23, 110, 80)

s1.showStudent()

print("***************")

p1=Teacher("Ramesh", 40, 60000, "Python")

p1.showTeacher()

# here the name and age are by default will come to child class and code we have
to write multiple time

# if parent class have 100 properties then in every child class have to take 100
proper intialization is

# coming in child
```

```
Super Constructor is called
Student Name : Raj
Student Age : 23
Student RollNo : 110
Student Marks : 80
***************
Super Constructor is called
Teacher Name : Ramesh
Teacher Age : 40
Teacher Salary : 60000
Teacher Subject : Python
```

In [13]:

```
# super() method - to call parent class members(__init__ and print) from child c
lass we use super() method

class Person:
    def __init__(self, name, age):
        print("Super Constructor is called")
        self.name=name
        self.age=age

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

class Student(Person):
    def __init__(self, name, age, rollno, marks):
        super().__init__(name, age)
        self.rollno=rollno
        self.marks=marks

    def showStudent(self):
        Person.showPerson(self)
        print("Student RollNo :",self.rollno)
        print("Student Marks :",self.marks)

class Teacher(Person):
    def __init__(self, name, age, salary, subject):
        super().__init__(name, age)
        self.salary=salary
        self.subject=subject

    def showTeacher(self):
        Person.showPerson(self)
        print("Teacher Salary :",self.salary)
        print("Teacher Subject :",self.subject)

s1=Student("Raj",23, 110, 80)

s1.showStudent()

print("***************")

p1=Teacher("Ramesh", 40, 60000, "Python")

p1.showTeacher()

# here the name and age are by default will come to child class and code we have
to write multiple time

# if parent class have 100 properties then in every child class have to take 100
proper intialization is

# coming in child
```

```
Super Constructor is called
Name : Raj
Age : 23
Student RollNo : 110
Student Marks : 80
***************
Super Constructor is called
Name : Ramesh
Age : 40
Teacher Salary : 60000
Teacher Subject : Python
```

In [212]:

```python
class Company1:
    def companyinfo(self):
        print("Company1 info is here")

class Company2(Company1):
    def companyinfo(self):
        print("Company2 info is here")
        super().companyinfo()

class Company3(Company2):
    def companyinfo(self):
        print("Company3 info is here")
        super().companyinfo()

class Company4(Company3):
    def companyinfo(self):
        print("Company4 info is here")
        super().companyinfo()

class Company5(Company4):
    def companyinfo(self):
        print("Company5 info is here")
        super().companyinfo()

c1=Company5()

c1.companyinfo()
```

```
Company5 info is here
Company4 info is here
Company3 info is here
Company2 info is here
Company1 info is here
```

In [213]:

```
class Company1:
    def companyinfo(self):
        print("Company1 info is here")

class Company2(Company1):
    def companyinfo(self):
        print("Company2 info is here")
        super().companyinfo()

class Company3(Company2):
    def companyinfo(self):
        print("Company3 info is here")
        super().companyinfo()

class Company4(Company3):
    def companyinfo(self):
        print("Company4 info is here")

class Company5(Company4):
    def companyinfo(self):
        print("Company5 info is here")
        super().companyinfo()

c1=Company5()

c1.companyinfo()
```

```
Company5 info is here
Company4 info is here
```

In [214]:

```
class Company1:
    def companyinfo(self):
        print("Company1 info is here")

class Company2(Company1):
    def companyinfo(self):
        print("Company2 info is here")
        super().companyinfo()

class Company3(Company2):
    def companyinfo(self):
        print("Company3 info is here")
        super().companyinfo()

class Company4(Company3):
    pass

class Company5(Company4):
    def companyinfo(self):
        print("Company5 info is here")
        super().companyinfo()

c1=Company5()

c1.companyinfo()
```

```
Company5 info is here
Company3 info is here
Company2 info is here
Company1 info is here
```

In [217]:

```
class Company1:
    def companyinfo(self):
        print("Company1 info is here")

class Company2(Company1):
    def companyinfo(self):
        print("Company2 info is here")

class Company3(Company2):
    def companyinfo(self):
        print("Company3 info is here")

class Company4(Company3):
    pass

class Company5(Company4):
    def companyinfo(self):
        super().companyinfo()

c1=Company5()

# c1.companyinfo()
```

```
Company3 info is here
```

In [14]:

```
# calling a particular parent class method

class Company1:
    def companyinfo(self):
        print("Company1 info is here")

class Company2(Company1):
    def companyinfo(self):
        print("Company2 info is here")

class Company3(Company2):
    def companyinfo(self):
        print("Company3 info is here")

class Company4(Company3):
    def companyinfo(self):
        print("Company4 info is here")

class Company5(Company4):
    def companyinfo(self):
        #super().companyinfo()
        super(Company2, self).companyinfo()

c1=Company5()

c1.companyinfo()
```

```
Company4 info is here
```

In [20]:

```
class Parent:

    x=10                    # here x is a static variable

    def __init__(self):     # constructor
        self.y=20
        print(self.y)

class Child(Parent):

    def __init__(self):     # constructor
        Parent.__init__(self)
        self.z=30
        print(self.z)

# even though Parent class is inherited in the child class it has to be intializ
ed again
# in the child class if __init__constructor is available in the child class

    def display(self):

        print(self.x)

c3=Child()
print("Value of x is:",c3.x)
print("Value of y is:",c3.y)
c3.display()
print("Value of x is:",c3.x)
print("Value of y is:",c3.y)
```

```
20
30
Value of x is: 10
Value of y is: 20
10
Value of x is: 10
Value of y is: 20
```

In [26]:

```
# from child constructor can we call the parent constructor, class method, stati
c method, instance method

# here super() is  used insted of the parent class name


class Demo:
    def __init__(self):
        print("parent constructor got executed")

    def test1(self):
        print("parent instance got executed")

    def test2(cls):
        print("parent class method got executed")

    def test3(cls):
        print("parent static method got executed")

class Child(Demo):

    def __init__(self):
        super().__init__()
        super().test1()
        super().test2()
        super().test3()

c1=Child()
```

```
parent constructor got executed
parent instance got executed
parent class method got executed
parent static method got executed
```

In [10]:

```
# from child instance method can we call the constructor, class method, static m
ethod, instance method
# can be called using super()


class Demo:
    def __init__(self):
        print("parent constructor got executed")

    def test1(self):
        print("parent instance got executed")

    @classmethod
    def test2(self):
        print("parent class method got executed")

    #@staticmethod
    def test3():
        print("parent static method got executed")

class Child(Demo):

    def display(self):
        #super().__init__()
        super().test1()
        super().test2()
        test3()

c1=Child()
c1.display()
```

```
parent constructor got executed
parent instance got executed
parent class method got executed
parent static method got executed
```

In [11]:

```
# from child class method can we call the constructor, class method, static meth
od, instance method
# can be called using super()

# Doubt - ?????????????


class Demo:
    def __init__(self):                              # this method will be called by
default, as it is a constructor
        print("parent constructor got executed") # but calling for the second ti
me using super() throws an error
                                                 # since self parameter is used
 and in the function call cls
    def test1(self):
        print("parent instance got executed")    # this method is a instance met
hod which cannot be called
                                                 # from a class method, since se
lf parameter is used and in the
                                                 # function call cls
    @classmethod
    def test2(cls):                              # this method is a class method
which can be called
        print("parent class method got executed")# from a class method

    @staticmethod
    def test3():                                   # this method is a static met
hod which can be called
        print("parent static method got executed") # from a class method

class Child(Demo):

    @classmethod
    def display(cls):
        super().__init__()
        super().test1()
        super().__init__(cls)
        super().test1(cls)
        #super().test2()
        #super().test3()

c1=Child()
c1.display()
```

```
parent constructor got executed

---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-11-cc62a66b9d64> in <module>
     33 
     34 c1=Child()
---> 35 c1.display()

<ipython-input-11-cc62a66b9d64> in display(cls)
     25     @classmethod
     26     def display(cls):
---> 27         super().__init__()
     28         super().test1()
     29         super().__init__(cls)

TypeError: __init__() missing 1 required positional argument: 'self'
```

In [9]:

```python
# how to call parent class constructor, instance method from the class method of child class

# Solution for the above problem - ???? - delf or self just a name here

# Doubt - ?????????????

class Demo:
    def __init__(self):
        print("parent constructor got executed")

    def test1(self):
        print("parent instance got executed")

    @classmethod
    def test2(cls):
        print("parent class method got executed")

    @staticmethod
    def test3():
        print("parent static method got executed")

class Child(Demo):

    @classmethod
    def display(delf):
    #def display(self):
        super(Child, delf).__init__(delf)
        super(Child, delf).test1(delf)
        super().test2()
        super().test3()

c1=Child()
c1.display()
```

```
parent constructor got executed
parent constructor got executed
parent instance got executed
parent class method got executed
parent static method got executed
```

In [393]:

```
# from child static method can we call the constructor, class method, static met
hod, instance method
# can be called using super()

# Doubt - ?????????????


class Demo:
    def __init__(self):                             # this method will be called by
default, as it is a constructor
        print("parent constructor got executed") # but calling for the second ti
me using super() throws an error
                                                # since self parameter is used
 and in the function call cls
    def test1(self):
        print("parent instance got executed")    # this method is a instance met
hod which cannot be called
                                                # from a class method, since se
lf parameter is used and in the
                                                # function call cls
    @classmethod
    def test2(cls):                             # this method is a class method
which can be called
        print("parent class method got executed")# from a class method

    @staticmethod
    def test3():                                  # this method is a static met
hod which can be called
        print("parent static method got executed") # from a class method

class Child(Demo):

    @staticmethod
    def display():
        super().__init__()
        super().test1()
        super().test2()
        super().test3()

c1=Child()
c1.display()
```

```
parent constructor got executed

---------------------------------------------------------------------
-------
RuntimeError                              Traceback (most recent cal
l last)
<ipython-input-393-5ec637b7ec63> in <module>
     29 
     30 c1=Child()
---> 31 c1.display()
     32 
     33 # parent constructor got executed  - this output is because
 of __init__() method, which gets

<ipython-input-393-5ec637b7ec63> in display()
     23     @staticmethod
     24     def display():
---> 25         super().__init__()
     26         super().test1()
     27         super().test2()

RuntimeError: super(): no arguments
```

In [402]:

```
# from child static method can we call the constructor, class method, static met
hod, instance method
# can be called using super()

# Doubt - ?????????????



class Demo:
    def __init__(self):                                 # this method will be called by
default, as it is a constructor
        print("parent constructor got executed") # but calling for the second ti
me using super() throws an error
                                                 # since self parameter is used
 and in the function call cls
    def test1(self):
        print("parent instance got executed")    # this method is a instance met
hod which cannot be called
                                                 # from a class method, since se
lf parameter is used and in the
                                                 # function call cls
    @classmethod
    def test2(cls):                              # this method is a class method
which can be called
        print("parent class method got executed")# from a class method

    @staticmethod
    def test3():                                   # this method is a static met
hod which can be called
        print("parent static method got executed") # from a class method

class Child(Demo):

    @staticmethod
    def display():
        #super().__init__()
        super(Child,Child).test1()
        super(Child,Child).test2()
        super(Child,Child).test3()

c1=Child()
c1.display()

# Child, Child - go to super class of child and then call the static method with
classname so 2 args
```

```
parent constructor got executed

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-402-4d75a4c53b8d> in <module>
     29 
     30 c1=Child()
---> 31 c1.display()
     32 
     33 # Child, Child - go to super class of child and then call the static method with classname so 2 args

<ipython-input-402-4d75a4c53b8d> in display()
     24     def display():
     25         #super().__init__()
---> 26         super(Child,Child).test1()
     27         super(Child,Child).test2()
     28         super(Child,Child).test3()

TypeError: test1() missing 1 required positional argument: 'self'
```

In [8]:

```
# Abstraction in Python
# abc => abstract base calss module

from abc import ABC, abstractmethod
#from abc import *
class Demo:
    @abstractmethod     # creating an abstract method
    def demomethod(self):
        pass
```

In [407]:

```
dir(abc)
```

Out[407]:

```
['ABC',
 'ABCMeta',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 '_abc_init',
 '_abc_instancecheck',
 '_abc_register',
 '_abc_subclasscheck',
 '_get_dump',
 '_reset_caches',
 '_reset_registry',
 'abstractclassmethod',
 'abstractmethod',
 'abstractproperty',
 'abstractstaticmethod',
 'get_cache_token']
```

In [3]:

```
import abc

class Demo():
    pass

d1=Demo()
```

```
In [4]:
```

```
#creating abstract class by extendding ABC class
import abc

class Demo(ABC):  # extending ABC class to create abstract class
    pass

d1=Demo()
```

```
--------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-4-f3e1252c367a> in <module>
      2 import abc
      3
----> 4 class Demo(ABC):  # extending ABC class to create abstract c
lass
      5     pass
      6

NameError: name 'ABC' is not defined
```

```
In [414]:
```

```
import abc

class Demo(ABC):    # extending ABC class to create abstract class

    @abstractmethod # creating an abstract method
    def demomethod(self):
        pass

d1=Demo()
```

```
--------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-414-176d8ab1010a> in <module>
      7         pass
      8
----> 9 d1=Demo()

TypeError: Can't instantiate abstract class Demo with abstract metho
ds demomethod
```

In [5]:

```
import abc

#class Demo(ABC):   # extending ABC class to create abstract class
class Demo():

    @abstractmethod # creating an abstract method
    def demomethod(self):
        pass

d1=Demo()
```

```
---------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-5-e32f01f2ae18> in <module>
      2
      3 #class Demo(ABC):   # extending ABC class to create abstract
class
----> 4 class Demo():
      5
      6     @abstractmethod # creating an abstract method

<ipython-input-5-e32f01f2ae18> in Demo()
      4 class Demo():
      5
----> 6     @abstractmethod # creating an abstract method
      7     def demomethod(self):
      8         pass

NameError: name 'abstractmethod' is not defined
```

In [7]:

```
# Eventhough the child class doesn't implement the abstract method of parent cla
ss, But its an abstract class
# then it cant create object for itself (child class)

import abc

class Musician(abc):
    def payment(self):
        print("Payment is happening for all musicians.......")

    @abstractmethod
    def thememusic(self):
        pass

class Guitarplayer(Musician):
    pass

gp=Guitarplayer()
#m1=Musician()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-7-23301fc66821> in <module>
      4 import abc
      5
----> 6 class Musician(abc):
      7     def payment(self):
      8         print("Payment is happening for all musician
s.......")

<ipython-input-7-23301fc66821> in Musician()
      8         print("Payment is happening for all musician
s.......")
      9
---> 10     @abstractmethod
     11     def thememusic(self):
     12         pass

NameError: name 'abstractmethod' is not defined
```

In [425]:

```
import abc

class Musician(ABC):
    def payment(self):
        print("Payment is happening for all musicians.......")

    @abstractmethod
    def thememusic(self):
        pass

class Guitarplayer(Musician):
    #def thememusic1(self):         # here the thememusic method of the child cl
ass overrides the abstract
                                    # method of the parent class, so it works he
re
    def thememusic(self):
        print("Strum Strum Strum Strum Strum.........")
    #pass

gp=Guitarplayer()
#gp.thememusic1()
gp.thememusic()
```

```
Strum Strum Strum Strum Strum.........
```

In [426]:

```
import abc

class Musician(ABC):
    def payment(self):
        print("Payment is happening for all musicians.......")

    @abstractmethod
    def thememusic(self):
        pass

class Guitarplayer(Musician):
    #def thememusic1(self):         # here the thememusic method of the child cl
ass overrides the abstract
                                    # method of the parent class, so it works he
re
    def thememusic(self):
        print("Strum Strum Strum Strum Strum.........")
    #pass

class Drumplayer(Musician):
    #def thememusic1(self):         # here the thememusic method of the child cl
ass overrides the abstract
                                    # method of the parent class, so it works he
re
    def thememusic(self):
        print("Dum Dum Dum Dum Dum.........")
    #pass

gp=Guitarplayer()
#gp.thememusic1()
gp.thememusic()

dp=Drumplayer()
#gp.thememusic1()
dp.thememusic()
```

```
Strum Strum Strum Strum Strum.........
Dum Dum Dum Dum Dum.........
```

In [431]:

```
# interfaces - if and abstract class contains only abstract methods such type of
abstract class
# is considered as interface

import abc

class DBInterface(ABC):   # this class is an interface

    @abstractmethod
    def connectdatabase(self):
        pass

    @abstractmethod
    def disconnectdatabase(self):
        pass

class MySQLDB(DBInterface):

    def connectdatabase(self):      # here the connectdatabase method of the chi
ld class overrides the abstract
                                    # method of the parent class, so it works he
re
        print("Connect with the mysql database.....")

    def disconnectdatabase(self):   # here the connectdatabase method of the chi
ld class overrides the abstract
                                    # method of the parent class, so it works he
re
         print("Disconnect with the mysql database.....")

class OracleDB(DBInterface):

    def connectdatabase(self):      # here the connectdatabase method of the chi
ld class overrides the abstract
                                    # method of the parent class, so it works he
re
        print("Connect with the oracle database.....")

    def disconnectdatabase(self):   # here the connectdatabase method of the chi
ld class overrides the abstract
                                    # method of the parent class, so it works he
re
         print("Disconnect with the oracle database.....")

dbname=input("Enter Database name:")
print(type(dbname))
classname=globals()[dbname]
x=classname()
x.connectdatabase()
x.disconnectdatabase()

# The inbuilt function globals()[str] converts the string 'str' into a class nam
e and returns the classname.
```

```
Enter Database name:MySQLDB
<class 'str'>
Connect with the mysql database.....
Disconnect with the mysql database.....
```

In [436]:

```
name = 'Pankaj'

print(globals()['name'])

# print(globals()[name]) - this will convert the string 'Pankaj' into a class na
me, But the same
# cannot be printed

# This function returns the dictionary of the current module.
```

```
Pankaj
```

In [442]:

```
def globalstuff():
    global g1
    g1 = 'Global'


globalstuff()  # calling globalstuff() to set the variables

# print(globals())

print(globals() ['g1'])
```

```
Global
```

In [443]:

```
# public, protected, private

# Default every candidate is public. We can access from anywhere either within t
he class or from
# outside of the class(No underscore)

# Protected attributes can be accessed within the class anywhere and from the ch
ild classes(Single underscore)

# Private attributes can be accessed only from inside the class not outside clas
s(Double underscore)
```

In [444]:

```
class Demo:

    x=10     # public    - We can access from anywhere either within the class o
r from outside of the class
    _y=20    # protected - can be accessed within the class anywhere and from th
e child
    __z=30   # private   - can be accessed only from inside the class not outsid
e class

    def getvalues(self):
        print(Demo.x)
        print(Demo._y)
        print(Demo.__z)

d1=Demo()
d1.getvalues()
```

```
10
20
30
```

In [447]:

```
class Demo:

    x=10     # public    - We can access from anywhere either within the class o
r from outside of the class
    _y=20    # protected - can be accessed within the class anywhere and from th
e child
    __z=30   # private   - can be accessed only from inside the class not outsid
e class

    def getvalues(self):
        print(Demo.x)
        print(Demo._y)
        print(Demo.__z)

d1=Demo()
d1.getvalues()

print("**********")

print(Demo.x)
print(Demo._y)
print(Demo.__z)
```

```
10
20
30
**********
10
20

---------------------------------------------------------------------
-------
AttributeError                            Traceback (most recent cal
l last)
<ipython-input-447-54ecb67a2e71> in <module>
     17 print(Demo.x)
     18 print(Demo._y)
---> 19 print(Demo.__z)

AttributeError: type object 'Demo' has no attribute '__z'
```

In [456]:

```
class Demo:

    x=10     # public    - We can access from anywhere either within the class o
r from outside of the class
    _y=20    # protected - can be accessed within the class anywhere and from th
e child
    __z=30   # private   - can be accessed only from inside the class not outsid
e class

    def getvalues(self):
        print(Demo.x)
        print(Demo._y)
        print(Demo.__z)

d1=Demo()
d1.getvalues()

print("**********")

print(Demo.x)
print(Demo._y)
print(d1._Demo__z)
```

```
10
20
30
**********
10
20
30
```

In [29]:

```
class Demo:

    x=10     # public    - We can access from anywhere either within the class o
r from outside of the class
    _y=20    # protected - can be accessed within the class anywhere and from th
e child
    __z=30   # private   - can be accessed only from inside the class not outsid
e class

    def getvalues(self):
        print(Demo.x)
        print(Demo._y)
        print(Demo.__z)

d1=Demo()
d1.getvalues()

print("**********")

print(d1.x)
print(d1._y)
print(d1.__z)


class Demo1(Demo):

    pass

d2=Demo1()
d2.getvalues()
```

```
10
20
30
**********
10
20

---------------------------------------------------------------------
-------
AttributeError                            Traceback (most recent cal
l last)
<ipython-input-29-5142e899cddb> in <module>
     17 print(d1.x)
     18 print(d1._y)
---> 19 print(d1.__z)
     20
     21

AttributeError: 'Demo' object has no attribute '__z'
```

In [477]:

```
class Demo:

    x=10     # public    - We can access from anywhere either within the class o
r from outside of the class
    _y=20    # protected - can be accessed within the class anywhere and from th
e child
    __z=30   # private   - can be accessed only from inside the class not outsid
e class

    def getvalues(self):
        print(Demo.x)
        print(Demo._y)
        print(Demo.__z)

d1=Demo()
d1.getvalues()

print("**********")

print(Demo.x)
print(Demo._y)
print(d1._Demo__z)


class Demo1():

    pass

d2=Demo1()
d2.getvalues()
```

```
10
20
30
**********
10
20
30

---------------------------------------------------------------------
-------
AttributeError                            Traceback (most recent cal
l last)
<ipython-input-477-214901e1b6ff> in <module>
     25
     26 d2=Demo1()
---> 27 d2.getvalues()

AttributeError: 'Demo1' object has no attribute 'getvalues'
```

In [9]:

```
# here how _y which is a protected variable is still printing, __z will not prin
t as it is a private varible

class Demo:

    x=10     # public    - We can access from anywhere either within the class o
r from outside of the class
    _y=20    # protected - can be accessed within the class anywhere and from th
e child
    __z=30   # private   - can be accessed only from inside the class not outsid
e class

    def getvalues(self):
        print(Demo.x)
        print(Demo._y)
        print(Demo.__z)

d1=Demo()
d1.getvalues()

print("**********")

print(Demo.x)
print(Demo._y)
print(d1._Demo__z)


class Demo1():

    def getvalues(self):
        print(Demo.x)
        print(Demo._y)
        #print(Demo.__z)

d2=Demo1()
d2.getvalues()
```

```
10
20
30
**********
10
20
30
10
20
```

In [46]:

```
class Parent:
    def __init__(self,a):
        print(a)

class Child(Parent):
    def __init__(self,a,b):
        super().__init__(a)
        print(b)

    def show1(self,b):
        print(b)

    def show2(self,c):
        print(c)

c1=Child(10,20)
```

```
10
20
```

In [54]:

```
#global a
a=10

def srini():
    nonlocal b
    b=20
    print(a)

def vasu():
    print(b)
    print(a)

srini()
vasu()
```

```
  File "<ipython-input-54-fe4a344b2f66>", line 5
    nonlocal b
    ^
SyntaxError: no binding for nonlocal 'b' found
```

In [2]:

```
class simple:
    pass

s1 = simple()

s1.name = 'xyz'

print(s1.name)
```

```
xyz
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```