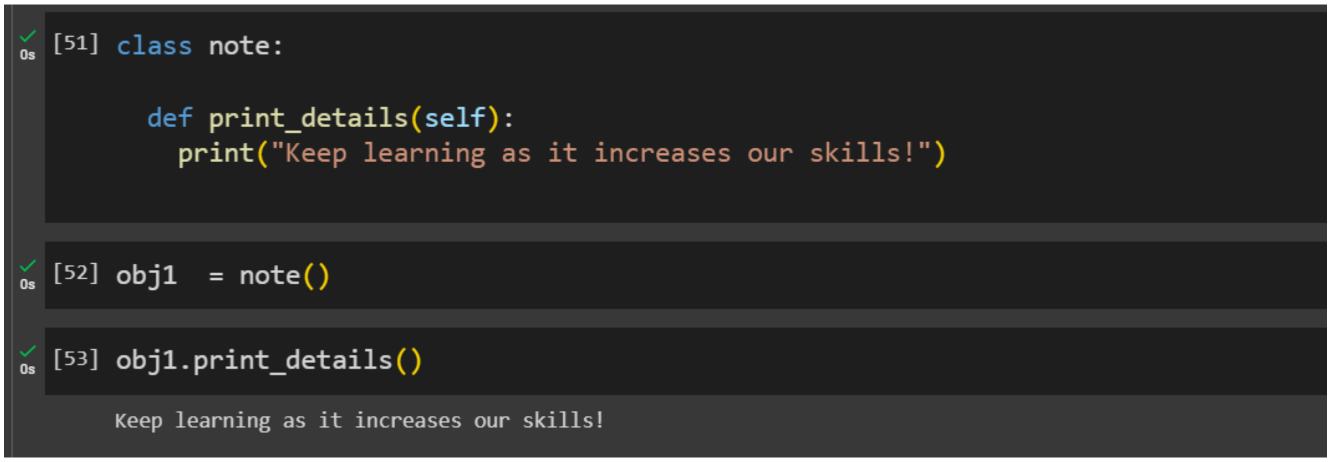
5 FEBRUARY ASSIGNMENT

Ques 1. Explain Class and Object with respect to Object-Oriented Programming. Give a suitable example.
Ans 1. Class is a blueprint representing a real world entity. It defines the properties and behavior of the objects of the class.
An object is an entity belonging to a particular class. It is an instance of the class. A class can have an infinite number of objects with different attribute values.
Example:



```
[51] class note:

       def print_details(self):
         print("Keep learning as it increases our skills!")
```

```
[52] obj1  = note()
```

```
[53] obj1.print_details()

     Keep learning as it increases our skills!
```

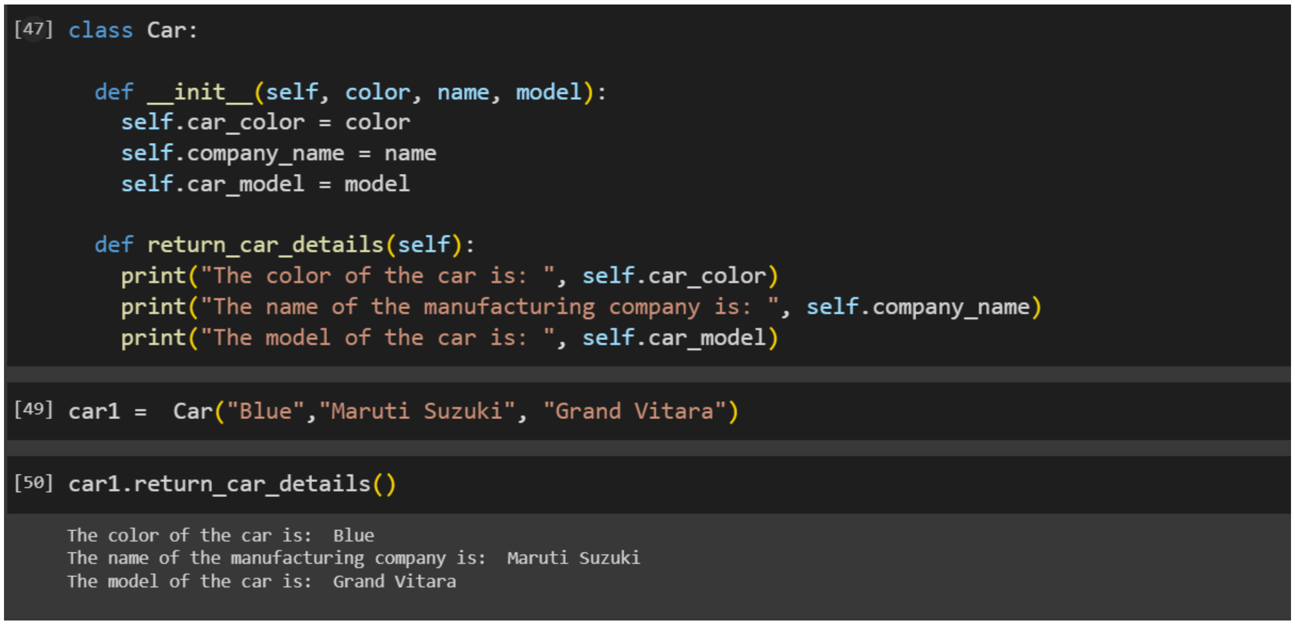
Ques 2. Name the four pillars of OOPs
Ans 2. The 4 pillars of OOPs are:

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

Ques 3. Explain why the __init__() function is used. Give a suitable example.
Ans 3. The __init__() function is a special method in Python classes that is automatically called when an object is created from the class. It is commonly used to initialize the attributes of the object and perform any necessary setup or configuration.

The primary purpose of the __init__() function is to ensure that all necessary attributes are assigned values when an object is created. It allows us to define and set the initial state of an object. By providing default or user-defined values for the attributes, we can ensure that every object created from the class starts with the desired initial values.

Example:


```
[47] class Car:

       def __init__(self, color, name, model):
         self.car_color = color
         self.company_name = name
         self.car_model = model

       def return_car_details(self):
         print("The color of the car is: ", self.car_color)
         print("The name of the manufacturing company is: ", self.company_name)
         print("The model of the car is: ", self.car_model)
```

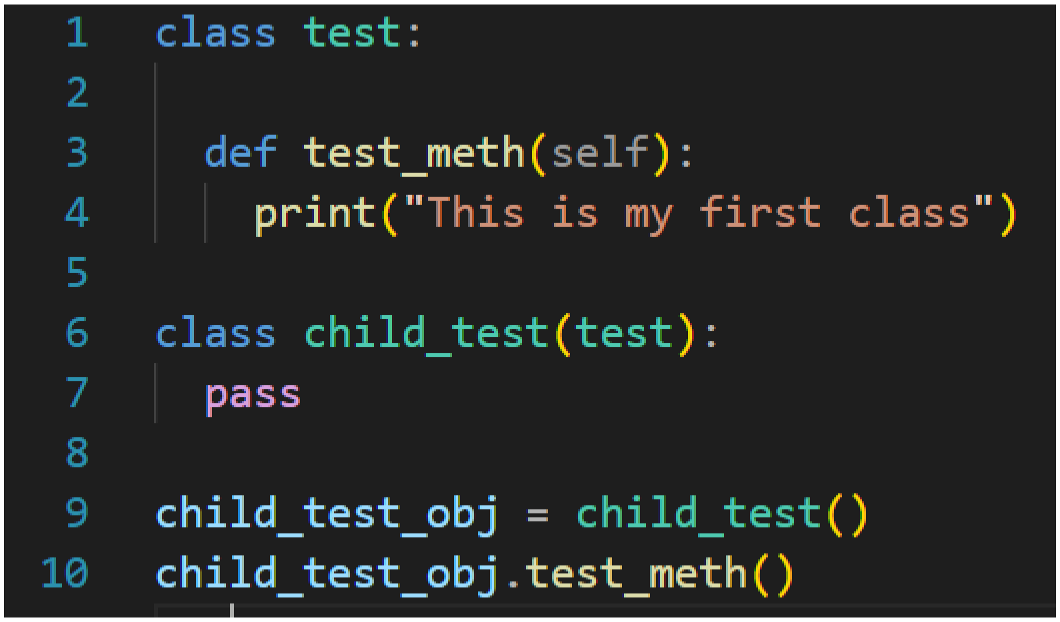
```
[49] car1 =  Car("Blue","Maruti Suzuki", "Grand Vitara")
```

```
[50] car1.return_car_details()

     The color of the car is:  Blue
     The name of the manufacturing company is:  Maruti Suzuki
     The model of the car is:  Grand Vitara
```

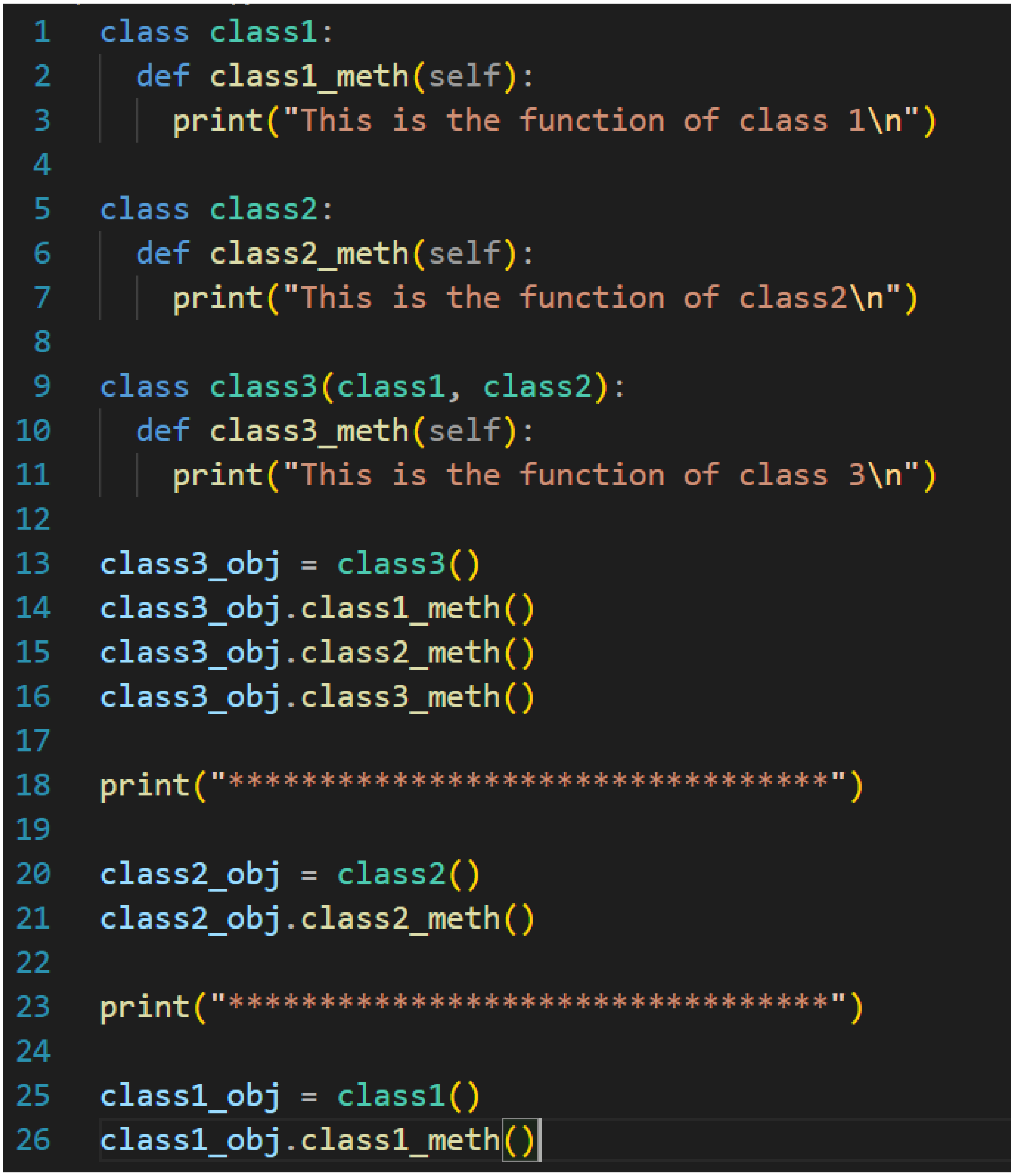
Ques 4. Why self is used in OOPs?
Ans 4. ‘Self’ is a convention used in Python to bind the instances and methods to the class. If we do not use self then the object of that class will give an error. It is used to manipulate the methods and attributes of an object belonging to a particular class.

Ques 5. What is inheritance? Give an example for each type of inheritance
Ans 5. Inheritance is defined as the mechanism of inheriting the properties of the base class to the child class.
__**Types of Inheritance:**__

1. Single Inheritance

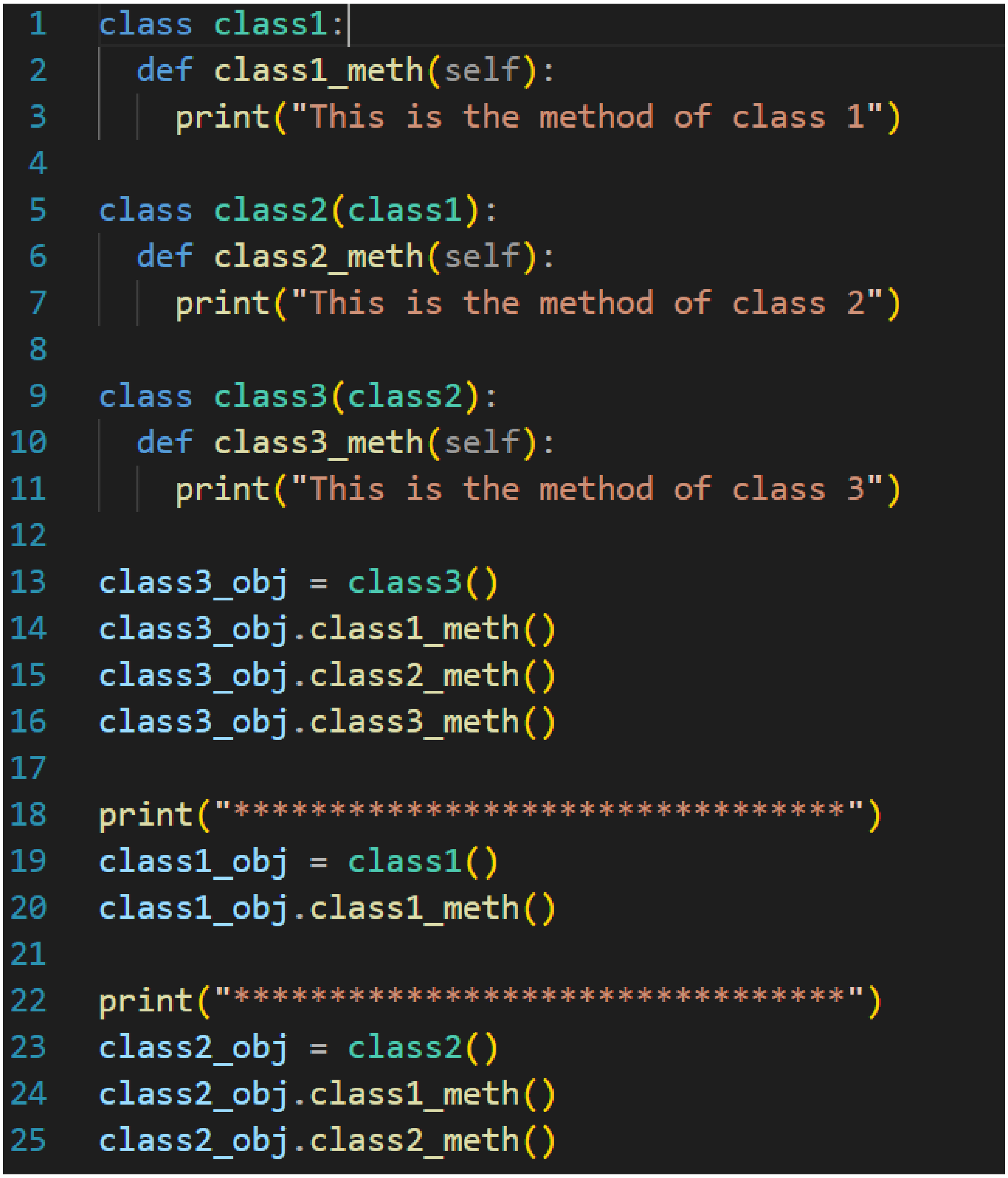

```
class test:

  def test_meth(self):
    print("This is my first class")

class child_test(test):
  pass

child_test_obj = child_test()
child_test_obj.test_meth()
```

2. Multiple Inheritance



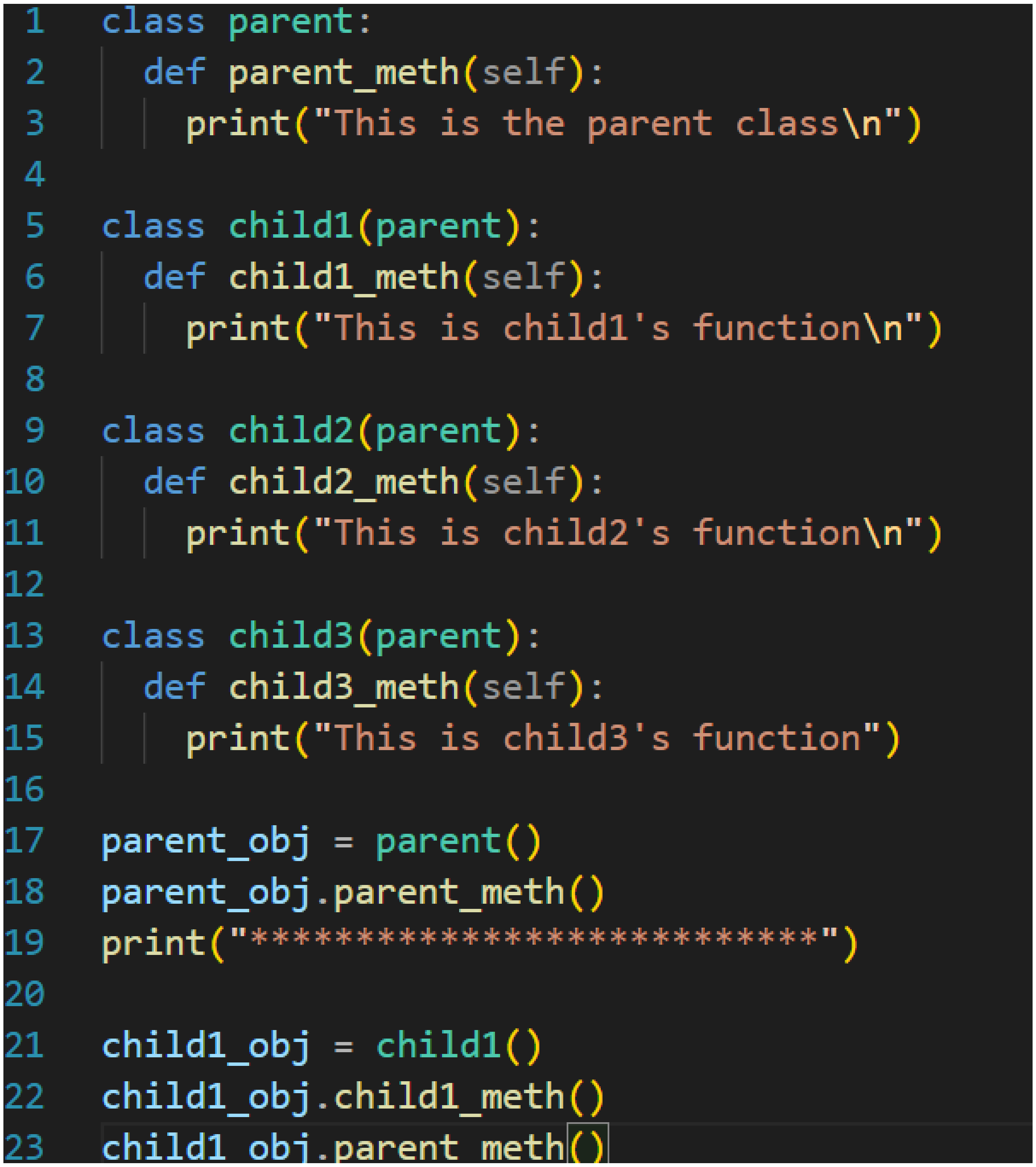
```python
class class1:
  def class1_meth(self):
    print("This is the function of class 1\n")

class class2:
  def class2_meth(self):
    print("This is the function of class2\n")

class class3(class1, class2):
  def class3_meth(self):
    print("This is the function of class 3\n")

class3_obj = class3()
class3_obj.class1_meth()
class3_obj.class2_meth()
class3_obj.class3_meth()

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

class2_obj = class2()
class2_obj.class2_meth()

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

class1_obj = class1()
class1_obj.class1_meth()
```

3. Multi level Inheritance



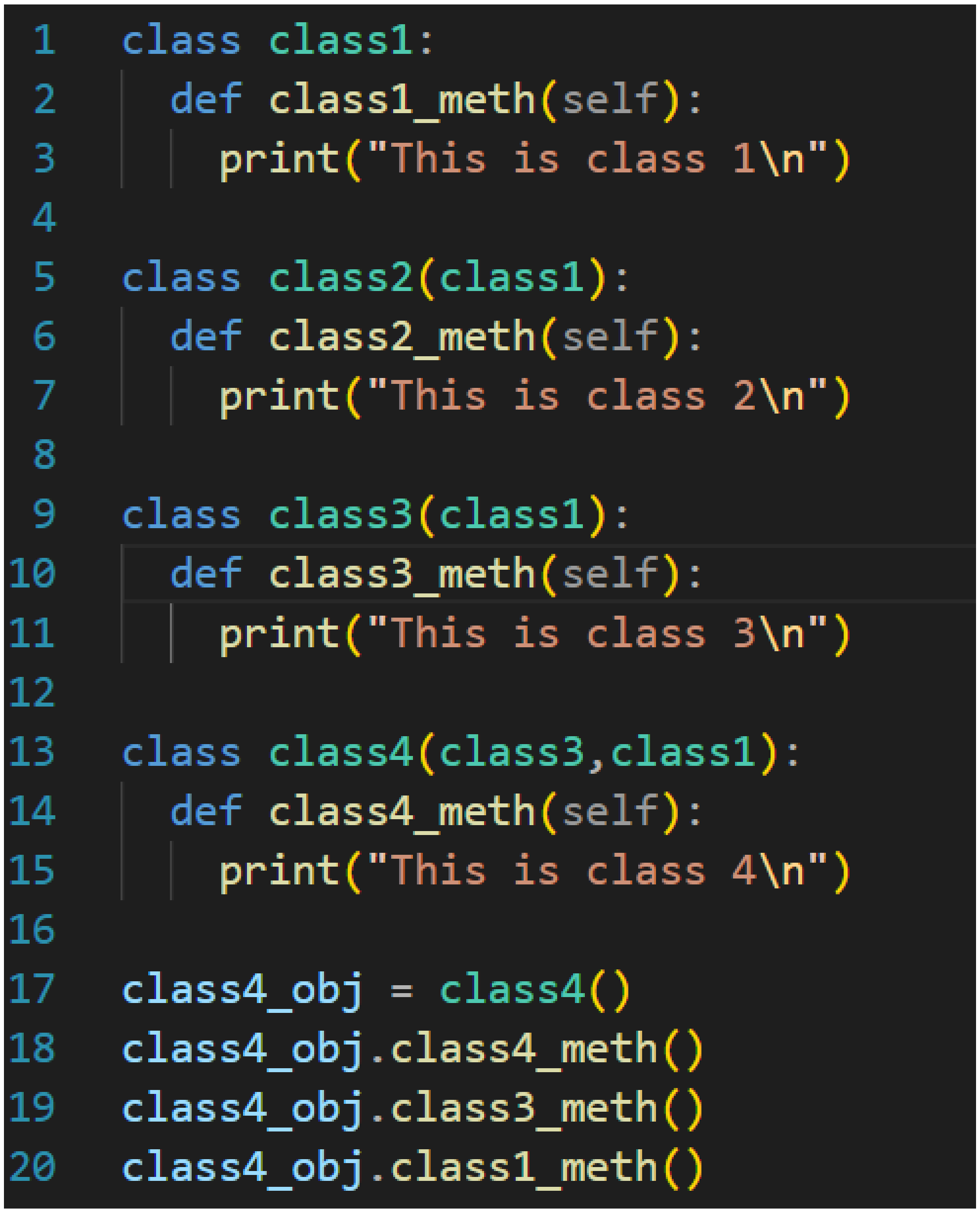
```python
class class1:
  def class1_meth(self):
    print("This is the method of class 1")

class class2(class1):
  def class2_meth(self):
    print("This is the method of class 2")

class class3(class2):
  def class3_meth(self):
    print("This is the method of class 3")

class3_obj = class3()
class3_obj.class1_meth()
class3_obj.class2_meth()
class3_obj.class3_meth()

print("********************************")
class1_obj = class1()
class1_obj.class1_meth()

print("********************************")
class2_obj = class2()
class2_obj.class1_meth()
class2_obj.class2_meth()
```

4. Hierarchical Inheritance



```
class parent:
    def parent_meth(self):
        print("This is the parent class\n")

class child1(parent):
    def child1_meth(self):
        print("This is child1's function\n")

class child2(parent):
    def child2_meth(self):
        print("This is child2's function\n")

class child3(parent):
    def child3_meth(self):
        print("This is child3's function")

parent_obj = parent()
parent_obj.parent_meth()
print("****************************")

child1_obj = child1()
child1_obj.child1_meth()
child1_obj.parent_meth()
```

5. Hybrid Inheritance



```python
class class1:
    def class1_meth(self):
        print("This is class 1\n")

class class2(class1):
    def class2_meth(self):
        print("This is class 2\n")

class class3(class1):
    def class3_meth(self):
        print("This is class 3\n")

class class4(class3,class1):
    def class4_meth(self):
        print("This is class 4\n")

class4_obj = class4()
class4_obj.class4_meth()
class4_obj.class3_meth()
class4_obj.class1_meth()
```