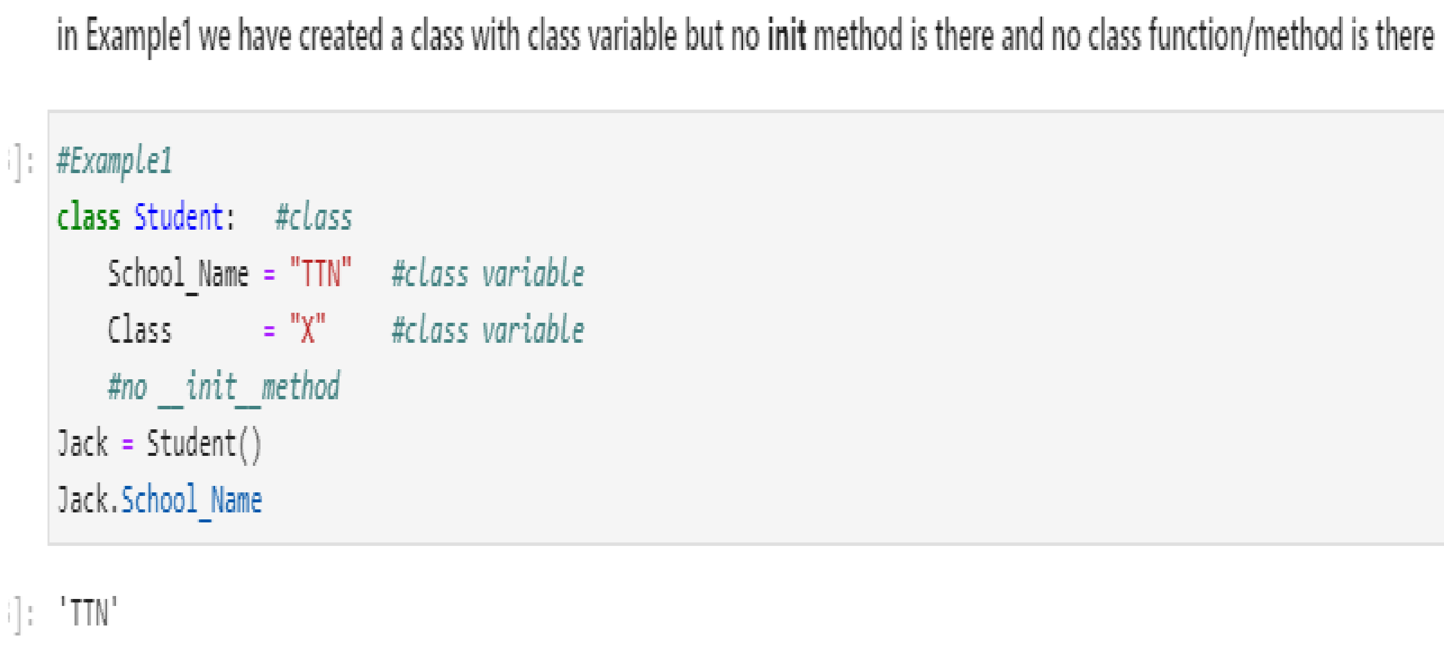
INHERITANCE

THE process of inheriting characters   i.e,   having the properties of our ancestors is called inheritance.

__init__

Two underscore before and two underscore after init

If an object is created of any class the first executed function is init.



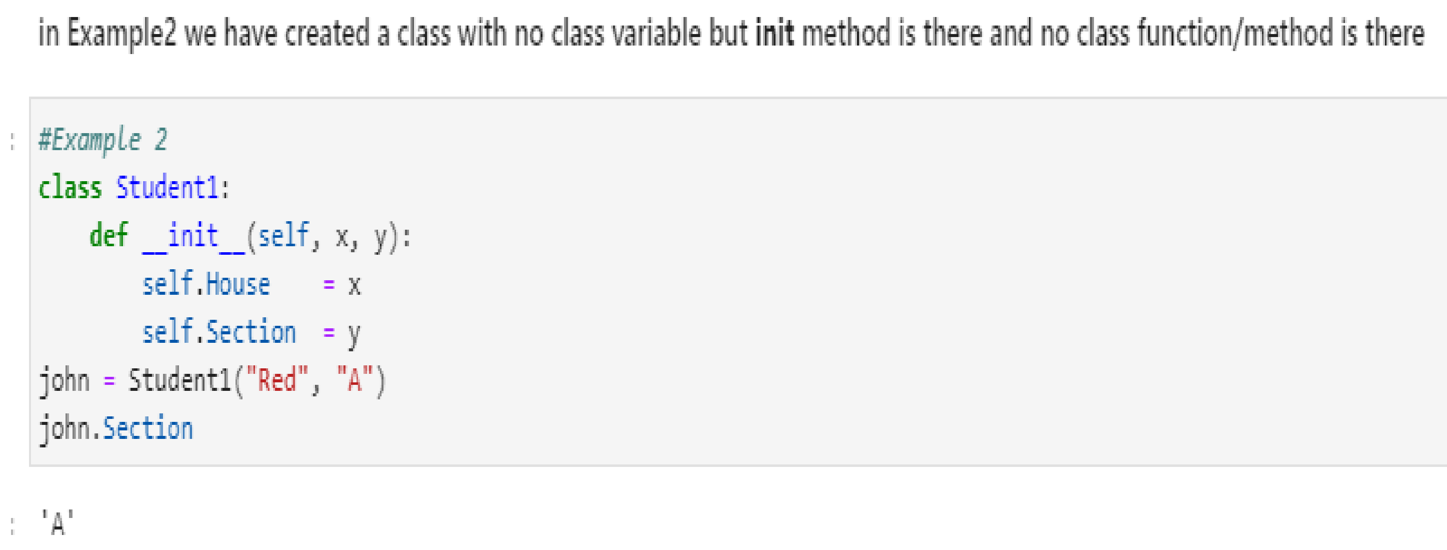
in Example1 we have created a class with class variable but no **init** method is there and no class function/method is there

```
]: #Example1
   class Student:   #class
       School_Name = "TTN"   #class variable
       Class       = "X"     #class variable
       #no __init__method
   Jack = Student()
   Jack.School_Name
```

```
]: 'TTN'
```

When the object is made of a class we have to write the the number of attributes which inside the init   method in the class parenthesis or ()   remember the attributes which we have to write inside the   class   ()   is excluding self but if the init is not given we don't have to write any attribute inside class ()   just   like the above statement when Jack = Student()   ,,   here nothing is mentioned inside the student ().If init is not there the class variables will be accessed by object _ name   and then using dot without giving space.

The process is first Object is created from the class then class variables and then the class variables go to the class give their values to the class then write object _name and when dot is put then the class variables will be shown.



in Example2 we have created a class with no class variable but **init** method is there and no class function/method is there

```
#Example 2
class Student1:
    def __init__(self, x, y):
        self.House    = x
        self.Section  = y
john = Student1("Red", "A")
john.Section
```

```
'A'
```

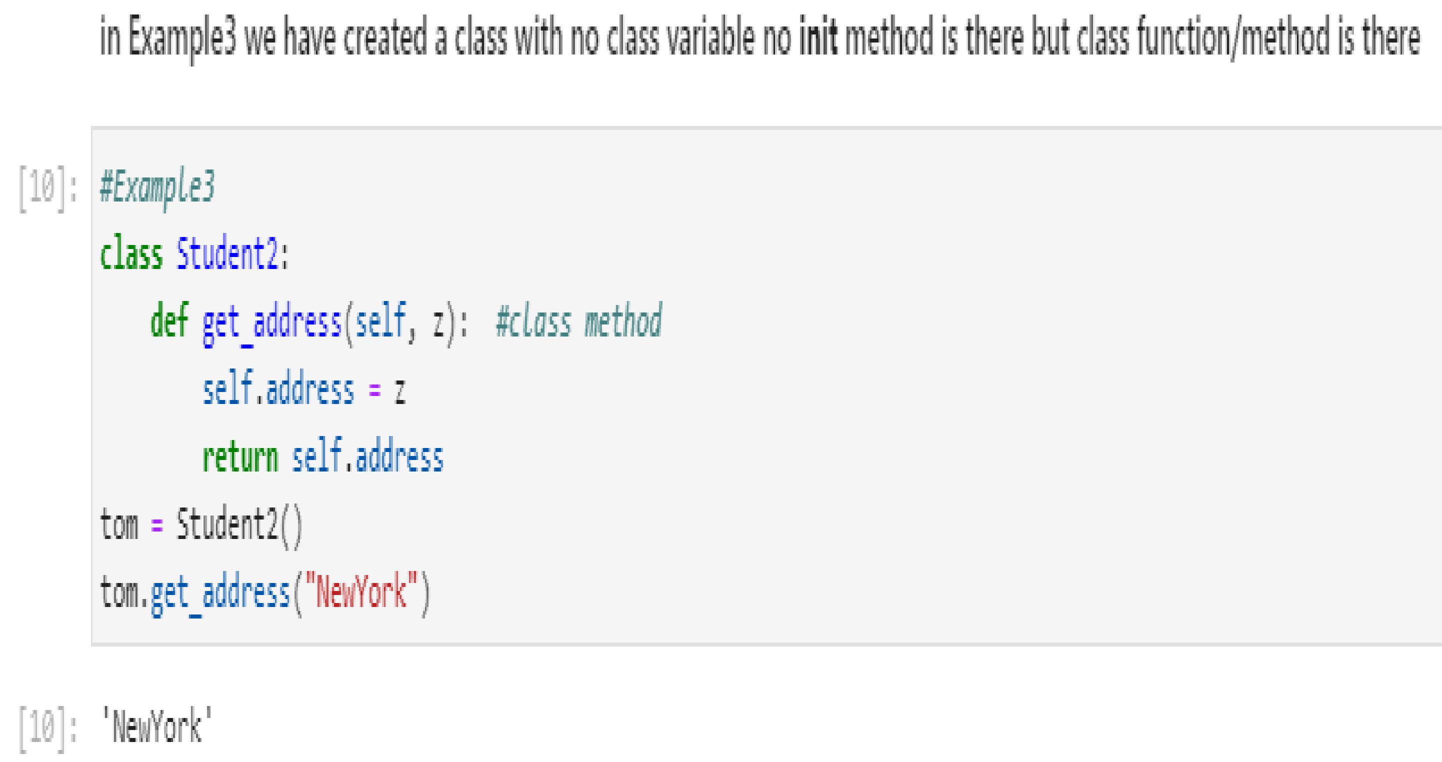
In example 2

Init has two attributes x and y (self will not be considered as an attribute),,so when we create an object as john = student() , we have to write values that is x = first value and y = second value accordingly.

Steps of execution

Create object of the created class i.e, student1 and write the attributes .Then the given values inside the () will be taken by __init__( ) and it will give the values to the x and y o by whichever name you have given and the given values will be taken by self .house as x and self.section as y and the value of self.house and self.section or if you have any other attributes will be transferred to the class directly ..

Remember in place of self we can write any other variable or name but then change the self variable in all other places accordingly as with the changed name which you have mentioned .



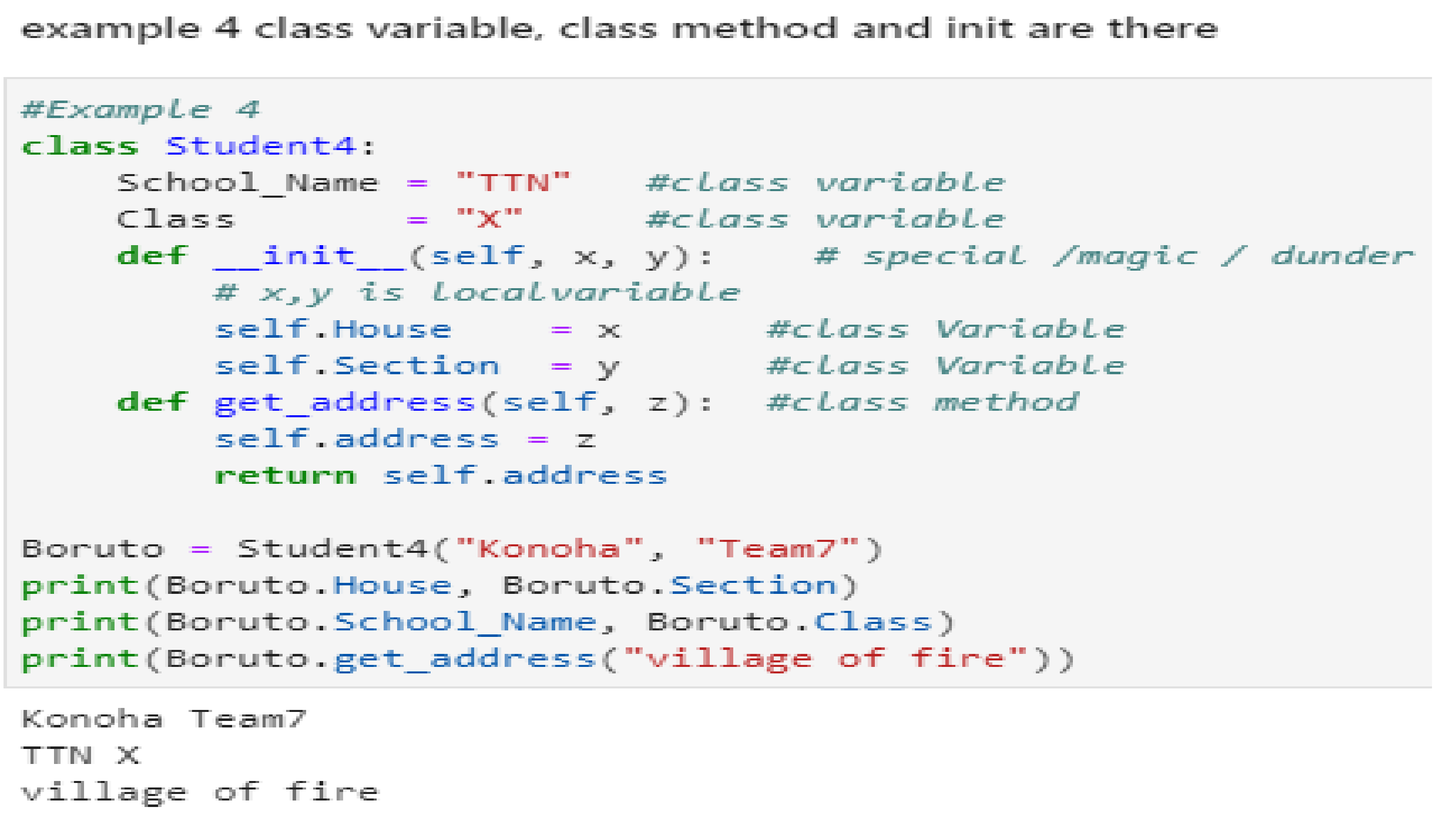
in Example3 we have created a class with no class variable no **init** method is there but class function/method is there

```
[10]: #Example3
      class Student2:
          def get_address(self, z):  #class method
              self.address = z
              return self.address
      tom = Student2()
      tom.get_address("NewYork")
```

```
[10]: 'NewYork'
```

Steps for execution

First object tom of class student2 is created but don't give any attributes inside parenthiesis() because there is no __init__ function so remember when no init function is there we don't have to write the attributes in above line mentioned parenthesis ..Then class method is executed and then further is executed .



example 4 class variable, class method and init are there

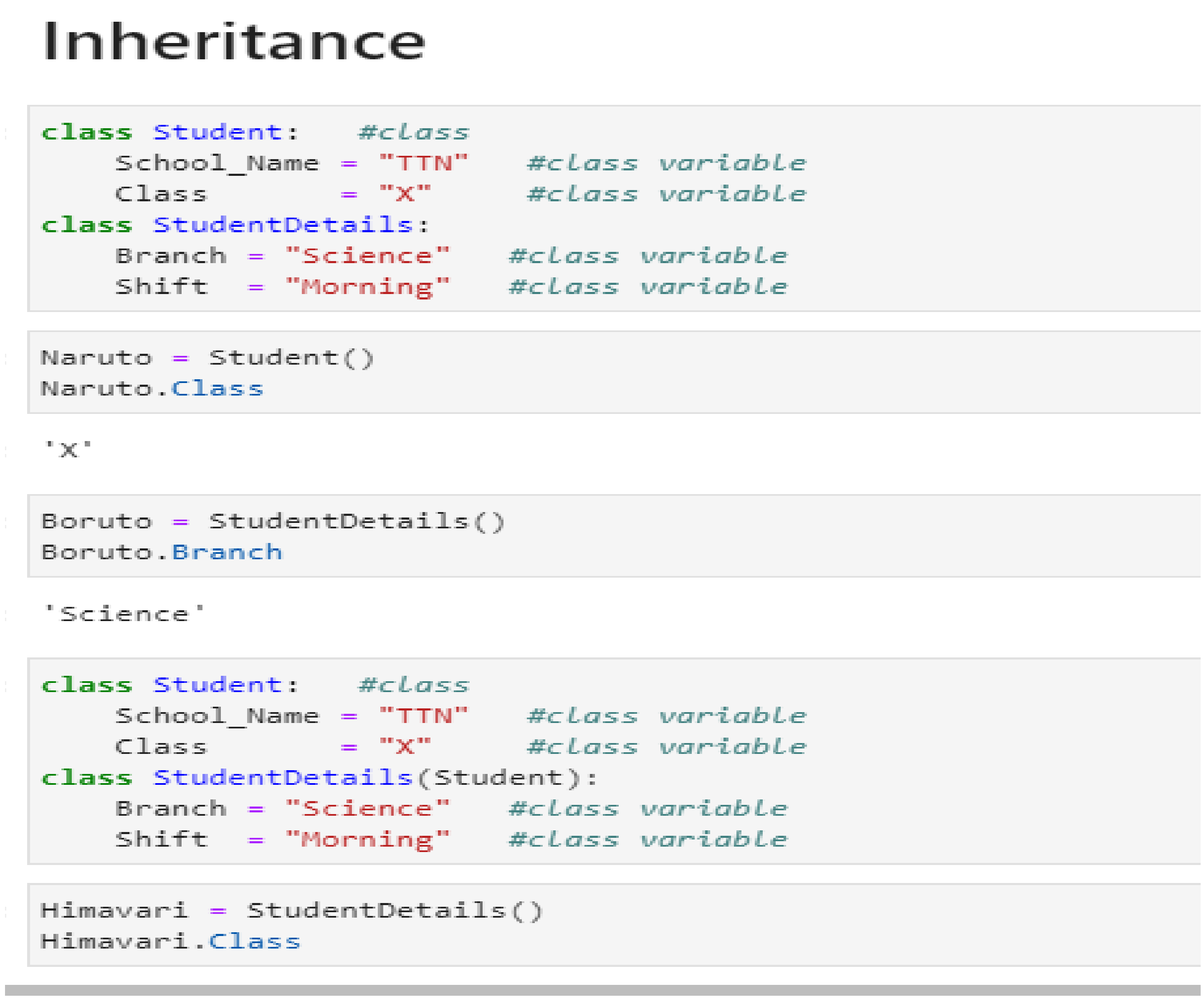
```
#Example 4
class Student4:
    School_Name = "TTN"    #class variable
    Class       = "X"      #class variable
    def __init__(self, x, y):     # special /magic / dunder
        # x,y is localvariable
        self.House     = x       #class Variable
        self.Section   = y       #class Variable
    def get_address(self, z):   #class method
        self.address = z
        return self.address

Boruto = Student4("Konoha", "Team7")
print(Boruto.House, Boruto.Section)
print(Boruto.School_Name, Boruto.Class)
print(Boruto.get_address("village of fire"))
```

```
Konoha Team7
TTN X
village of fire
```

Steps for execution

When both init and class method are give and then first init is executed then rest other.

For calling the variables of the class method , first write the object name and then dot and then we write class method then execute it and after it the write the object name class variables will be seen.



Inheritance

```
class Student:    #class
    School_Name = "TTN"    #class variable
    Class       = "X"      #class variable
class StudentDetails:
    Branch = "Science"    #class variable
    Shift  = "Morning"    #class variable
```

```
Naruto = Student()
Naruto.Class
```

```
'X'
```

```
Boruto = StudentDetails()
Boruto.Branch
```

```
'Science'
```

```
class Student:    #class
    School_Name = "TTN"    #class variable
    Class       = "X"      #class variable
class StudentDetails(Student):
    Branch = "Science"    #class variable
    Shift  = "Morning"    #class variable
```

```
Himavari = StudentDetails()
Himavari.Class
```

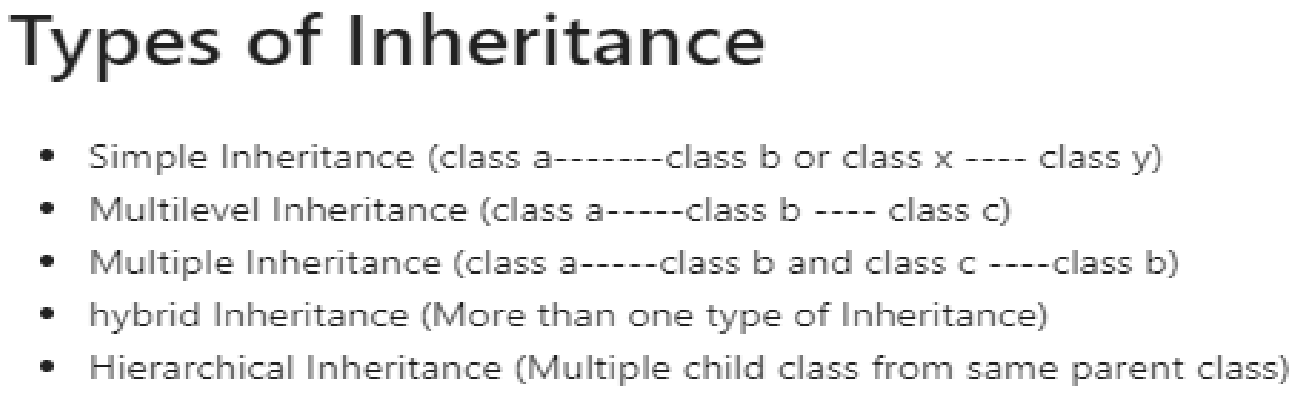
We should all know that the class which is inherited by some other class is parent class and the class which is inheriting any other class is child class. Here Student is parent class and StudentDetails is child class.

First create class Student and then class variable. Then create class StudentDetails and create class variables.

Now create object as Naruto = Student( )

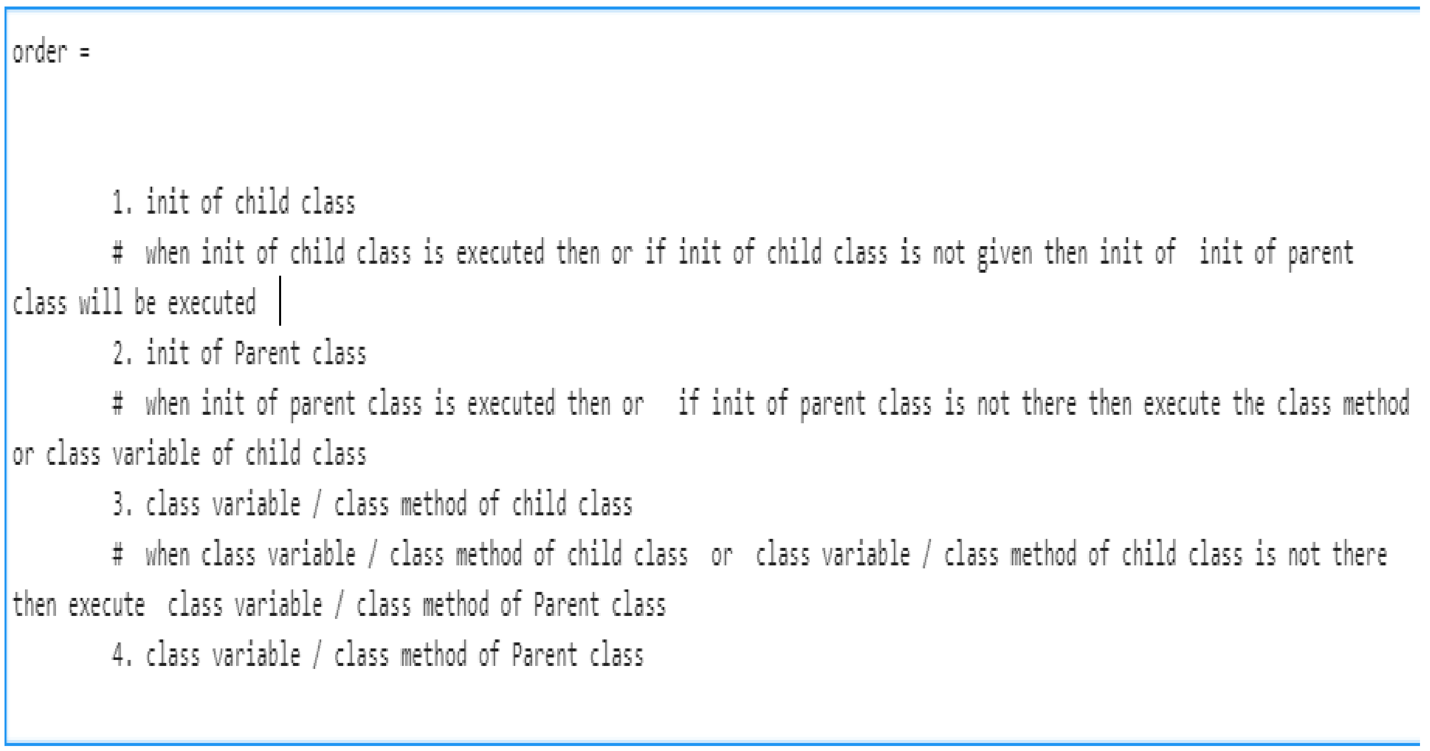
Naruto will have the properties of class Student( ) but not the class StudentDetails ( ) . So to have the property of class StudentDetails( ) write as StudentDetails(Student).

Now as usual write the object name and then period and then choose the variable name.



Types of Inheritance

- Simple Inheritance (class a-------class b or class x ---- class y)
- Multilevel Inheritance (class a-----class b ---- class c)
- Multiple Inheritance (class a-----class b and class c ----class b)
- hybrid Inheritance (More than one type of Inheritance)
- Hierarchical Inheritance (Multiple child class from same parent class)

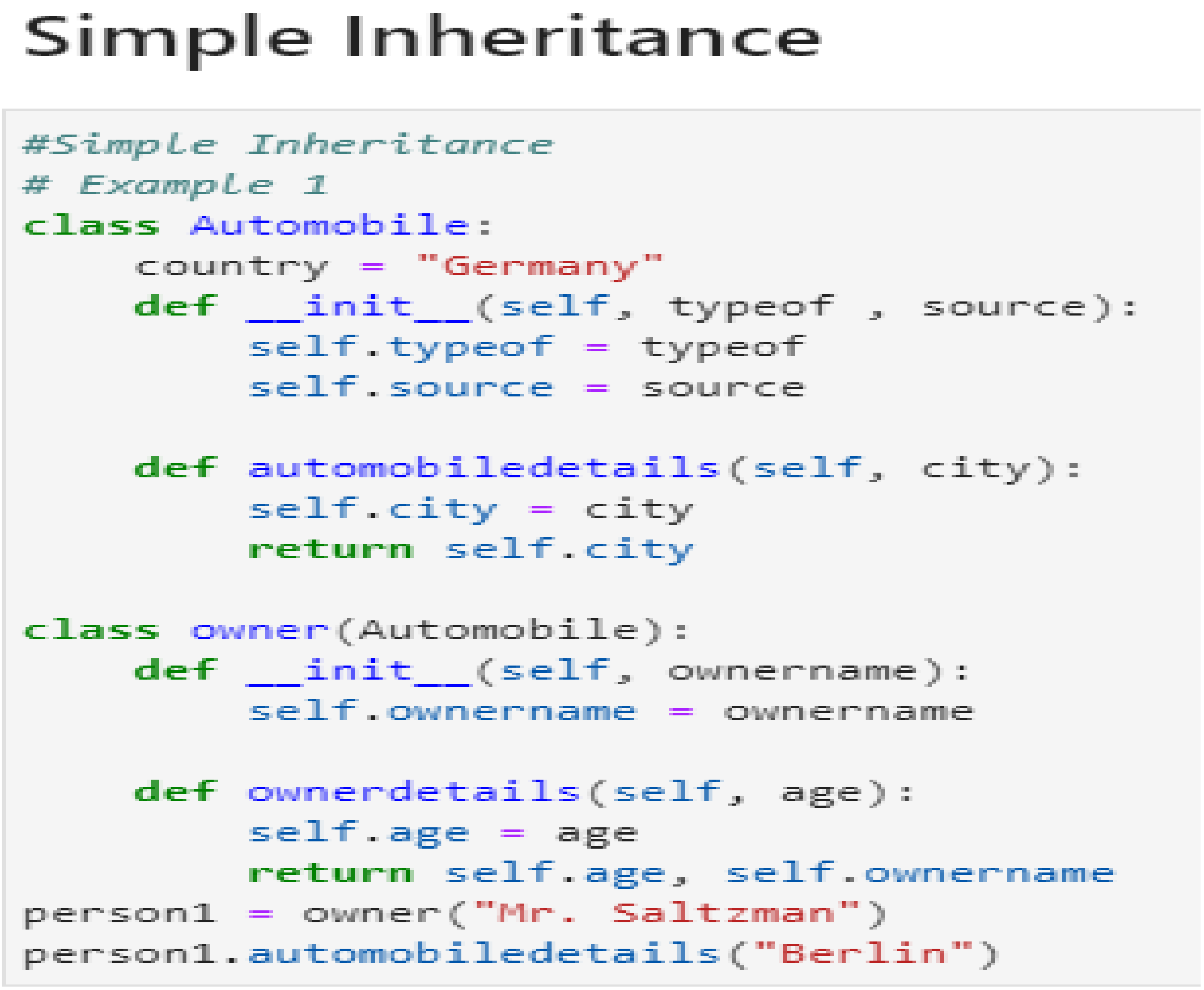


```
order =



        1. init of child class
        #  when init of child class is executed then or if init of child class is not given then init of  init of parent
class will be executed
        2. init of Parent class
        #  when init of parent class is executed then or   if init of parent class is not there then execute the class method
or class variable of child class
        3. class variable / class method of child class
        #  when class variable / class method of child class  or  class variable / class method of child class is not there
then execute  class variable / class method of Parent class
        4. class variable / class method of Parent class
```

----

1 … SIMPLE INHERITANCE

In this type of inheritance there is only two class we can make and the first class which is the parent class is inherited by the second class which is the child class.



Simple Inheritance

```
#Simple Inheritance
# Example 1
class Automobile:
    country = "Germany"
    def __init__(self, typeof , source):
        self.typeof = typeof
        self.source = source

    def automobiledetails(self, city):
        self.city = city
        return self.city

class owner(Automobile):
    def __init__(self, ownername):
        self.ownername = ownername

    def ownerdetails(self, age):
        self.age = age
        return self.age, self.ownername
person1 = owner("Mr. Saltzman")
person1.automobiledetails("Berlin")
```

Make an object from the child class, then within the parenthesis ( ) of child class write the parent class and then as usual . We will not get the init parent class variables of as we have to use super method for it.

----

2 ..

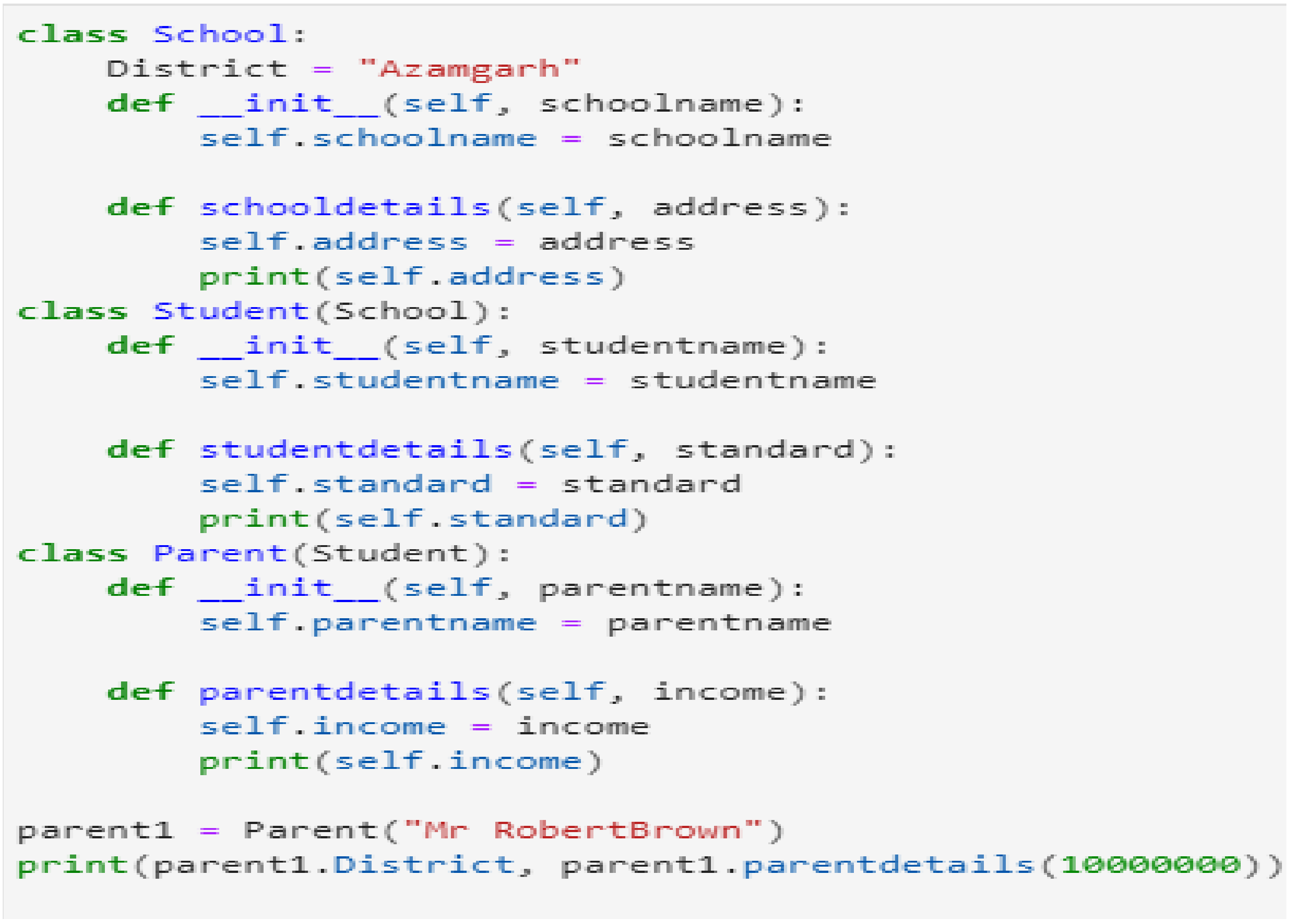
MULTILEVEL INHERITANCE

In this type of inheritance we create parent class first then we create child class which contains the properties of their parent class and then we create another child class which contains properties of previous child and parent class.

First we create parent class whose properties are inherited within child class. For that we have to write as child(parent) and then next child class will inherit the previous child(parent)   class properties as   child(child(parent)).

This is multilevel inheritance.

Multilevel Inheritance



```python
class School:
    District = "Azamgarh"
    def __init__(self, schoolname):
        self.schoolname = schoolname

    def schooldetails(self, address):
        self.address = address
        print(self.address)
class Student(School):
    def __init__(self, studentname):
        self.studentname = studentname

    def studentdetails(self, standard):
        self.standard = standard
        print(self.standard)
class Parent(Student):
    def __init__(self, parentname):
        self.parentname = parentname

    def parentdetails(self, income):
        self.income = income
        print(self.income)

parent1 = Parent("Mr RobertBrown")
print(parent1.District, parent1.parentdetails(10000000))
```

----

3 …

MULTIPLE INHERITANCE

When we have to secure our data and prevent it from knowing to any other except only one who can be any but only one , than for such type of inheritance we have to use multiple inheritance.

In this type of inheritance we will create a class 1 and then another class 2 and then another class 3 as

class3(class1,class2) , so that, these class3(class1,class2) will contain properties of class1 and class 2 both and it will prevent both class 1 and class 2 from knowing each of their own properties.

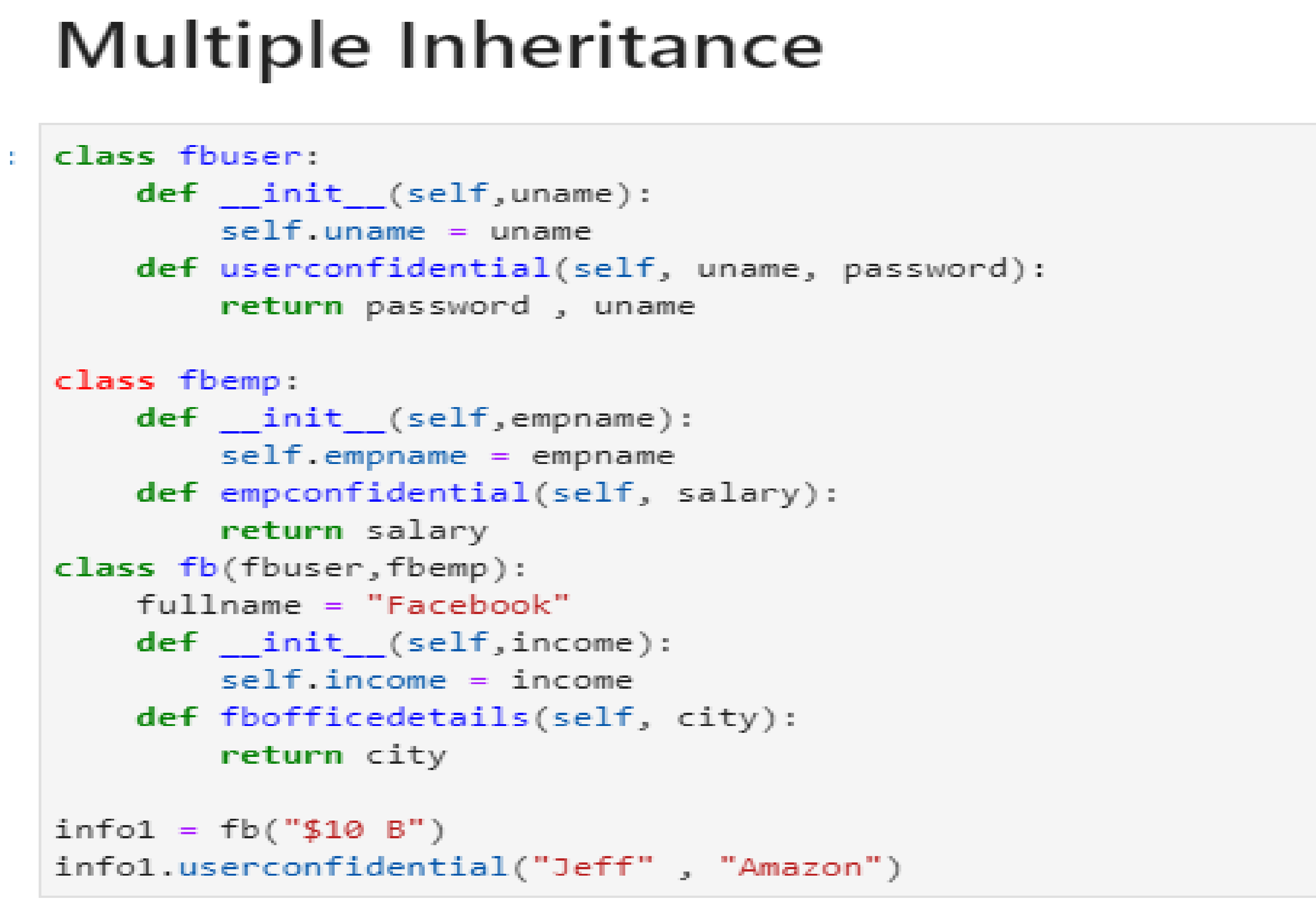
Here remember we have to write first class 1 (as an example: fbuser) , then as usual its details.

Then we have to write class 2 (as an example:fbemp) , then its details .

Then write class3(class1,class2) (as an example: fb(fbuser,fbemp)).

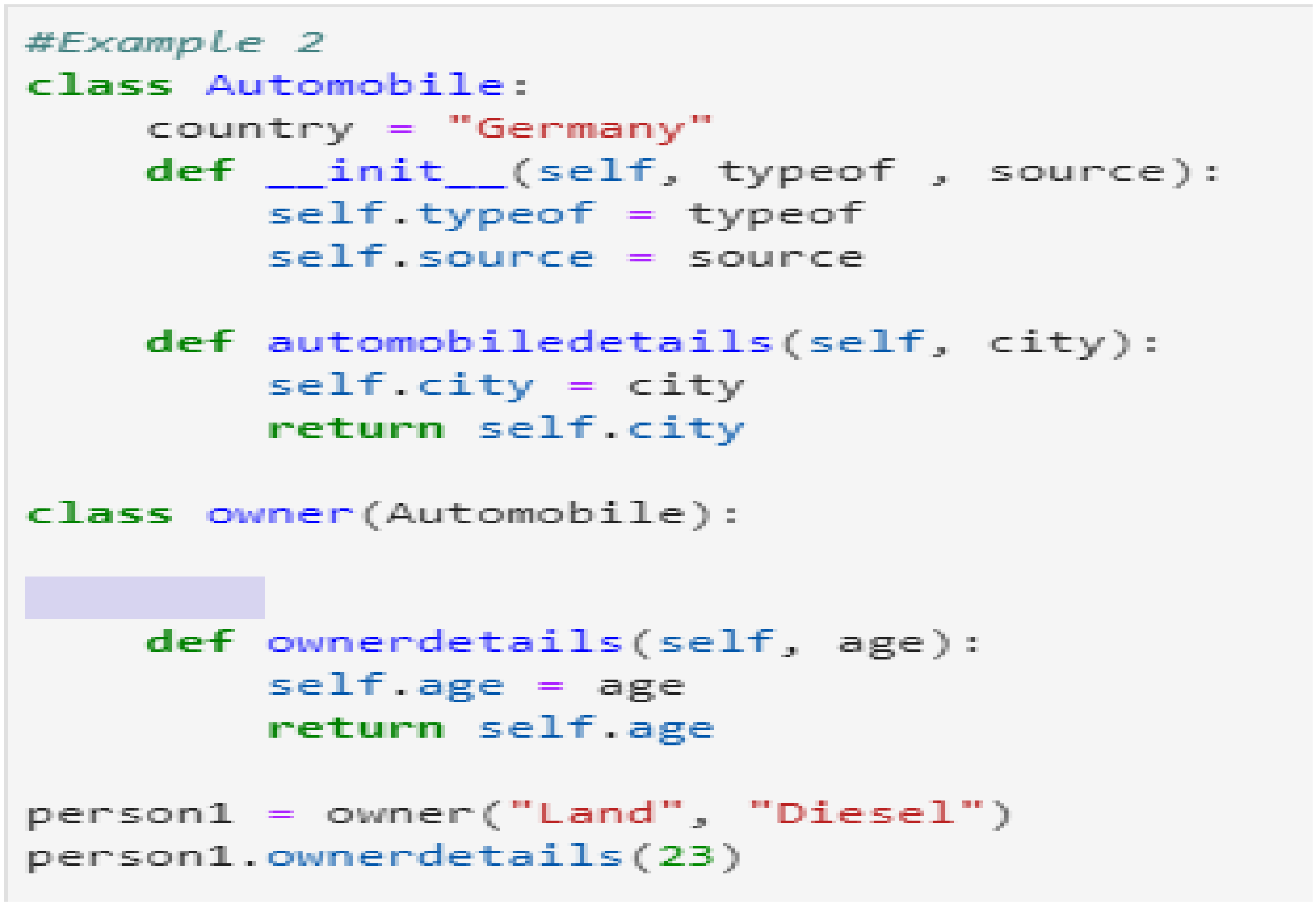
The upper line will contain both the properties of class 1 and class 2.

And also the information of class 1 and class 2 is not shared between each other .It is only known by the class3 .



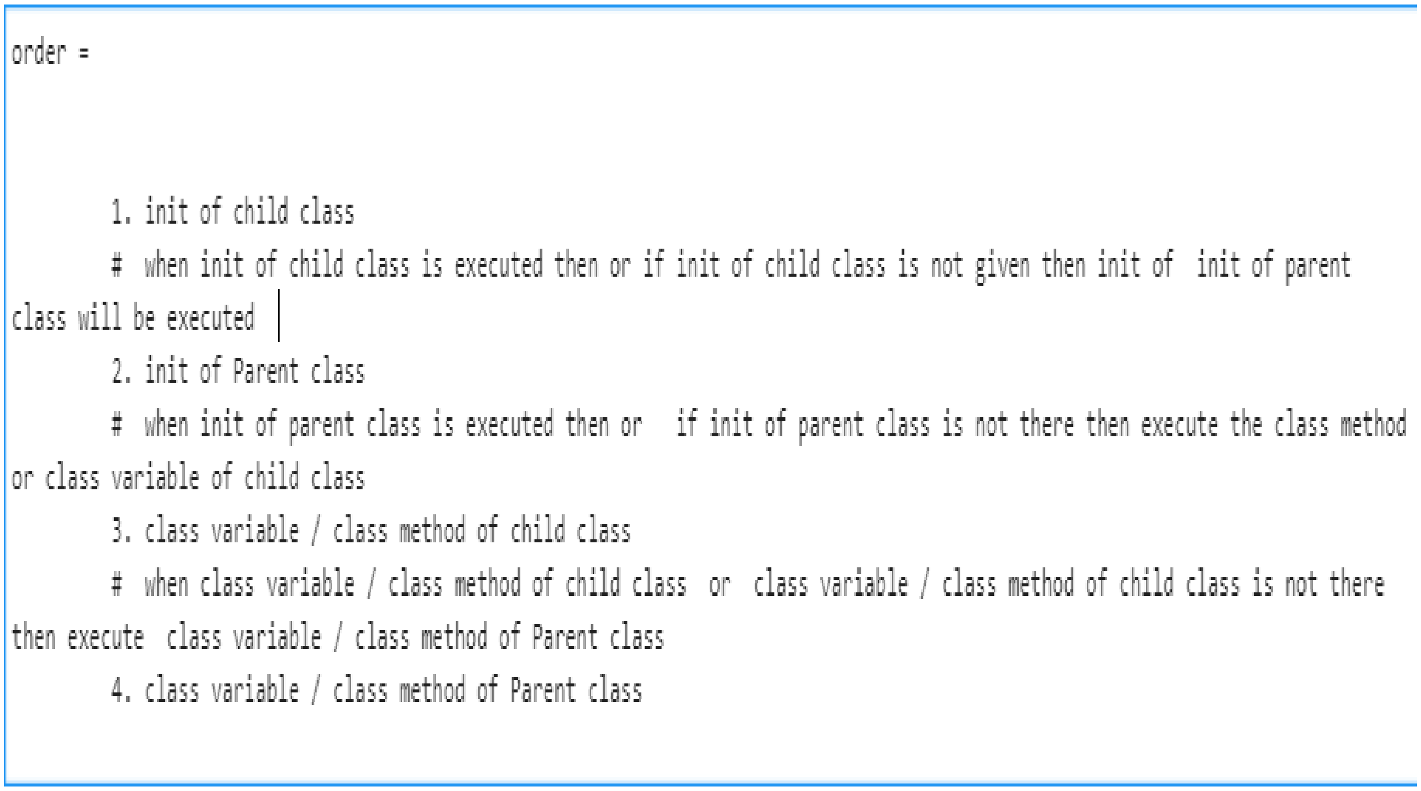
Multiple Inheritance

```python
class fbuser:
    def __init__(self,uname):
        self.uname = uname
    def userconfidential(self, uname, password):
        return password , uname

class fbemp:
    def __init__(self,empname):
        self.empname = empname
    def empconfidential(self, salary):
        return salary
class fb(fbuser,fbemp):
    fullname = "Facebook"
    def __init__(self,income):
        self.income = income
    def fbofficedetails(self, city):
        return city

info1 = fb("$10 B")
info1.userconfidential("Jeff" , "Amazon")
```

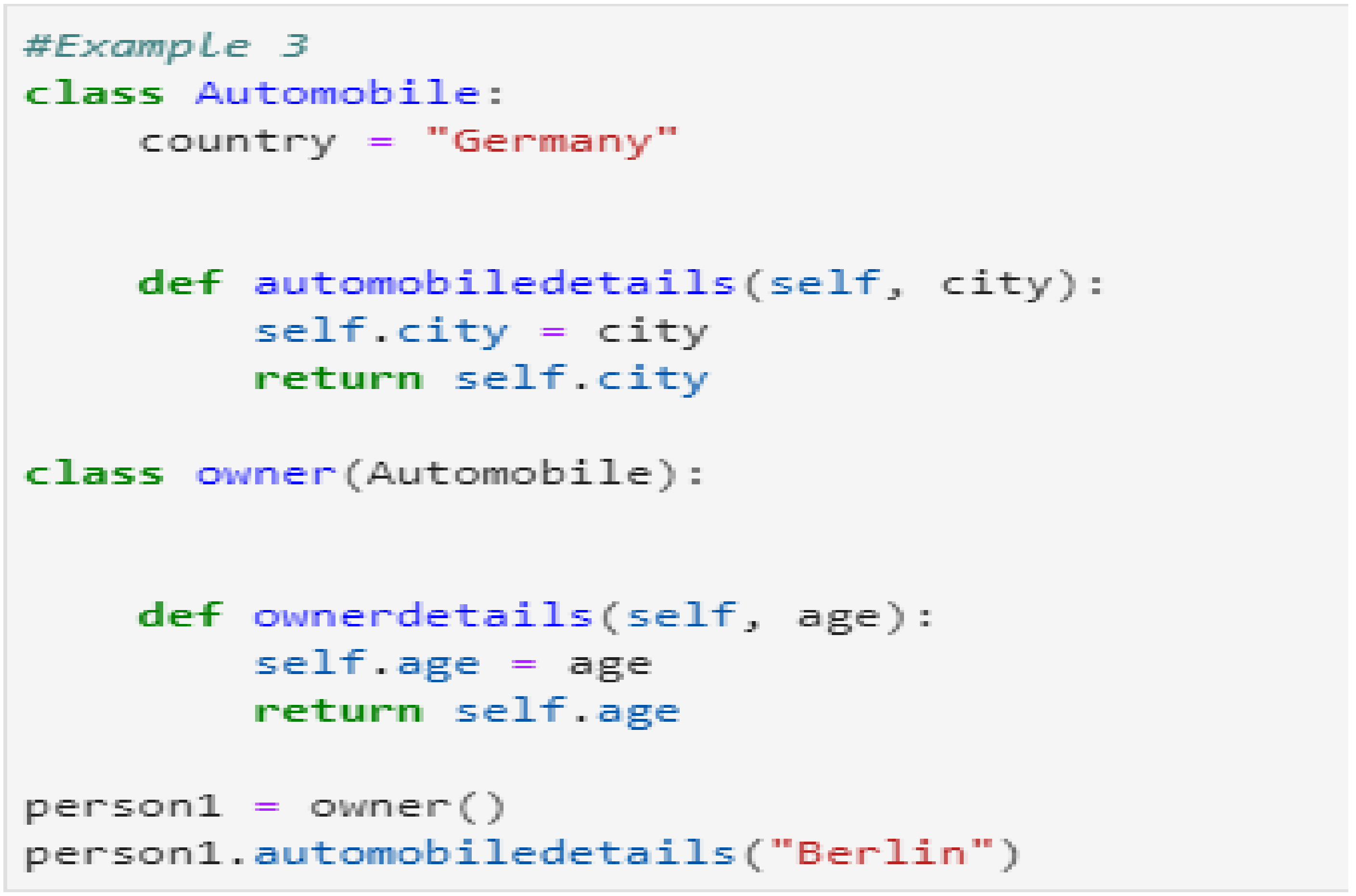



```
#Example 2
class Automobile:
    country = "Germany"
    def __init__(self, typeof , source):
        self.typeof = typeof
        self.source = source

    def automobiledetails(self, city):
        self.city = city
        return self.city

class owner(Automobile):


    def ownerdetails(self, age):
        self.age = age
        return self.age

person1 = owner("Land", "Diesel")
person1.ownerdetails(23)
```

Here in example 2 there is no init method in child class so the steps of execution will be different.



```
order =



        1. init of child class
        #  when init of child class is executed then or if init of child class is not given then init of  init of parent
class will be executed
        2. init of Parent class
        #  when init of parent class is executed then or   if init of parent class is not there then execute the class method
or class variable of child class
        3. class variable / class method of child class
        #  when class variable / class method of child class  or  class variable / class method of child class is not there
then execute  class variable / class method of Parent class
        4. class variable / class method of Parent class
```



```
#Example 3
class Automobile:
    country = "Germany"


    def automobiledetails(self, city):
        self.city = city
        return self.city

class owner(Automobile):


    def ownerdetails(self, age):
        self.age = age
        return self.age

person1 = owner()
person1.automobiledetails("Berlin")
```

Here both the init class method are not here so steps for execution will be changed .

Init of child class is not in example 3 so it will look for the init of parent class.

Init of parent class is also not present so it will look for the class variable or the class method of child class. It is present here .

So it class variable or the class method of child class wil be executed and after it class variable or the class method of parent class.

4.

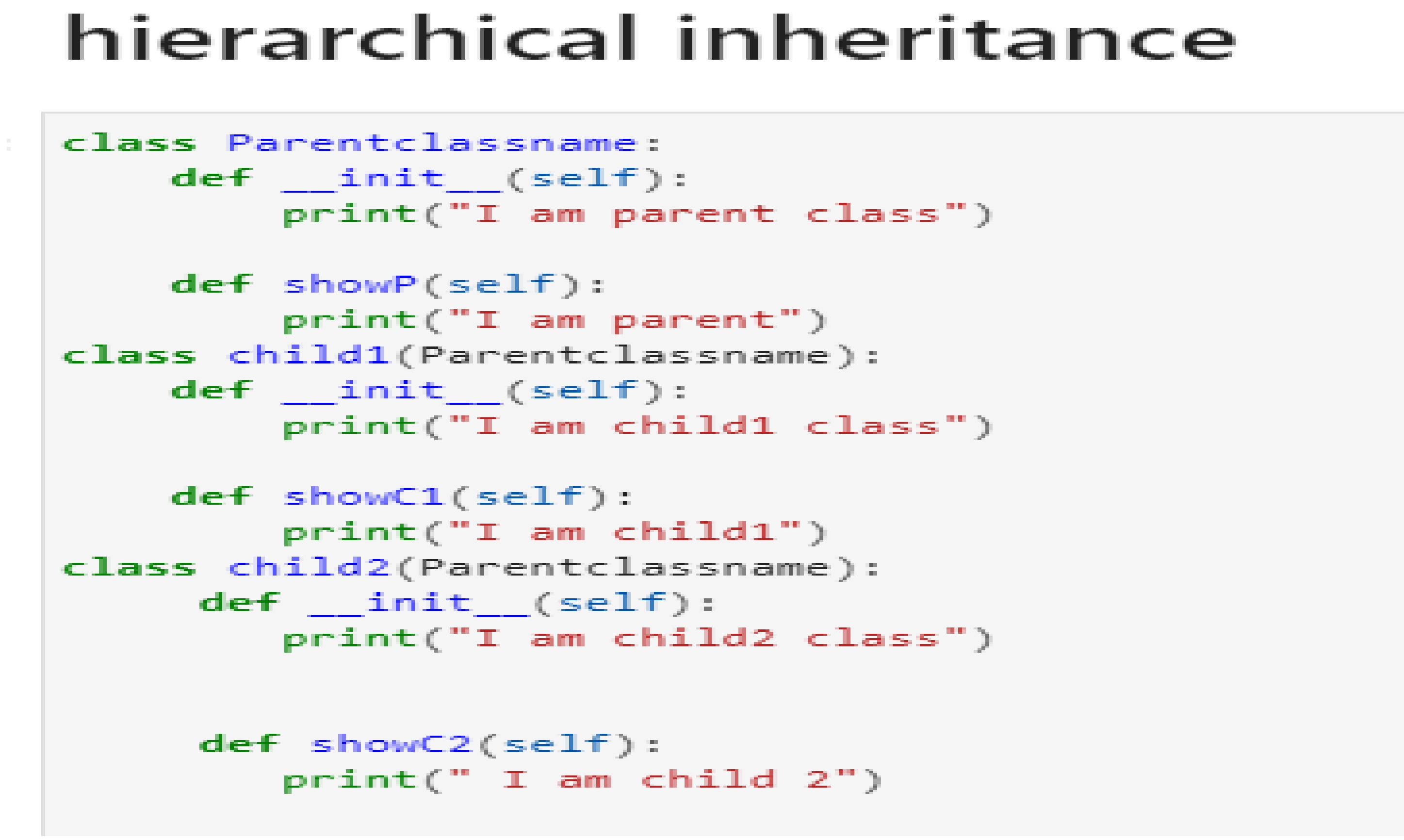
HIERARCHICAL INHERITANCE

It is the inheritance where there is two child from same parent.

First create class parent(we can give any name to parent class) . Then create child 1 class (we can give any name to child1 1 class).

Then create child 2 class (we can give any name to child1 2 class) and in both the parenthesis( ) of child 1 class and child 2 class write parent class name.

It is also helpful in preventing the data of both child to be shared between each other and only let the parent or 1 people to know the all data.



hierarchical inheritance

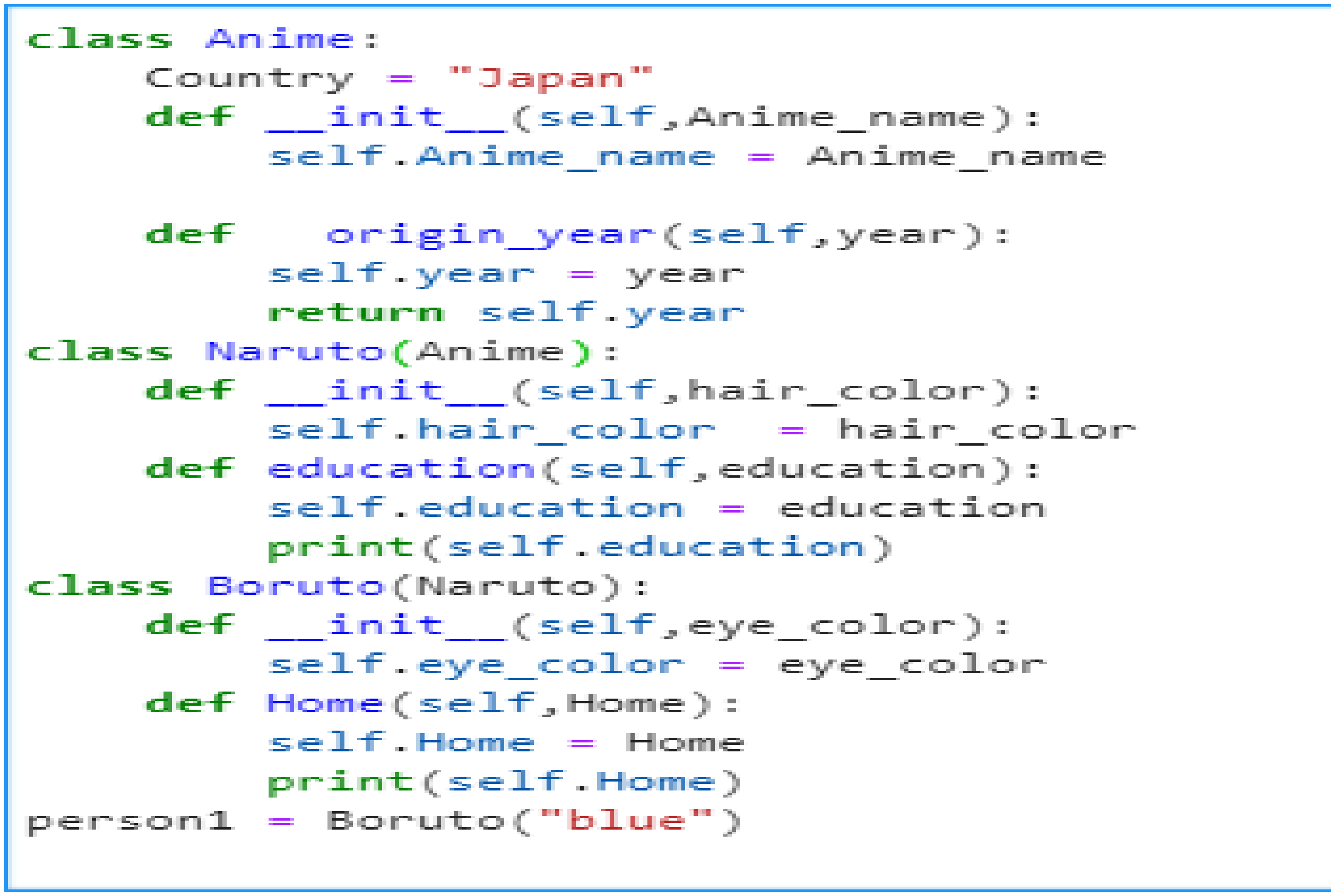
```
class Parentclassname:
    def __init__(self):
        print("I am parent class")

    def showP(self):
        print("I am parent")
class child1(Parentclassname):
    def __init__(self):
        print("I am child1 class")

    def showC1(self):
        print("I am child1")
class child2(Parentclassname):
     def __init__(self):
        print("I am child2 class")


     def showC2(self):
        print(" I am child 2")
```

----

5 ..

HYBRID INHERITANCE

More than one type inheritance like

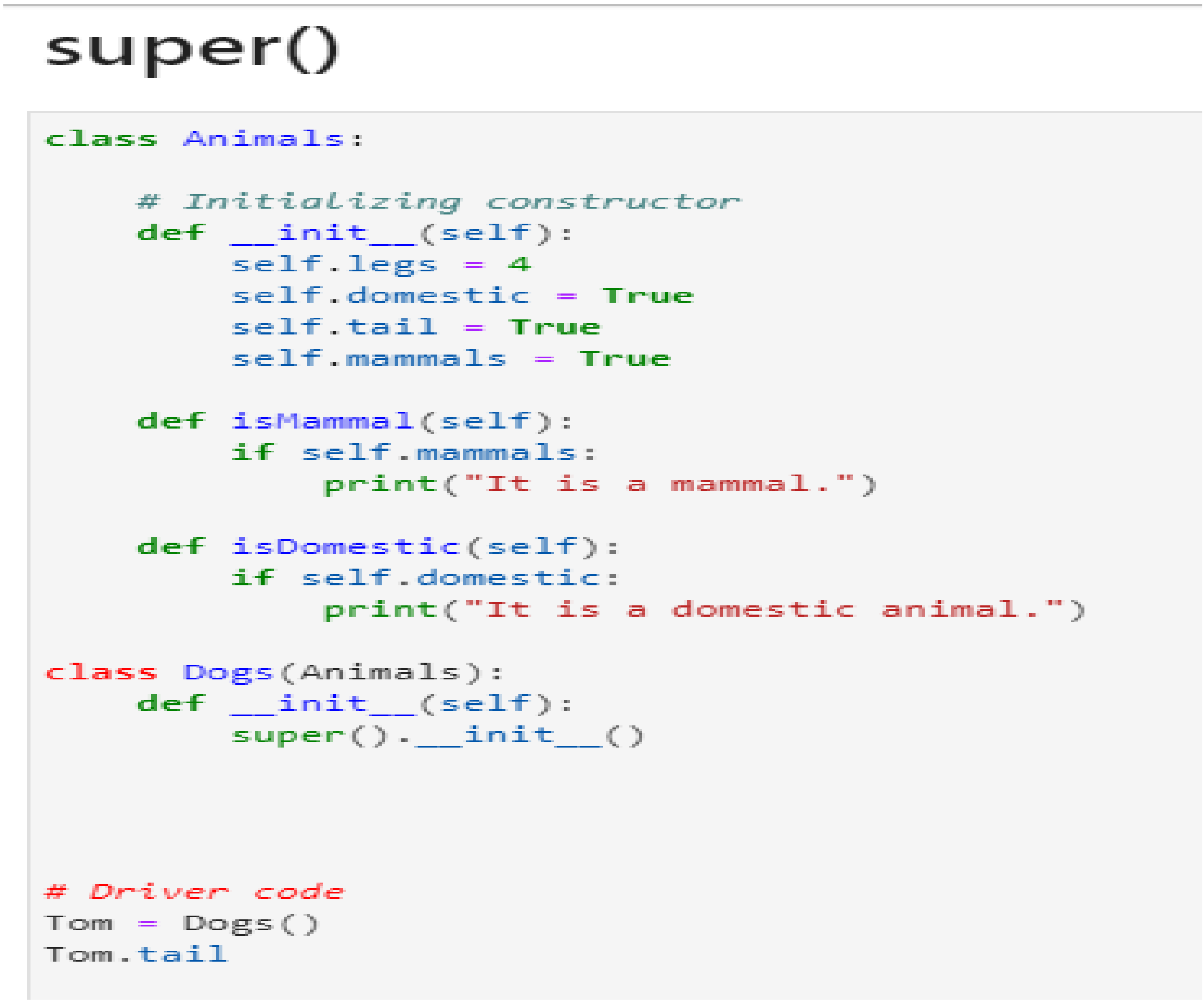


```
class Anime:
    Country = "Japan"
    def __init__(self,Anime_name):
        self.Anime_name = Anime_name

    def   origin_year(self,year):
        self.year = year
        return self.year
class Naruto(Anime):
    def __init__(self,hair_color):
        self.hair_color  = hair_color
    def education(self,education):
        self.education = education
        print(self.education)
class Boruto(Naruto):
    def __init__(self,eye_color):
        self.eye_color = eye_color
    def Home(self,Home):
        self.Home = Home
        print(self.Home)
person1 = Boruto("blue")
```

If we consider class Anime and class Naruto they are simple inheritance ,but of we look at class Anime,Naruto, Boruto , then they are multilevel inheritance.

So the whole could be considered as Hybrid inheritance.

SUPER



super()

```
class Animals:

     # Initializing constructor
     def __init__(self):
         self.legs = 4
         self.domestic = True
         self.tail = True
         self.mammals = True

     def isMammal(self):
         if self.mammals:
             print("It is a mammal.")

     def isDomestic(self):
         if self.domestic:
             print("It is a domestic animal.")

class Dogs(Animals):
     def __init__(self):
         super().__init__()



# Driver code
Tom = Dogs()
Tom.tail
```

We have seen that when we write parent class and then when we write child class both having init function we can't get the values of those class variables that are inside the init function (when we create the object of the class variables and when we write period, then don't see the class variables inside the init function)

So to get those class variables values that are inside the init function we will use SUPER function.

STEPS ..

First write the parent class (as in above example ,it is Animals),then write class variables or if there is no class variables then write init function and its class variables .

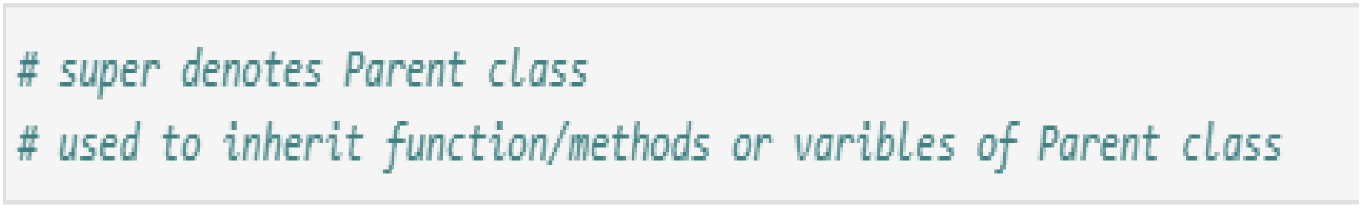
And then after write a class method and its details.

Then after it write a new class.

Inherit the parent class in new class(child class, here the child class is Dogs, Parent class is Animals.)

Write the init function of child class and after it write super.( )__init__( )

Just upper line will call the init class variables of of parent class and then when we will create object of child class we will see the class variables of parent class.



```
# super denotes Parent class
# used to inherit function/methods or varibles of Parent class
```

Overriding Concept:

What is overriding concept ?

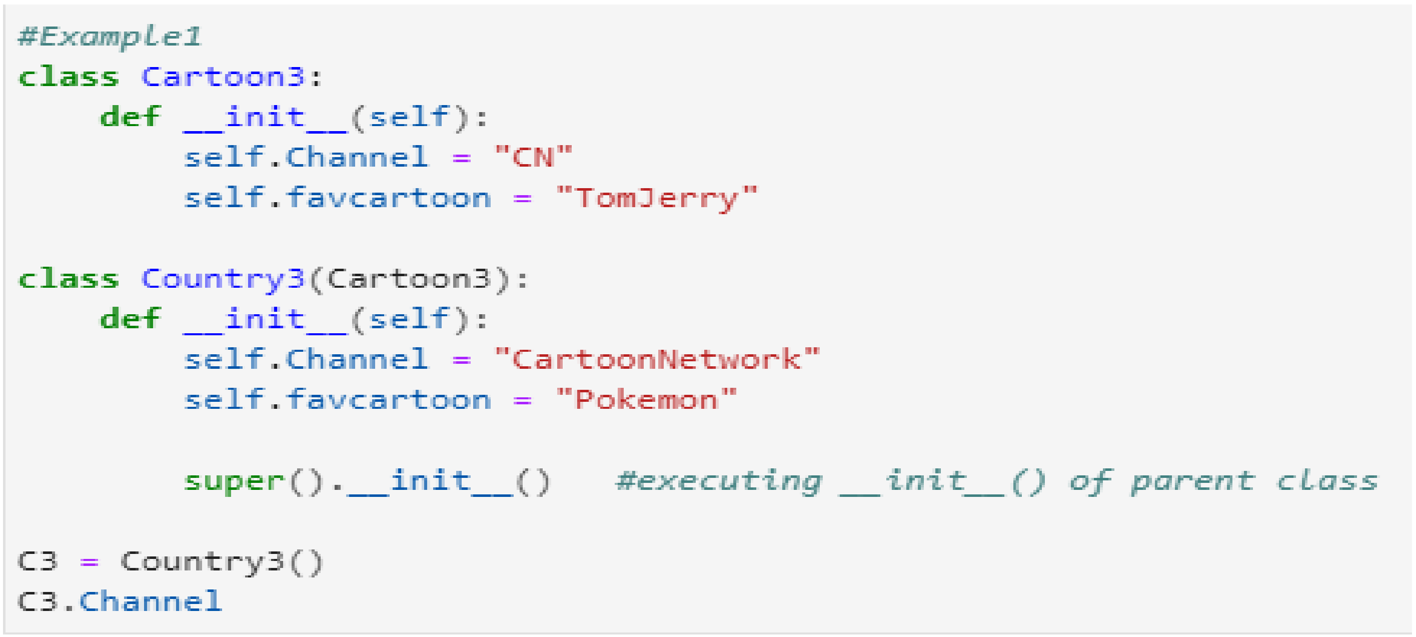
Overriding means first we give some value to a variable (object1) and just after it again create an object(object1) with same name just as previous object(object1) name.

Write a new value(value 2) but don't change the object name(object1) so that after we have have changed the value(value2) of object(object1) ,the object(object1) name remains same.

So that when we execute it first the first object(object1) name is executed and its value (value1)and then suddenly the after it the same object (object1) name with different value (value 2)is executed .

Then after we we print the (object1) ,we will not get the value as value 1 instead we will get value 2 .

Which shows the value 1 has been overridden as value 2…… This is overriding concept.



```python
#Example1
class Cartoon3:
    def __init__(self):
        self.Channel = "CN"
        self.favcartoon = "TomJerry"

class Country3(Cartoon3):
    def __init__(self):
        self.Channel = "CartoonNetwork"
        self.favcartoon = "Pokemon"

        super().__init__()   #executing __init__() of parent class

C3 = Country3()
C3.Channel
```

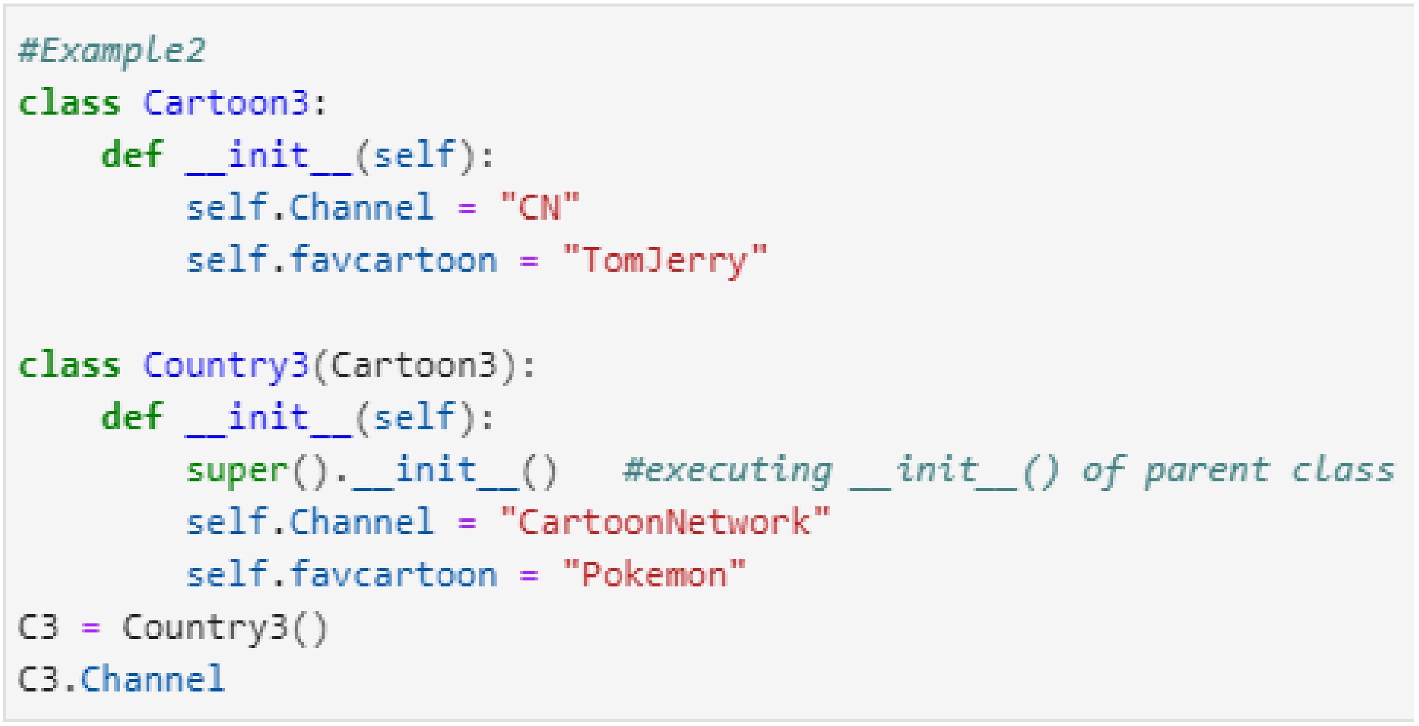
When super().__init__() at the bottom.

First write the parent class(here Cartoon3 as an example) and then write init function and the write class variables and its values.

Then write child class (here Country3 as an example) and then inherit parent class within child class as (Country3(Cartoon3)).Write init function and then create class variables with same of object as in the parent class but with different values.

Now we will write super().__init__ so that when we will create object of child class , period, press tab key and we will see class variables ,then we will click on class objects and we will see the outcomes as the result of parent class not child class because child class values had been overridden by parent class values.

When super().__init__() at the top.



```
#Example2
class Cartoon3:
    def __init__(self):
        self.Channel = "CN"
        self.favcartoon = "TomJerry"

class Country3(Cartoon3):
    def __init__(self):
        super().__init__()   #executing __init__() of parent class
        self.Channel = "CartoonNetwork"
        self.favcartoon = "Pokemon"
C3 = Country3()
C3.Channel
```

Here in the above example

Create parent class and inside it create init and its class variables.

After creating child class first write init function and after this write super ( ).__init__( ) .

So super will call the class variables of parent class ,now the class variables is having the value of parent class.

After this write the class variables and give its values.

But here we have wrote class variables(object) having same name as in parent class.

Now when we will create object from child class and execute it whole.

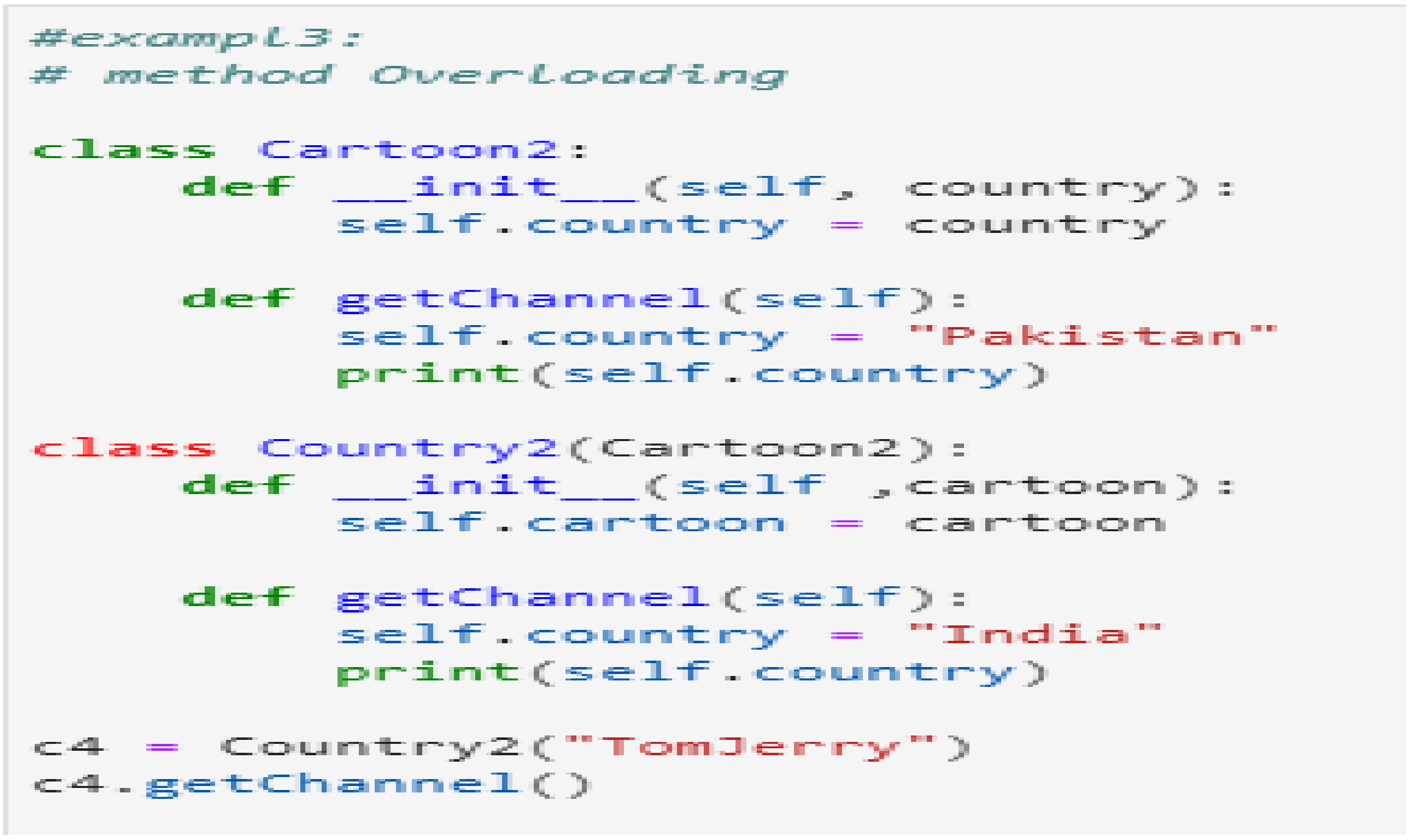
Now write object , period , press tab key and then we will see class object name and click on one of those variables name.

We will get the values of child class not the class variables values of parent class.

This happens because child class before class variables of child class we wrote **super** function.

This is overriding concept.

METHOD OVERLOADING



```
#exampl3:
# method Overloading

class Cartoon2:
        def __init__(self, country):
            self.country = country

        def getChannel(self):
            self.country = "Pakistan"
            print(self.country)

class Country2(Cartoon2):
        def __init__(self ,cartoon):
            self.cartoon = cartoon

        def getChannel(self):
            self.country = "India"
            print(self.country)

c4 = Country2("TomJerry")
c4.getChannel()
```

The steps of execution in method overloading are –

first the child class.

Then the init of child class

Then the class function of child class

And similarly for parent class

first the parent class.

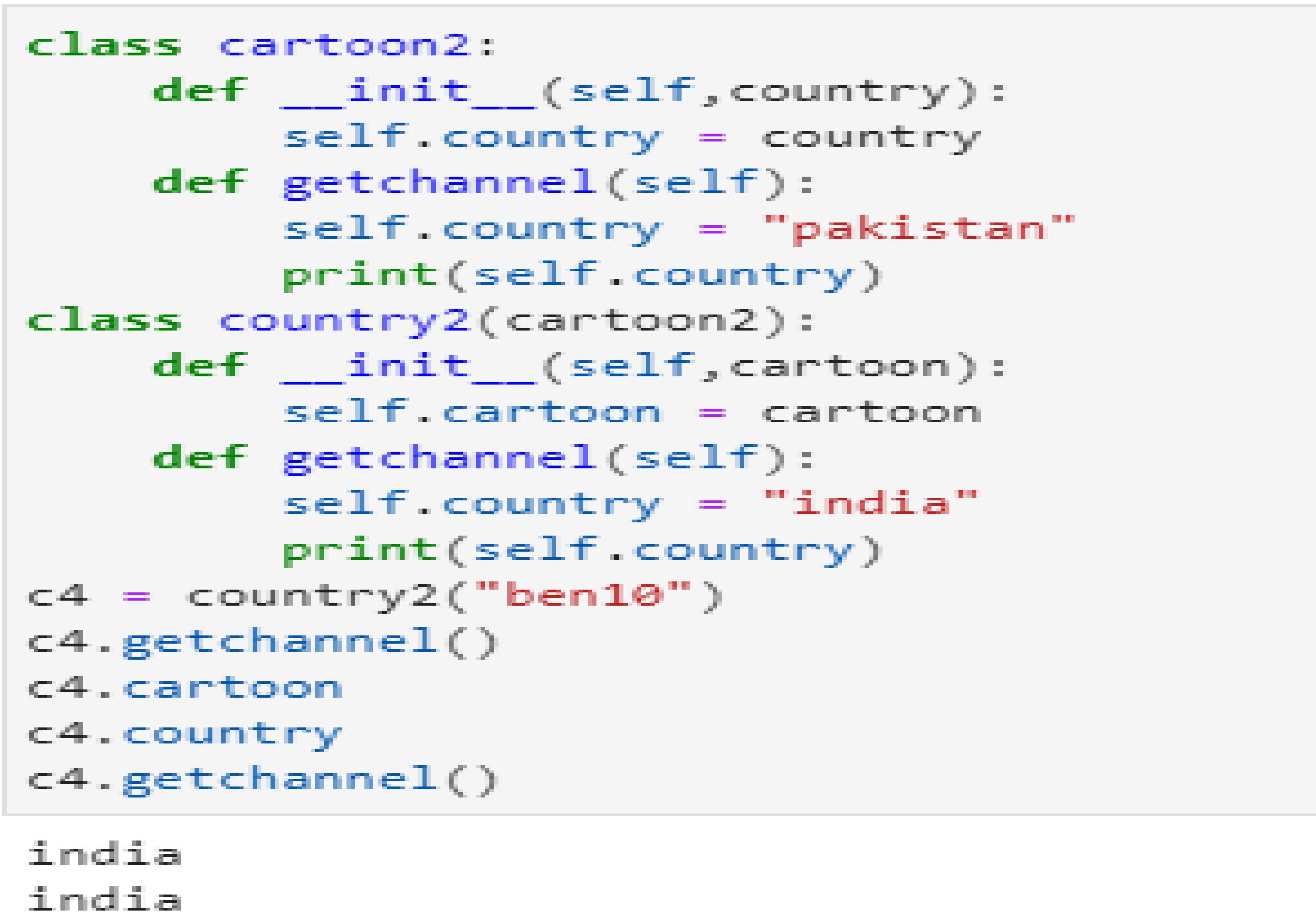
Then the init of parent class

Then the class function of parent class

Remember that when you create object from child (inheriting parent class) class , the result of the class variable which you get after writing object ,period , class variable name will be of the first place where it find itself in the coding in process of execution..

Meanwhile in example 3 on creating object from the child class Country2 and executing it and then after writing c4.getchannel() which is the class variable from class function of child class we will see the outcome as the “India” not “Pakistan” .

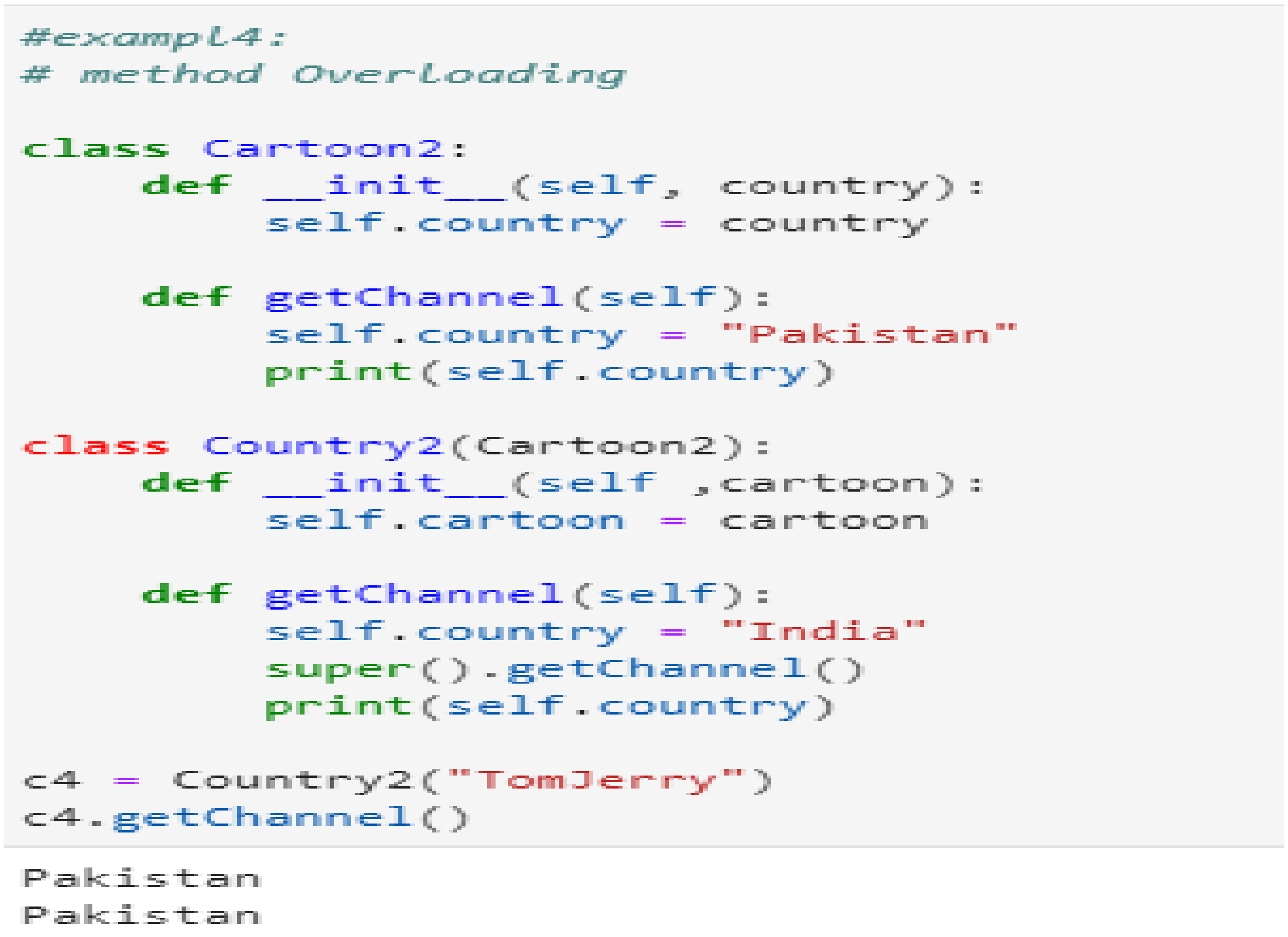
 And this happens because of the steps for execution in method overloading.



```
class cartoon2:
    def __init__(self,country):
        self.country = country
    def getchannel(self):
        self.country = "pakistan"
        print(self.country)
class country2(cartoon2):
    def __init__(self,cartoon):
        self.cartoon = cartoon
    def getchannel(self):
        self.country = "india"
        print(self.country)
c4 = country2("ben10")
c4.getchannel()
c4.cartoon
c4.country
c4.getchannel()
```

```
india
india
```

In the above image we haven’t written super function anywhere ,so it will first go to the child class and it will show the result of child class variables there.



```
#exampl4:
# method Overloading

class Cartoon2:
    def __init__(self, country):
        self.country = country

    def getChannel(self):
        self.country = "Pakistan"
        print(self.country)

class Country2(Cartoon2):
    def __init__(self ,cartoon):
        self.cartoon = cartoon

    def getChannel(self):
        self.country = "India"
        super().getChannel()
        print(self.country)

c4 = Country2("TomJerry")
c4.getChannel()
```

```
Pakistan
Pakistan
```

In the example 4 first it enters in child class , from there init of child class ,after that getChannel.

In class method of child class(getChannel)   , the first class variable i.e, self.country will be executed but suddenly after it super() is there which sends the execution step to class method getChannel of parent class.

Then it will execute the same class variable of parent class which was in child class and it will give the result of what is written inside the print function(i.e, self.country =”Pakistan” , print(self.country)).

Now as it have printed result of parent class which was possible only due to the super() function of child class in child class class function.

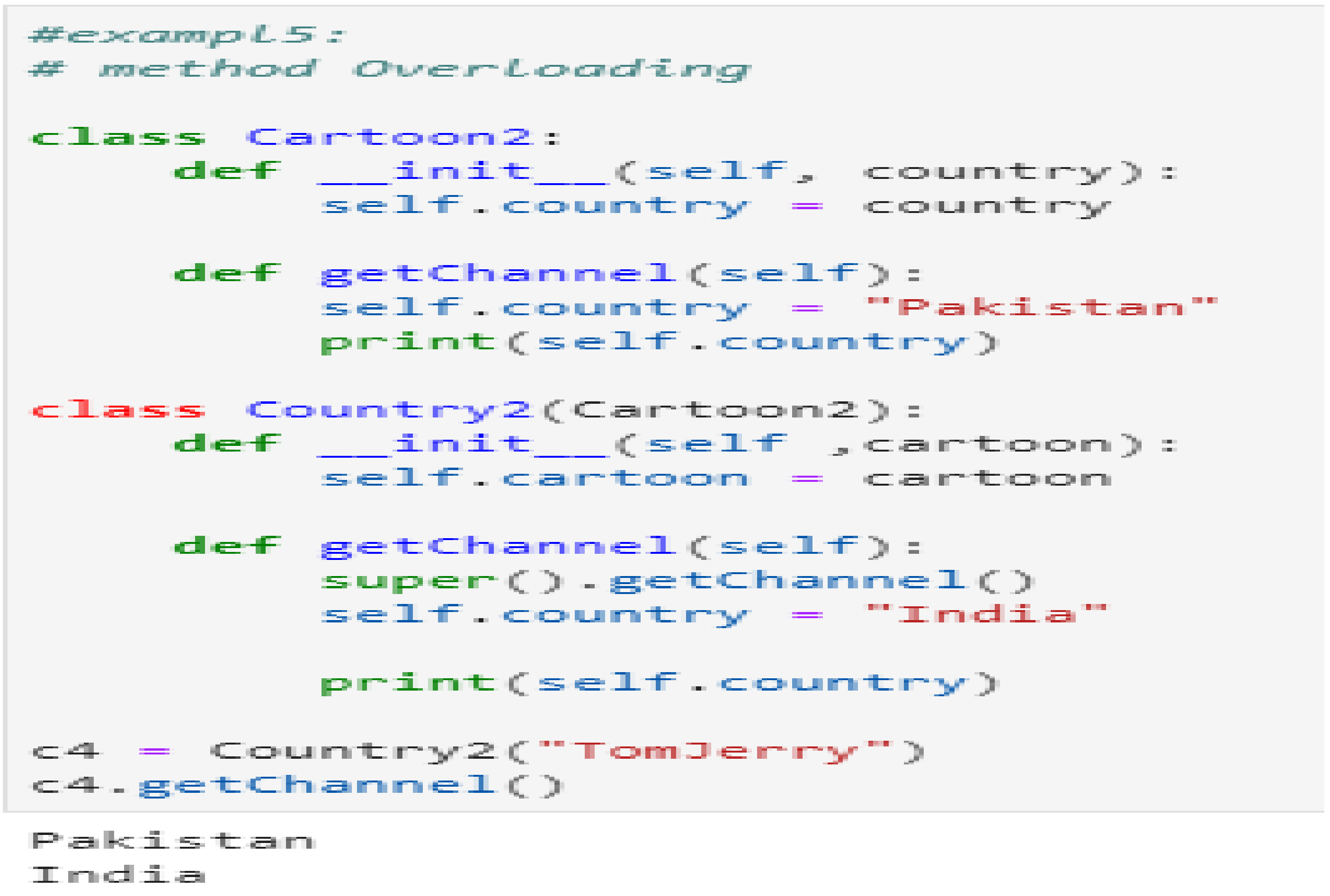
After that, now,it has executed super function in child class it will now execute further written function but if if the class variables(objects) name are same even if the values are different then it will overwrite the objects value by overriding concept.

And after that it will execute the further functions and will give results ,which is print(self.Country).

So that’s why we got two times as------- Pakistan

Pakistan

First Pakistan is of   parent class getChannel when super of child class was executed and then it was overridden and second was from print(self.country) of child class.



```
#exampl5:
# method Overloading

class Cartoon2:
    def __init__(self, country):
        self.country = country

    def getChannel(self):
        self.country = "Pakistan"
        print(self.country)

class Country2(Cartoon2):
    def __init__(self ,cartoon):
        self.cartoon = cartoon

    def getChannel(self):
        super().getChannel()
        self.country = "India"

        print(self.country)

c4 = Country2("TomJerry")
c4.getChannel()
```

```
Pakistan
India
```

Here in above example ,it is just similar to example 4,the only difference is that we have wrote super function a little before in example 5 than in example 4.

So first enter in init of child class then class function of child class i.e, getChannel.

Then after super() came so it will direct the execution steps to class function getChannel of parent class and firstly it will give result of parent class as in getChannel of parent class there is print function so when it executes the print function of parent class it shows Pakistan.

Then again it will come to getChannel of child class .

But now this time it will look after super of child class as it had executed till super of child class so it will execute the result after it.

After that it is written self.country = “India”

And then print(self.country)

So it is showing as---- Pakistan

India

################################ the end
#####################################