Let’s recall that we can mention or initialize variables and methods inside a class.

Now can we nest a class inside a class?

Let’s look at below code, execute and check the output

Graphical user interface, text, application

Description automatically generated

Instead we can print the data with the help of another function as shown below

Graphical user interface, timeline

Description automatically generated

* Now let’s say student also has a laptop and we want to know which laptop the student is using.
* When we speak about laptop we have different things like brand, configuration, ram etc.. and we can ask some details as shown below.

Graphical user interface, application

Description automatically generated

🡪Other option which we have is we can create a separate class or we can create a ***class*** inside the student class.

**-🡪 since we had created the laptop class as a inner class, so now we need to create the object of the laptop and we need to create it inside the outer class(which is student class in this clase) as highlighted below.**

Graphical user interface, text, application

Description automatically generated

Whenever you want to use a lap object we need to call it using s1.lap.brand(the lap object is inside the student class)

Graphical user interface, text, application

Description automatically generated

What if you want to create another object of it. We need to mention as shown below. Print the id’s of the objects and check their namespaces.

Graphical user interface, text, application

Description automatically generated

Remember that we can create a class inside the class and use it. Else we can create a separate class.

* Now let’s look at a separate class which is outside the student class.

A person wearing glasses

Description automatically generated with low confidence

To enhance the example we can create show methods as shown below 1) show for the student 2)for the laptop

Graphical user interface

Description automatically generated with low confidence

Now if we run s1.show() we will get the name and roll number

Graphical user interface, text, application

Description automatically generated

Now if we run s1.show() we will see the name roll no, lap brand, processor, ram

A picture containing graphical user interface

Description automatically generated

**Inheritance:**

Parent child relationship

Graphical user interface, application, Teams

Description automatically generated

Now let’s create another Class with the features as shown below:

Graphical user interface, application, Teams

Description automatically generated

* Now if we create a new object b1 from class B and try to access the methods we get access to only feature 3 and feature 4

Graphical user interface, application

Description automatically generated

Now if we want to access the features from class A for the object b1 which is created from class B.

We need to make the class B as a child class to A by mentioning as ***class B(A):***

By doing so the class B will inherit all the features from class A as shown below. – Now check b1.--- and we can access the features available from class A as well.

Graphical user interface, application, Teams

Description automatically generated

Terminology:

A picture containing text, person, screenshot, smiling

Description automatically generated

The above example is called as single level inheritance. We also have multi level inheritance, example shown below with class C

Graphical user interface, website

Description automatically generated

Something like grandparent, parent, child relationship.

Now B is not inheriting from A which means we will remove A from class B(**A**) which results in class B: and class B is on its own now where as class C is inheriting the features from A and B class C(A,B):

Graphical user interface, website

Description automatically generated

Class C(A,B) type of inheritance is called multiple inheritance

Graphical user interface

Description automatically generated with medium confidence

class A:

    def featureA1(self):

        print('feature A1 is working')

    def featureA2(self):

        print('feature A2 is working')

class B:

    def featureB1(self):

        print('feature B1 is working')

    def featureB2(self):

        print('feature B2 is working')

class C(A,B):

    def featureC(self):

        print('feature C is working')

a1 = A()

a1.featureA1()

a1.featureA2()

b1 = B()

b1.

c1 = C()

class Student():

    def \_\_init\_\_(self, name, rollno):

        self.name = name

        self.rollno = rollno

        self.lap = self.Laptop()

    def show(self):

        print(self.name , self.rollno)

        self.lap.show()

    class Laptop:

        def \_\_init\_\_(self):

            self.brand = 'HP'

            self.cpu = 'i7'

            self.ram = 16

        def show(self):

            print(self.brand, self.cpu, self.ram)

s1 = Student('rishi', 1)

s2 = Student('bunny', 2)

s2.show()

class Student:

    def \_\_init\_\_(self, name, rollno):

        self.name = name

        self.rollno = rollno

        self.lap = self.Laptop()

    def show(self):

        print(self.name, self.rollno)

    class Laptop:

        def \_\_init\_\_(self):

            self.brand = 'HP'

            self.config = 'i7'

            self.ram = 16

        def show(self):

            print(self.brand, self.config, self.ram)

student1 = Student('rishi', 1)

student2 = Student('bunny', 2)

student1.show()

lap1 = student1.lap

lap2 = student2.lap

lap1.show()

<https://www.programiz.com/python-programming/examples/factorial-recursion>

def rec\_fact(n):

    if n == 1:

        return n

    else:

        return n\*rec\_fact(n-1)

num = 5

if num < 0:

    print('fact does not exist for negative no')

elif num == 0:

    print('factorial of zero is 1')

else:

    print('factorial of ', num, 'is', rec\_fact(num))

<https://www.programiz.com/python-programming/examples/fibonacci-recursion>

<https://www.programiz.com/python-programming/examples/natural-number-recursion>