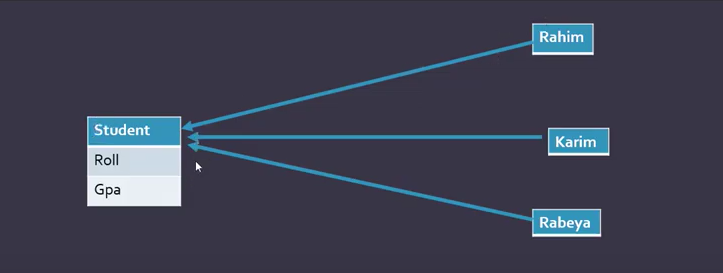
**Object-oriented Python:**

In Python, object-oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming.

**Class**

**A Python class is like an outline or a template for creating a new object.** An object is anything that you wish to manipulate or change while working through the code. Every time a class object is instantiated, which is when we declare a variable, a new object is initiated from scratch.

****

Here, Student is a class and Rahim, Karim & Rabeya are objects.

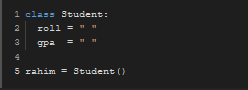


This is how a class is declared/ created. Before being able to use it, **We need to create objects of this class first.** It is conventional to make the first alphabet of the a class capital lettered.

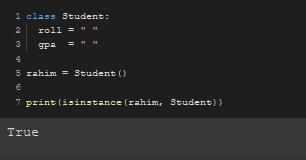
**Object**

An object is a real-life entity. An object is the collection of various data and functions that operate on those data.

Python is an object-oriented programming language. Everything is in Python treated as an object, including variable, function, list, tuple, dictionary, set, etc. Every object belongs to its class.

****

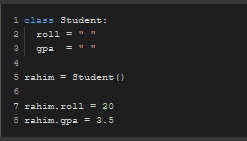
This is how an object is defined/created. **Rahim is an object which belongs to the Student class**. To verify the fact if “rahim” really belongs to the “Student” class or not, We can use the ***“insinstance()”*** function. It will return “True” rahim really belongs to the Student class.



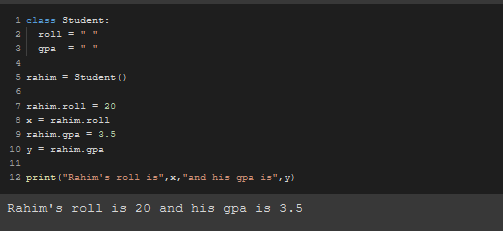
**Note:** ***“isinstance()”*** function syntax.

isinstance(object\_name, class\_name)

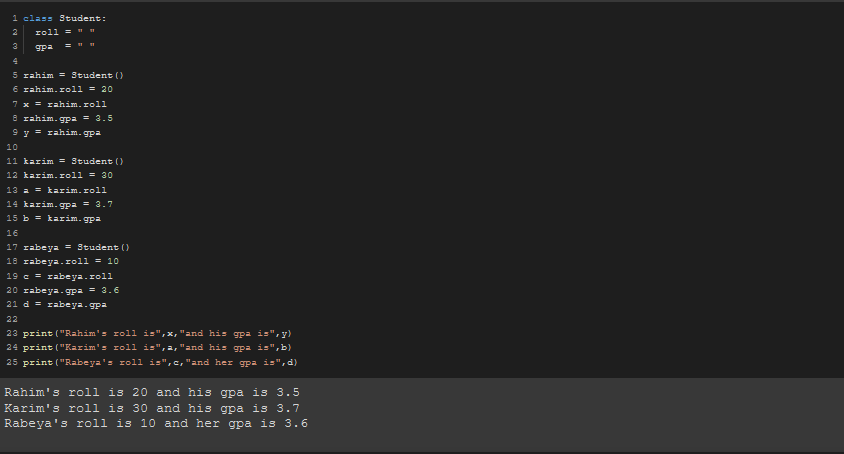
***Now, rahim has the properties of the class Student.*** Rahim can use the “roll” and “gpa" attributes. ***To assign values to those attributes, We use the dot (.) method.***



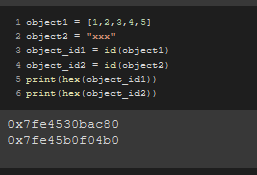
To see the assigned values, We can print them.



We can assign as many as objects we want under the one “Student” class.



Every object is stored in the ram with an unique identifier. To see that identifier, We use the “id()” function.

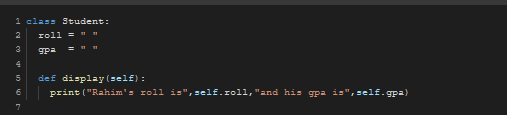


**Method**

In Python, a method is a function that is available for a given object because of the object's type.

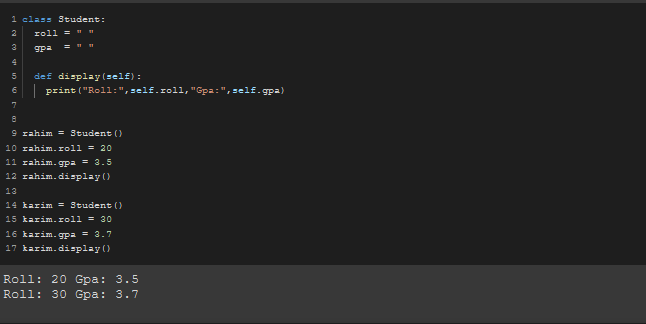
After creating a method, We can use this method as many times as we want under an object.

We need to create the method in the class block and the parameter should be **“self”**.



This screenshot above shows how to declare a method.

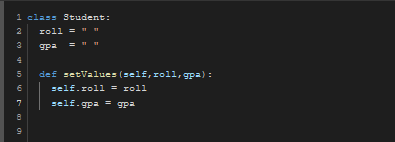
To use it under an object, We use the dot operator.



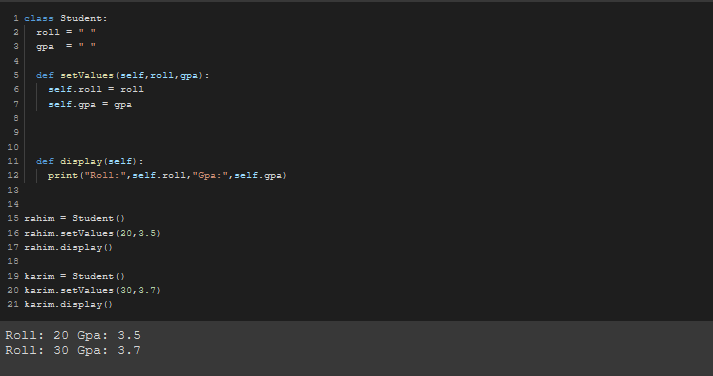
**We can set the values too within the class using another method named “setValues”**.

We can set the parameters this way using **“self”** as the first parameter.

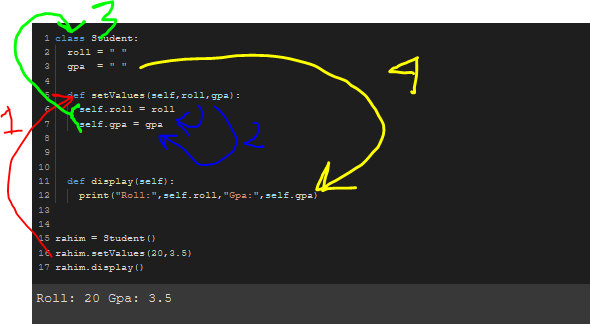
**We then set the variables where the object’s sent values to the class will be saved. These are called “instance variable”.**



**Now, When we call out that setValue function in the object with the given parameter, We can easily print out the desired results with our passed values in the setValue method**



The image below shows the whole process in steps.

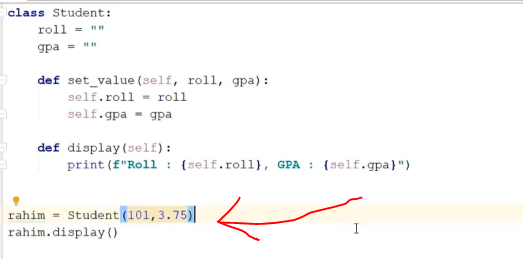


**Note:** Using a method is a much more efficient way to code because we don’t have to print the output manually every time, We can just refer to the class’s method using the object.

**Constructors**

In class-based object-oriented programming, a constructor is a special type of subroutine called to create an object. It prepares the new object for use, often accepting arguments that the constructor uses to set required member variables.

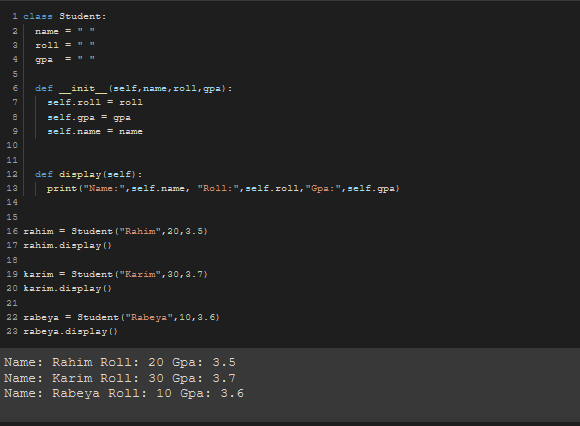
***With a constructor, We can directly pass arguments in a method or class directly when assigning the class into an object simply like this.***



To create a constructor, We use the ***"\_\_init\_\_"*** method.

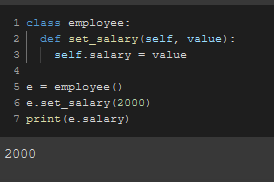
Same as other methods but here, the method name will be “***\_\_init\_\_***” and the rest are the same.

Using this “\_\_init\_\_” method, we can easily pass on the values in a function in a class directly.



**Instance variables:** Instance variables are owned by instances of the class. This means that for each object or instance of a class, the instance variables are different. ... Instance variables, owned by objects of the class, allow for each object or instance to have different values assigned to those variables.

It is initialized with ***self.***

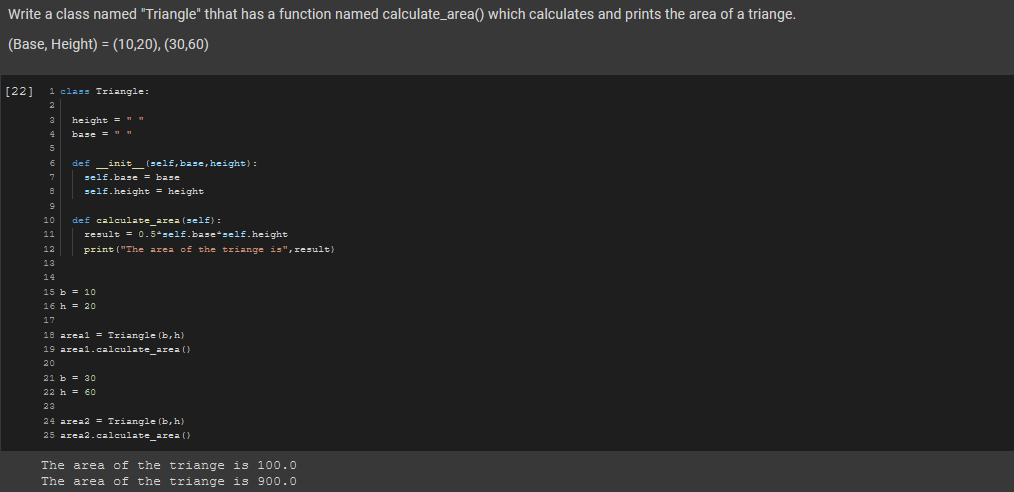
****

Here, ***self.salary*** is an instance variable.

To call methods along with the constructor, We do this.



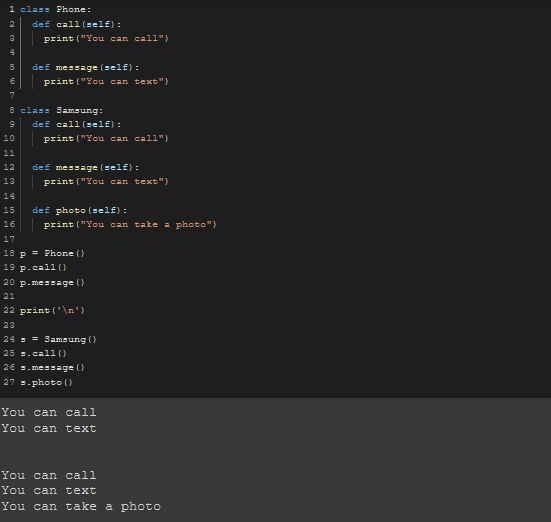
**#Test#**

****

**Inheritance**

**Inheritance is the procedure in which one class inherits the attributes and methods of another class.** *The class whose properties and methods are inherited is known as the Parent class. And the class that inherits the properties from the parent class is the Child class.*

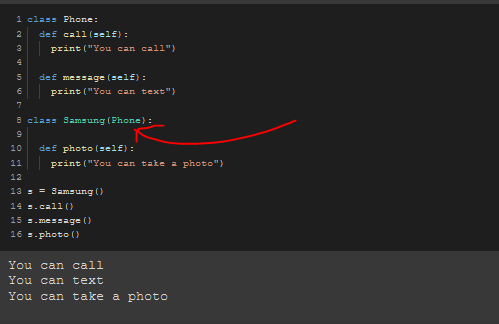
In this code below, There are 2 classes where oen has 2 and other has 3 methods.



In both of the classes, 2 methods like “call” & “message” methods are common. If we could somehow create these 2 common methods inside one class only and call out from there in the second or third or in another class, That’ll be a much efficient way of coding.

We can do this using ***“Inheritance”***.

We can just simply pass the parent class as an argument to the child class to make the child class inherit the attributes of the parent class.

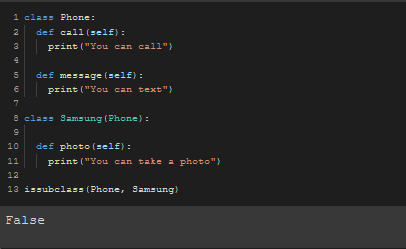
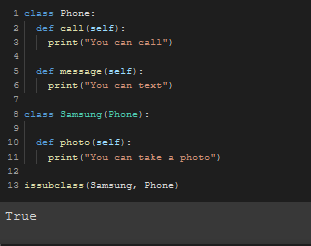


In this code above, There’s no “call” or “message” method in the child “Samsung” class yet the code worked because It inherited the “call” and “message” methods from its parent “Phone” class.

To check if one class is the subclass of another, We use the ***issubclass()*** function.

**Syntax:**

issubclass(child,parent)



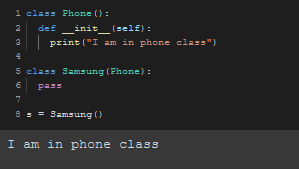
Other name of parent and child classes are,

Parent: Super class, Base class

Child: Sub class, Derived class

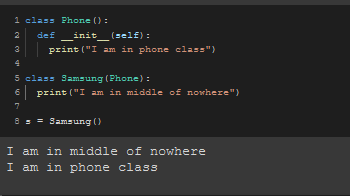
**Method overriding**

In object-oriented programming, is a language feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its superclasses or parent classes.

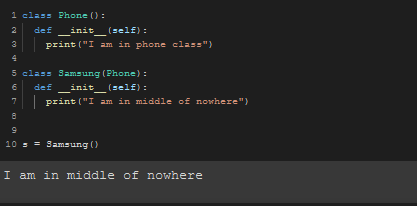


In this code above, There’s nothing in the Samsung class yet printed because it inherits the phone class’s attributes and the phone class has a constructor in it.

If the Samsung class had anything to do like just simply print a message, then it would’ve executed it’s own tasks and then executed the inherited task.



But if the samsung class had it’s own constructor built inside it, It would’ve executed the task of it’s own constructor and not the inherited one due to the ***method overriding***.



But if we needed to execute both of the class’s constructor’s tasks, Then we need to use the ***“super().\_\_init\_\_()”*** method inside the child class’s constructor.

