Python Programming

Lambda Function

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.

```
Lambda arguments : expression
```

```
x = lambda a : a + 10
print(x(5))
```

```
Output:
15
```

Map Function

- **map()** function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)
- Syntax:

```
map(fun, iter)
```

fun : It is a function to which map passes each element of given iterable.

iter : It is a iterable which is to be mapped.

Map Function

map.py > ...

```python
def double(n):
    return n * 2

numbers = [5, 6, 7, 8]
result = map(double, numbers)
print(list(result))
```

Reduce Function

- The **reduce()** function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along.This function is defined in “functools” module.
- Syntax:

reduce(func, iterable[, initial])

fun : *It is a function to execuate on each element of the iterable objec*

iter : *It is iterable to be reduced*

Reduce Function

```python
import functools

# Define a list of numbers
numbers = [1, 2, 3, 4]

# Use reduce to compute the product of list elements
product = functools.reduce(lambda x, y: x * y, numbers)
print("Product of list elements:", product)
```

Filter Function

- The **filter()** method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.
- Syntax:

filter(function, sequence)

fun : *f*unction that tests if each element of a sequence is true or not.

seq : sequence which needs to be filtered, it can be sets, lists, tuples, or containers of any iterators.

Filter Function

```
# Define a function to check if a number is even
def is_even(n):
    return n % 2 == 0

# Define a list of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Use filter to filter out even numbers
even_numbers = filter(is_even, numbers)
print("Even numbers:", list(even_numbers))
```

Exception Handling

- Try except block is used to handle error with code
- SyntaxError, cannot be handling
- The try block lets you test a block of code for errors.
- The except block lets you handle the error.
- The else block lets you execute code when there is no error.
- The finally block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling

```
try:
    .....# Code where error may occur
except IndexError:
    .....# Handle IndexError
except (NameError,ZeroDivisionError):
    .....# Handle multiple exception types
except:
    ..... # Handle all other exception
else:
    .... #else block lets you execute code when there is no error
finally:
    .... # finally block lets you execute code, regardless of the result of the try- and except blocks.
```

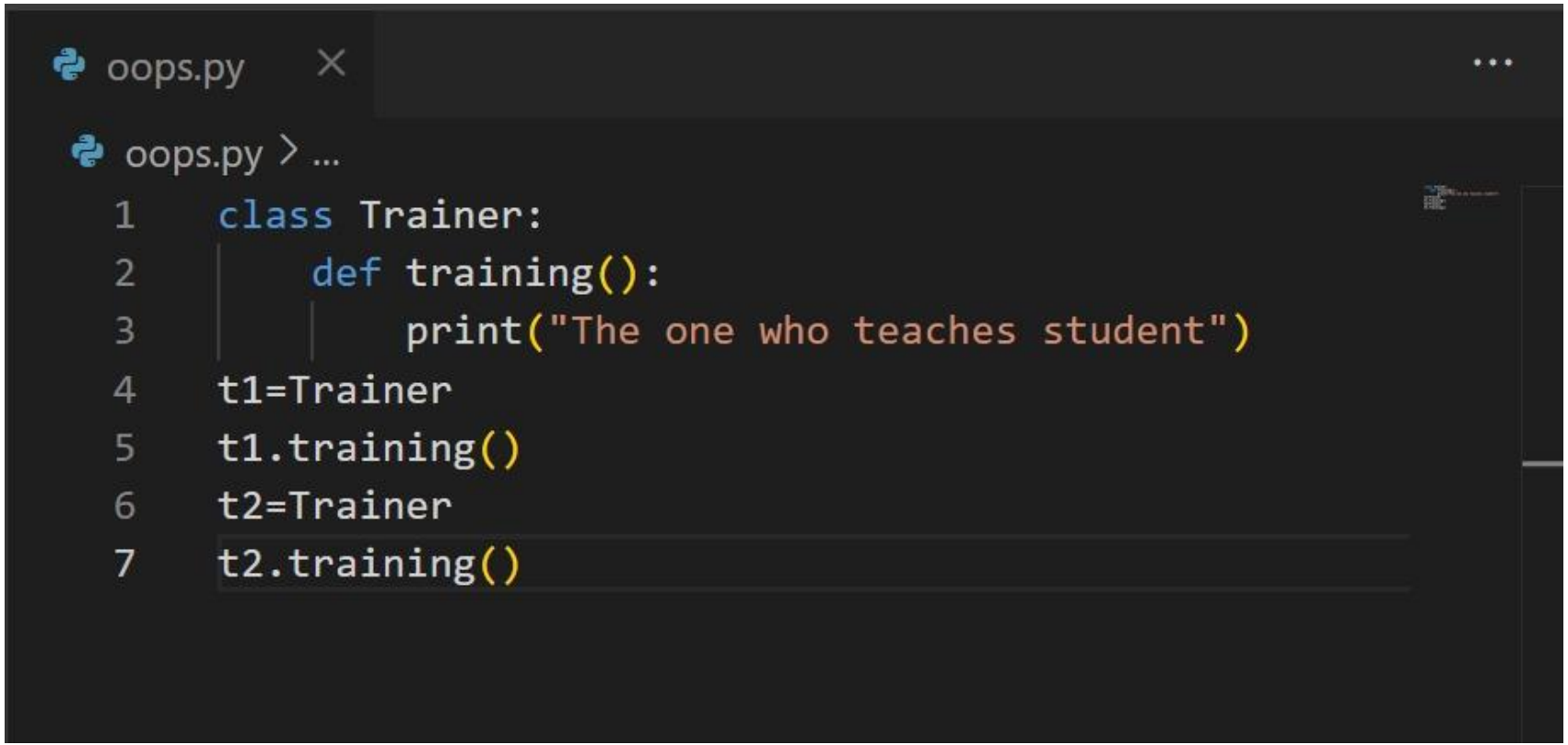
Classes and Objects

- Python is an object oriented programming language, almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.
- An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values.ccc

An Object Consists of

- **State:** It is represented by the attributes of an object. It also reflects the properties of an object.
- **Behaviour:** It is represented by the methods of an object. It also reflects the response of an object to other objects.

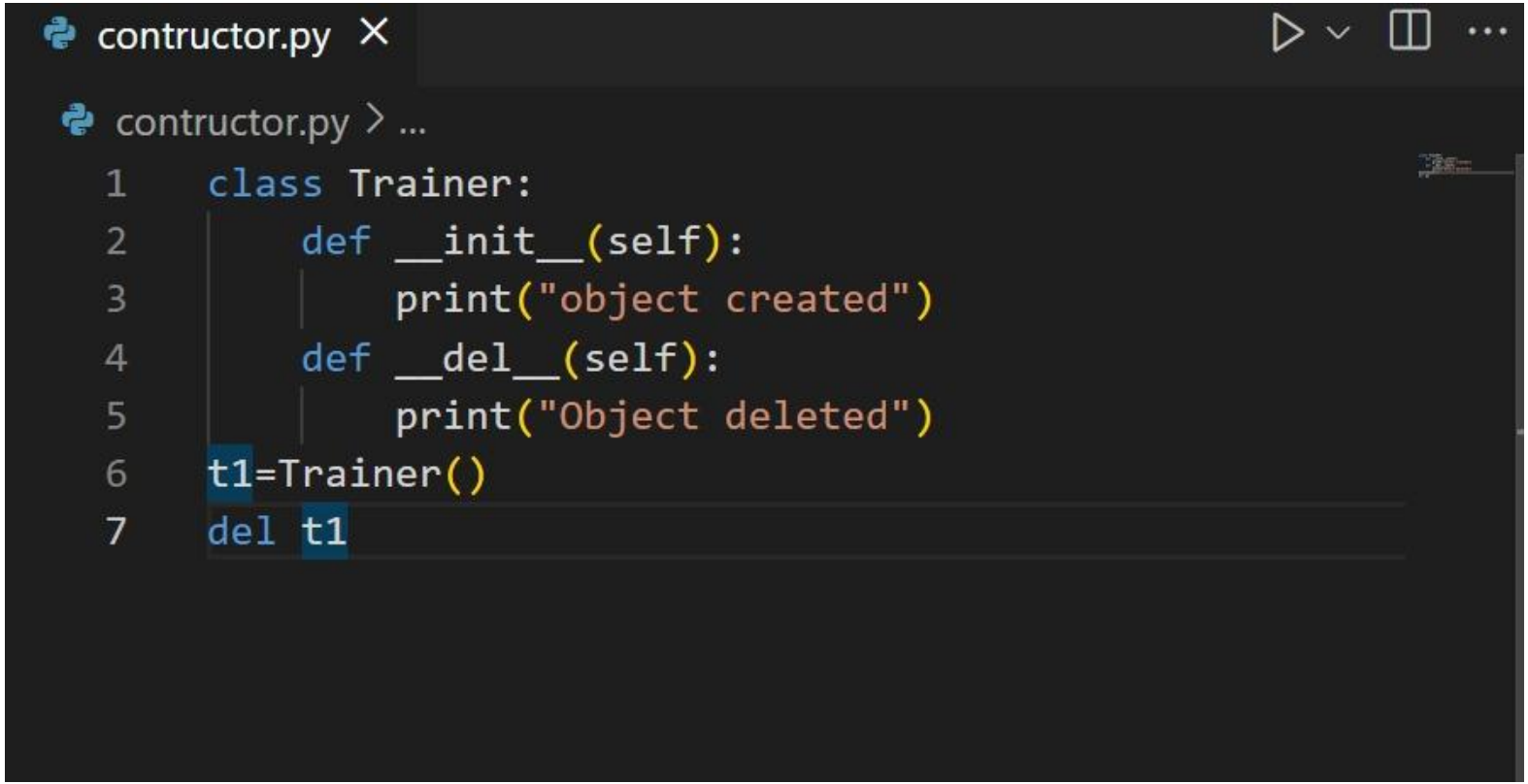
Classes and Objects



Constructors and Destructors

- These methods are called when an object is created, and are used to initialize the object's attributes and set up its initial state. Constructors are defined with the __init__ method.
- These methods are called when an object is about to be destroyed, and are used to perform cleanup actions like releasing resources, closing files, or freeing memory. Destructors are defined with the __del__ method.

Constructors and Destructors



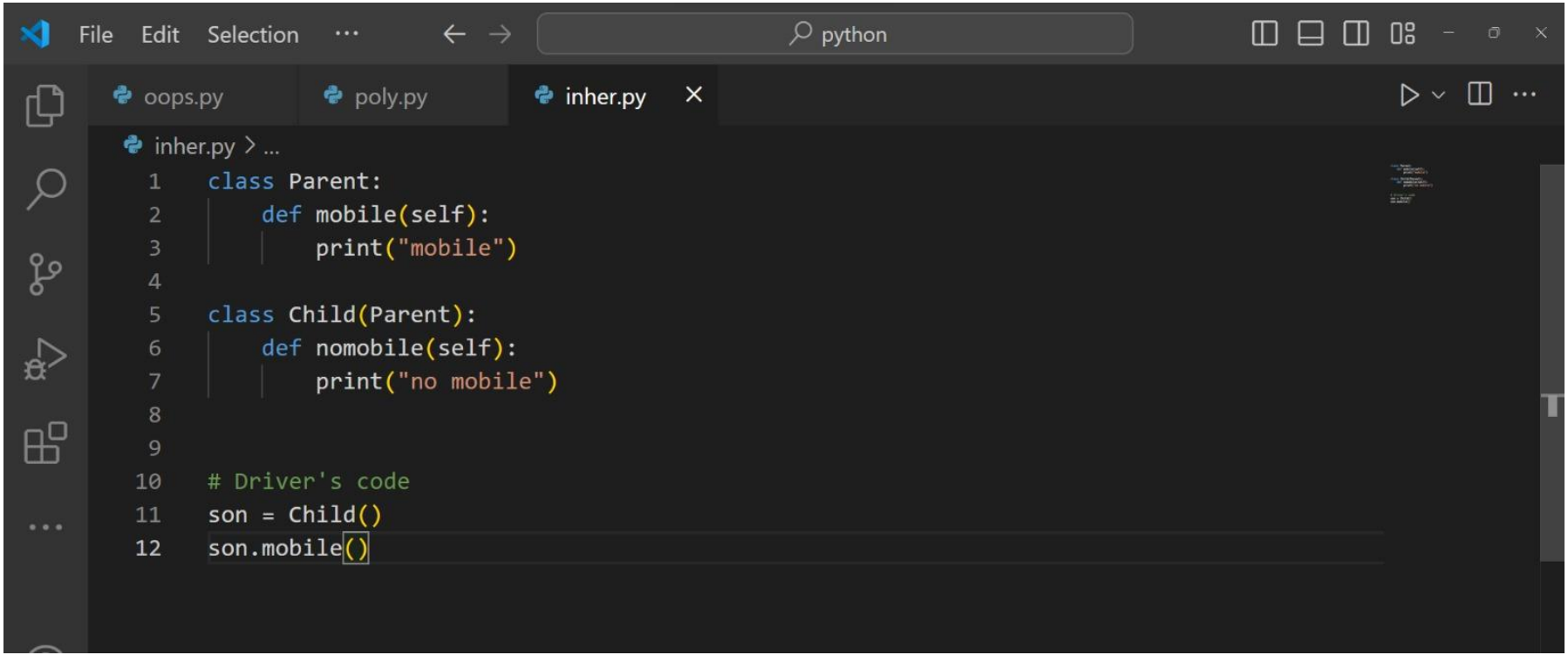
Inheritance

One of the core concepts in **object-oriented programming (OOP)** languages is inheritance. It is a mechanism that allows you to create a hierarchy of classes that share a set of properties and methods by deriving a class from another class. Inheritance is the capability of one class to derive or inherit the properties from another class.

Types of Inheritance

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Single Inheritance

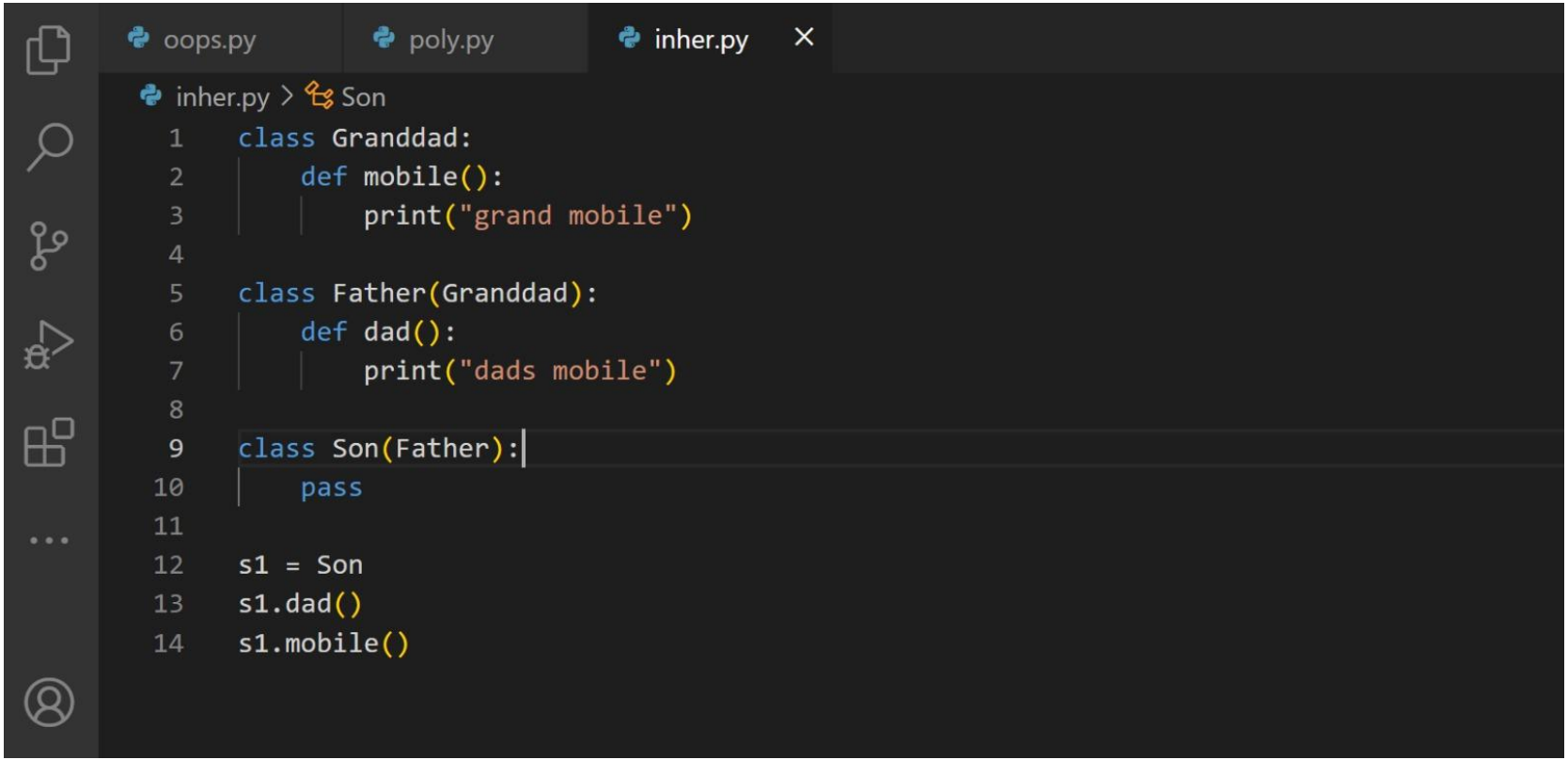


Multiple Inheritance

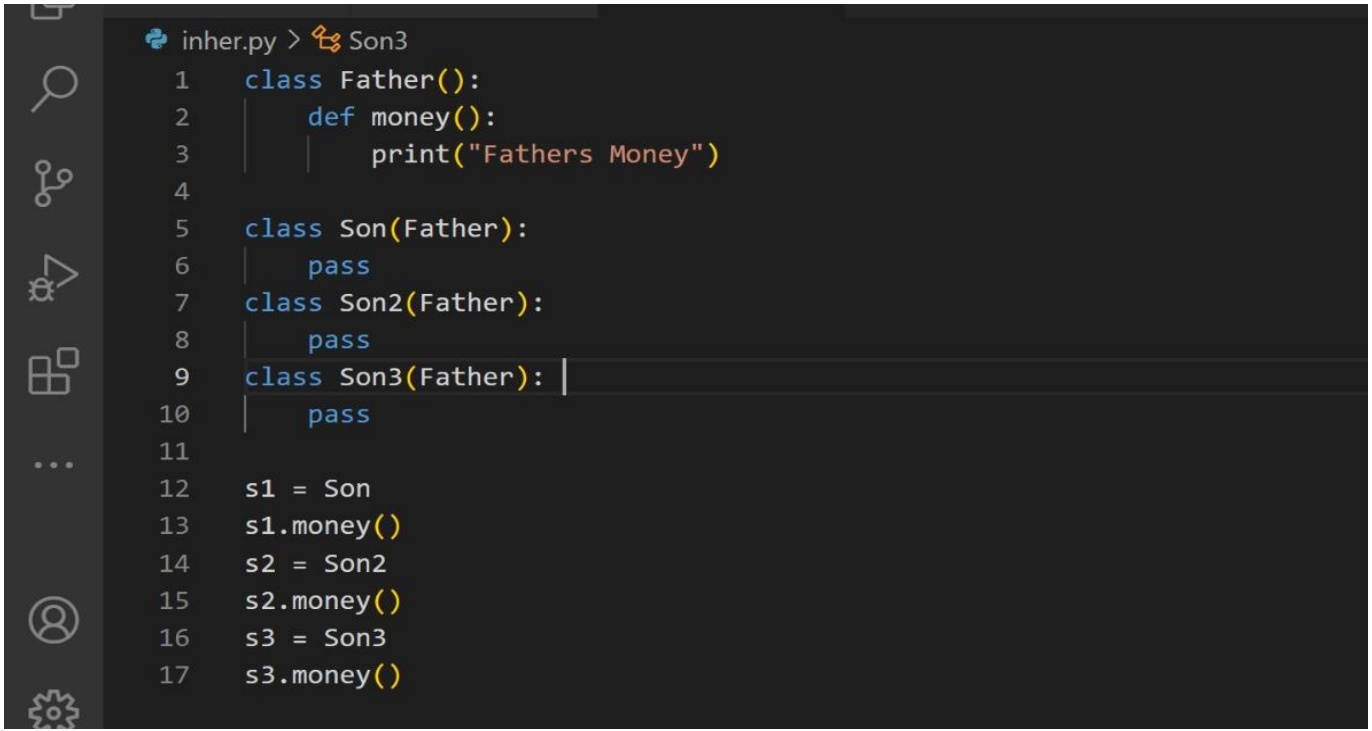
inher.py > ...

```
1   class Mother:
2       def mom():
3           print("moms mobile")
4
5   class Father:
6       def dad():
7           print("dads mobile")
8
9   class Son(Mother, Father):
10      pass
11
12  s1 = Son
13  s1.dad()
14  s1.mom()
```

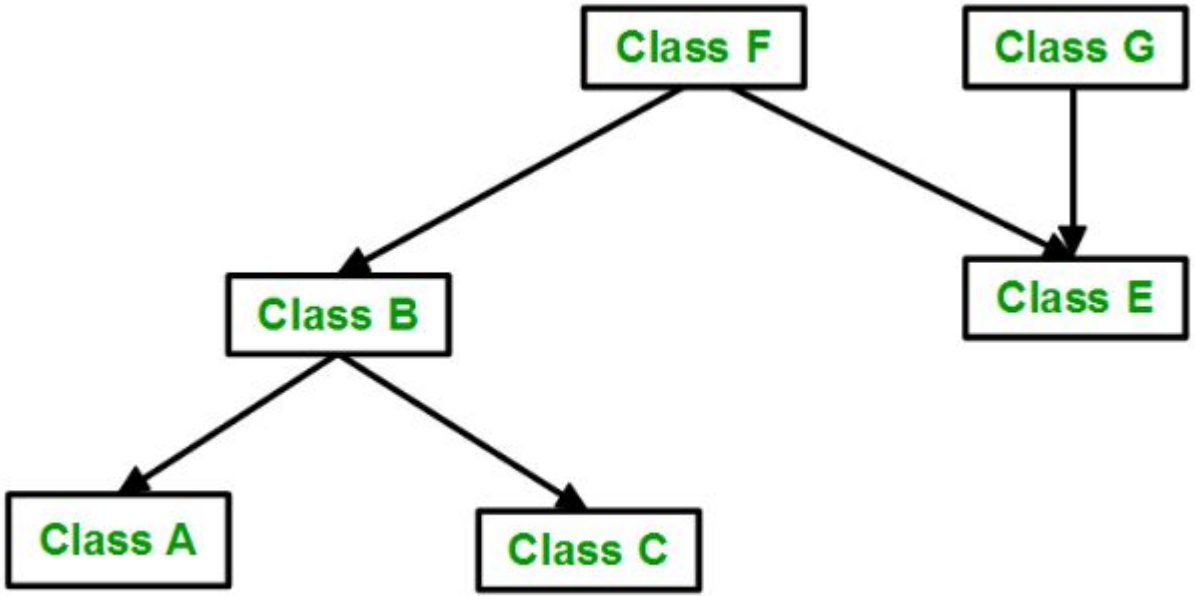
Multilevel Inheritance



Hierarchical Inheritance



Hybrid Inheritance



Hybrid Inheritance

```
class School:
    def func1(self):
        print("This function is in school.")


class Student1(School):
    def func2(self):
        print("This function is in student 1. ")


class Student2(School):
    def func3(self):
        print("This function is in student 2.")


class Student3(Student1, School):
    def func4(self):
        print("This function is in student 3.")


# Driver's code
object = Student3()
object.func1()
object.func2()
```

Encapsulation

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object’s variable can only be changed by an object’s method. Those types of variables are known as **private variables.**

Encapsulation

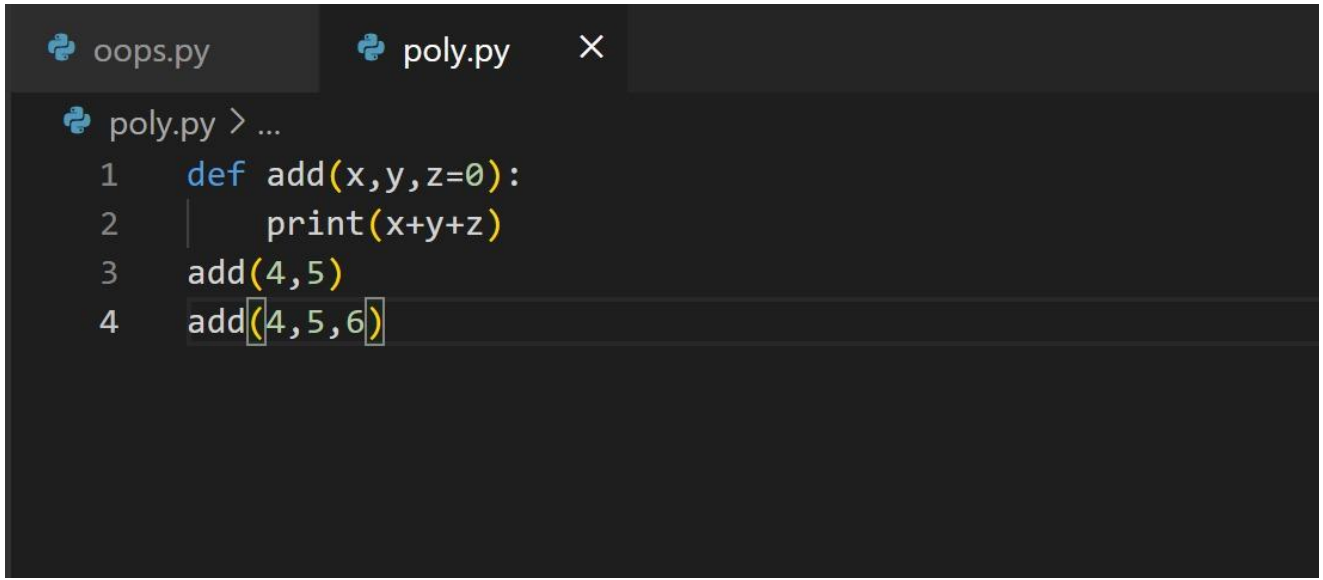
```
# Python program to
# demonstrate protected members

# Creating a base class
class Base:
    def __init__(self):

        # Protected member
        self._a = 2

# Creating a derived class
class Derived(Base):
    def __init__(self):

        # Calling constructor of
        # Base class
        Base.__init__(self)
        print("Calling protected member of base class: ",
              self._a)

        # Modify the protected variable:
        self._a = 3
        print("Calling modified protected member outside class: ",
              self._a)


obj1 = Derived()

obj2 = Base()

# Calling protected member
# Can be accessed but should not be done due to convention
print("Accessing protected member of obj1: ", obj1._a)

# Accessing the protected variable outside
print("Accessing protected member of obj2: ", obj2._a)
```

Encapsulation

```
# Python program to
# demonstrate private members

# Creating a Base class


class Base:
    def __init__(self):
        self.a = "GeeksforGeeks"
        self.__c = "GeeksforGeeks"

# Creating a derived class
class Derived(Base):
    def __init__(self):

        # Calling constructor of
        # Base class
        Base.__init__(self)
        print("Calling private member of base class: ")
        print(self.__c)


# Driver code
obj1 = Base()
print(obj1.a)
```

Polymorphism

The word "polymorphism" means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.



Abstraction

Data abstraction is one of the most essential concepts of Python OOPs which is used to hide irrelevant details from the user and show the details that are relevant to the users.

For eg: To Turn on Computer your are pressing the Power Button. **(In Behind)** After pressing power button from battery power supply should need to go to RAM and CPU. From storage to OS go to ram and send to bios setup and windows get booted after loaded completely returns to Ram and windows screened to users.

Module

As our program grows bigger, it may contain many lines of code. Instead of putting everything in a single file, we can use modules to separate codes in separate files as per their functionality. This makes our code organized and easier to maintain.

Module is a file that contains code to perform a specific task. A module may contain **variables, functions, classes** etc

OS Module

Python has a built-in os module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.

Sys Module

Python has a built-in os module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.