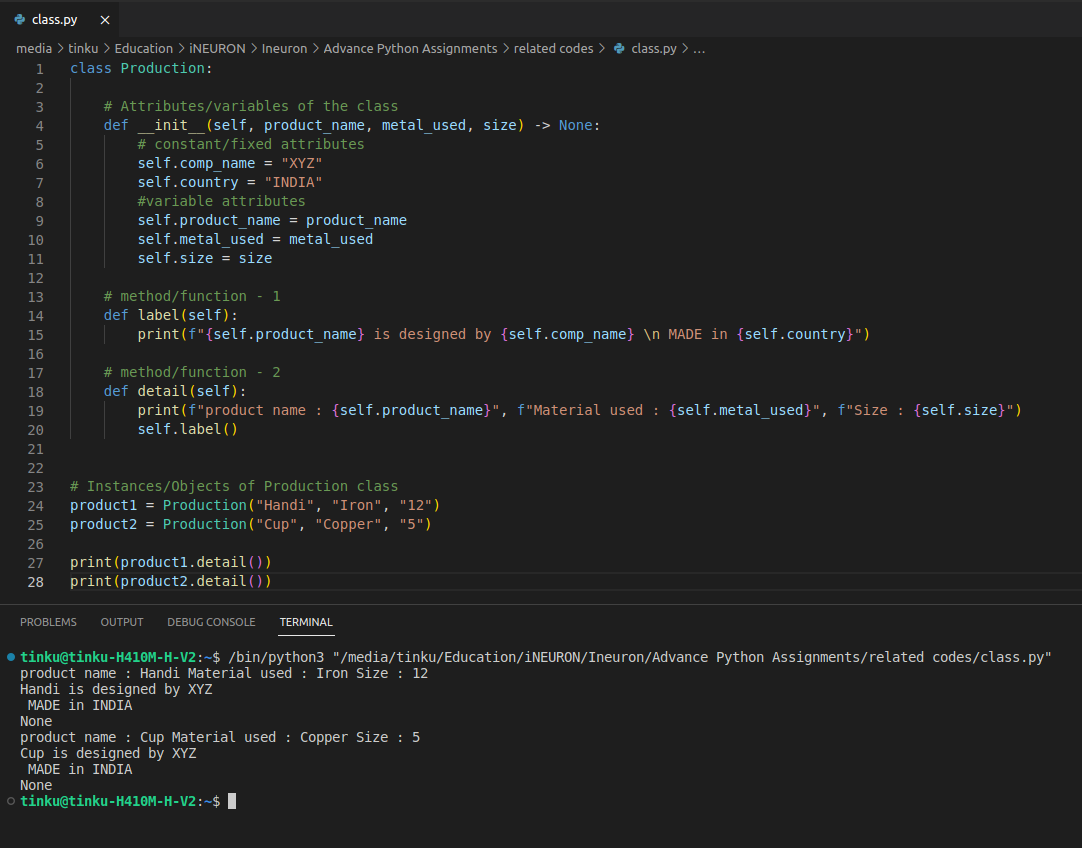
Q1. What is the relationship between classes and modules?

Ans : Classes are the blueprint or templates with which we can instantiate multiple Objects of exactly same type without writing the same code again and again. It can contain constants, variables and functions and even inherit another class.

Module is a file that can contain constants, variables, functions, classes or executable python scripts ehich we can use by importing that module as a whole or a part of it. But we can’t create instances or objects of a module.

Q2. How do you make instances and classes?

Ans :

Q3. Where and how should be class attributes created?

Ans : The attributes should be created within the special method called “\_\_init\_\_” function. These attributs will be assigned to all the objects of that class and it gets called automatically whenever an object is created.

Q4. Where and how are instance attributes created?

Ans : Instance attributes are those which to be provided as arguments while making any object/instance of that class.

Q5. What does the term "self" in a Python class mean?

Ans : “self” is the first or default argument of any class which reffers to the instance or the object itself. In other words, “self” is the pointer which helps the program to indentify and differentiate among all the instances of a class.

For example,

a = “Apple”

b = “Batsman”

here both a and b are two instances of the same class called “str”.

Now, if we use one method of str class, say lower()

Code: a.lower() -->> output: “apple”

The main question is how does the “str” class know that for which instance (a or b) it has to apply the lower() method?

The answer to this question is “self”.

Q6. How does a Python class handle operator overloading?

Ans : In Python we can redefine the default operation of any regular operator.

For example, 3 + 2 >>> 5 but “Tinku” + “Hore” >>> “Tinku Hore”

Here + operator performing different operations depending on the class of the objects which is calling it.

We can do the same for any custom class also. Consider the following example

The Algebra class takes an instance variable x which is any Real number.

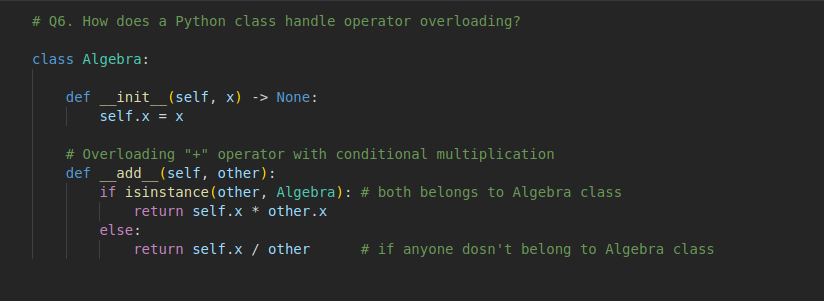
It has a method named “\_\_add\_\_” which reffers to “+” operator. In this method the operation of “+” operator is redefined as follows:

“+” will take two inputs, first one belongs to Algebra class and 2nd one can be of any class.

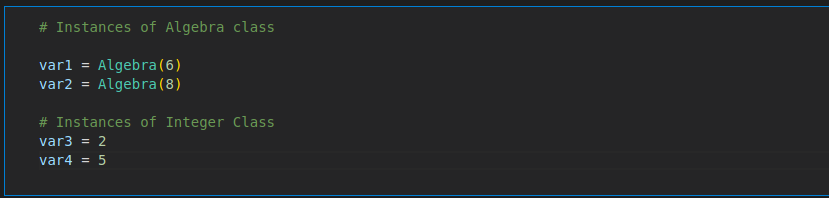
If both the inputs are of Algebra class, then it will return the product as output.

If the 2nd input is not of Algebra class, then it will return the quotient as output.

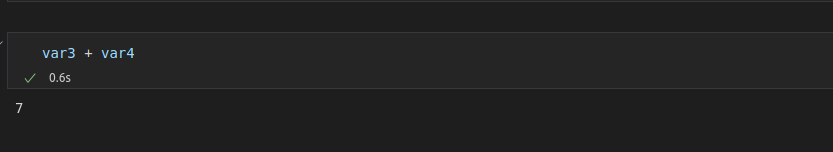
If both inputs are not of Algebra class then “+” will perform its default operation



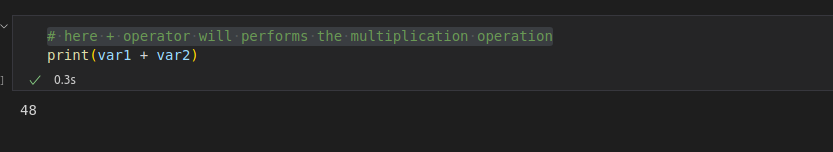
Now, four objects are created, first 2 of Algebra class and last two of Int class.



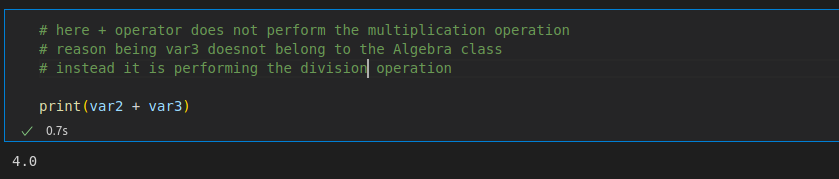
When “+” operates with objects that are not of Algebra class, it simply add them



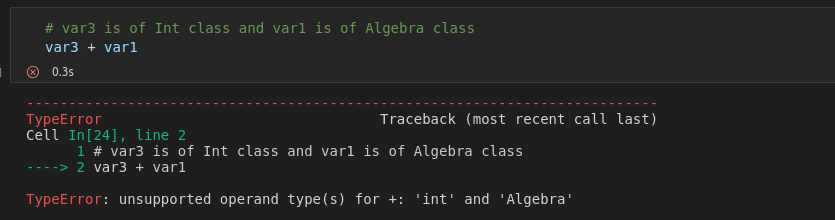
When “+” operates with objects both of Algebra class, it multiplies them



When operates with objects of Algebra and Int class respectively, it performs division



When “+” operates with objects of Int and Algebra class respectively, it raises an error as it is neither satisfying the default condition nor the condition defined in Algebra class



Q7. When do you consider allowing operator overloading of your classes?

Ans : In situations where a functionality is used very frequently, if we assign that particular functionality of our custom class to any particular operator, by using operator overloading method, then the coding will be much easier and time saving compared to defining a function to perform that functionality and calling it again and again. But we should resist the temptation of overloading operators "just for coolness"

Q8. What is the most popular form of operator overloading?

Ans : I think, “+” is the most popular and most used form of operator overloading.

Q9. What are the two most important concepts to grasp in order to comprehend Python OOP code?

Ans : There are four fundamental concepts of Object-oriented programming –

Inheritance,

Encapsulation,

Polymorphism, and

Data abstraction.

It is very important to know about all of these in order to understand OOPs. Out of them two most important are **Inheritance and Polymorphism.**