**Day 13 Assignment - 18/12/2023 - Vamsi Viswanadham**

**OOPS**

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. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

**Class and Object**

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

The object is an entity that has a state and behavior associated with it. It may be any real-world object like a mouse, keyboard, chair, table, pen, etc.

Here is an example of Class and objects:

class Dog:

atr1 = "mammal" #class attribute

def \_\_init\_\_(self, name): #instance attribute

self.name = name

Rotweiler = Dog("Rotty")

Tommy = Dog("Tommy")

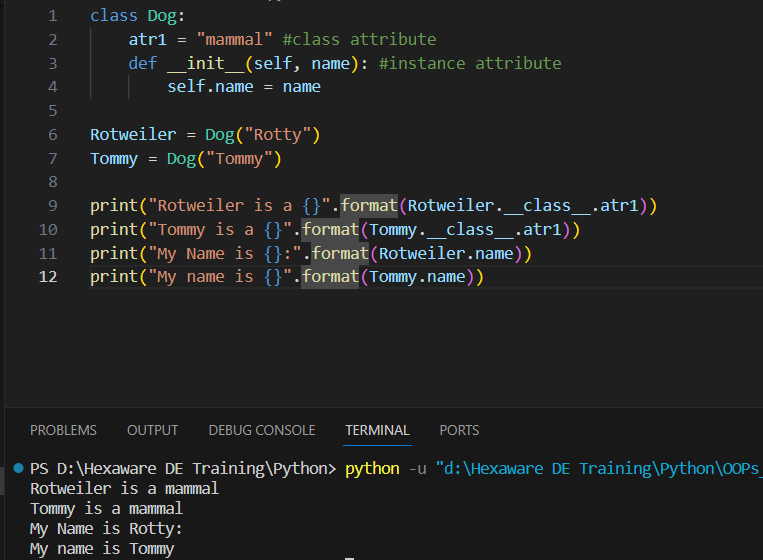
print("Rotweiler is a {}".format(Rotweiler.\_\_class\_\_.atr1))

print("Tommy is a {}".format(Tommy.\_\_class\_\_.atr1))

print("My Name is {}:".format(Rotweiler.name))

print("My name is {}".format(Tommy.name))

Here we have created a class named “Dog” that contains a class attribute and instance attribute.



**Constructor**

The \_\_init\_\_ method is similar to constructors in C++ and Java. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object. Now let us define a class and create some objects using the self and \_\_init\_\_ method.

The above is considered as an example for the init method. There we have just intialized the name of the object to be assigned.

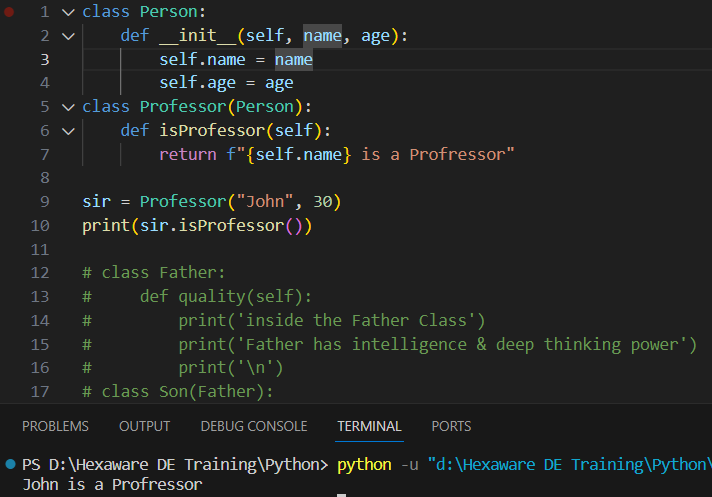
**Inheritance**

Inheritance allows a new class (known as a child or subclass) to acquire properties and behaviors (methods) from another class (called the parent or superclass). This means the subclass can use the code already written in the superclass without having to write it again.

**Types of Inheritance:**

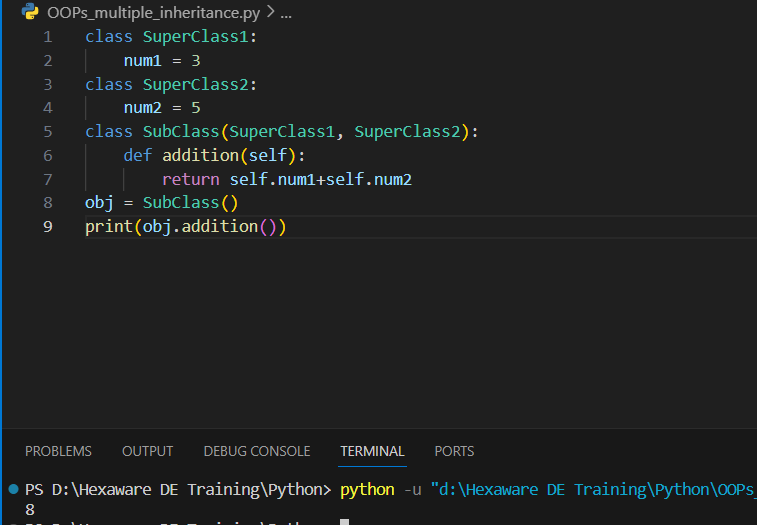
**Single Inheritance:** In single inheritance, a subclass inherits from one superclass. It's a straightforward, one-to-one relationship.

Eg:



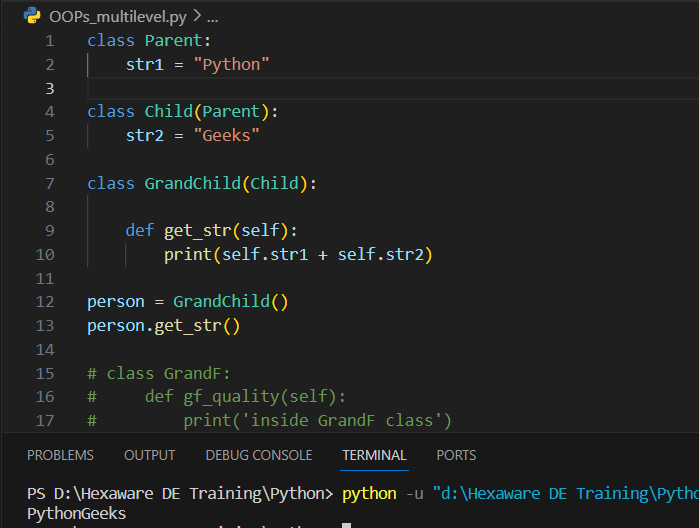
**Multiple Inheritance:** A subclass inherits from more than one superclass. This means it can combine multiple behaviors and attributes from different superclasses.

Eg:



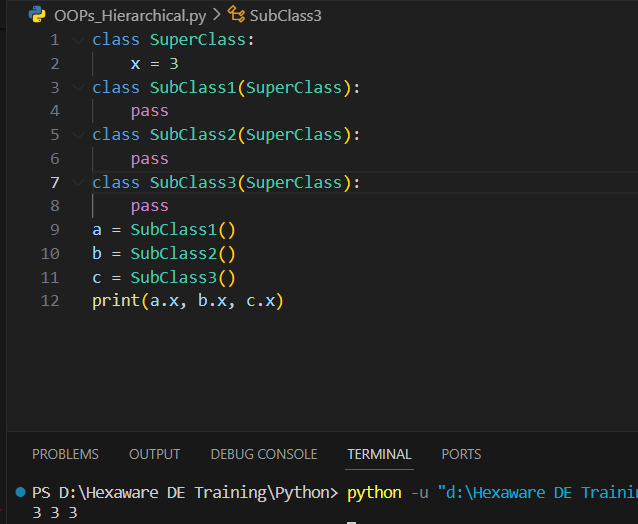
**Multilevel Inheritance:** This involves a chain of inheritance. For example, a class inherits from a second class, which in turn inherits from a third class.

Eg:



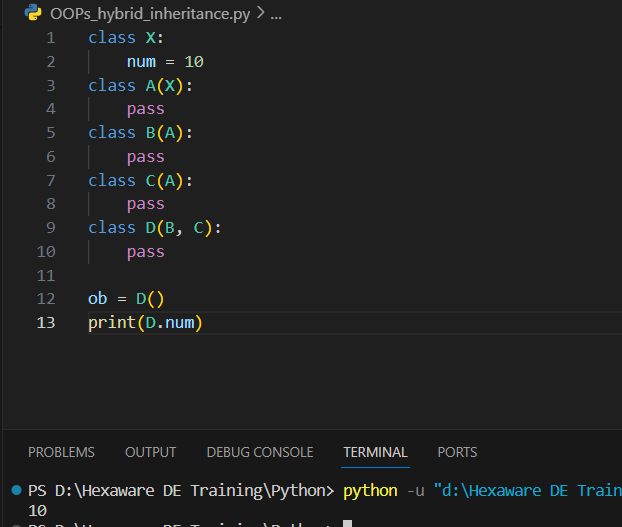
**Hierarchical Inheritance**: In this type, multiple subclasses inherit from a single superclass. Each subclass will have all the features of the superclass but can also have additional features unique to them.

Eg:



**Hybrid Inheritance:** This is a combination of two or more of the above types of inheritance. It can be a mixture of multiple, multilevel, and hierarchical inheritance.

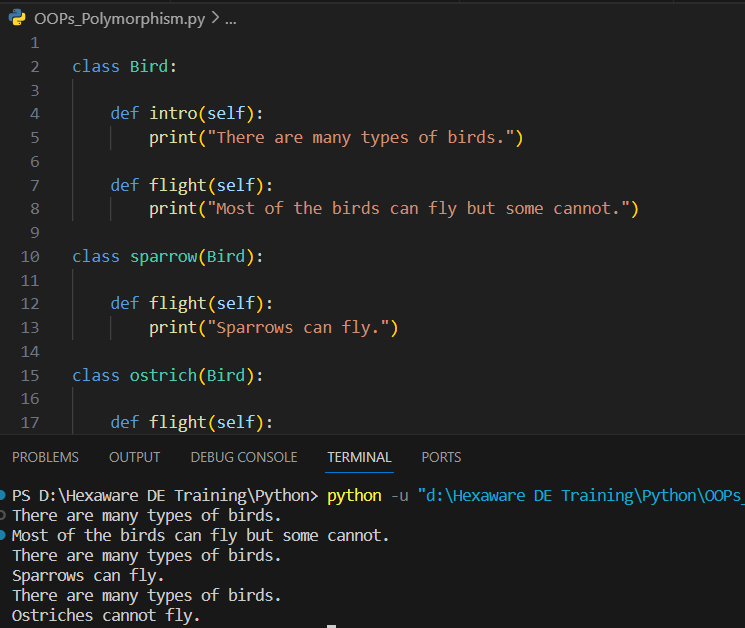
Eg:



**Polymorphism:**

Polymorphism **simply means having many forms**. For example, we need to determine if the given species of birds fly or not, using polymorphism we can do this using a single function.

Here is an example:

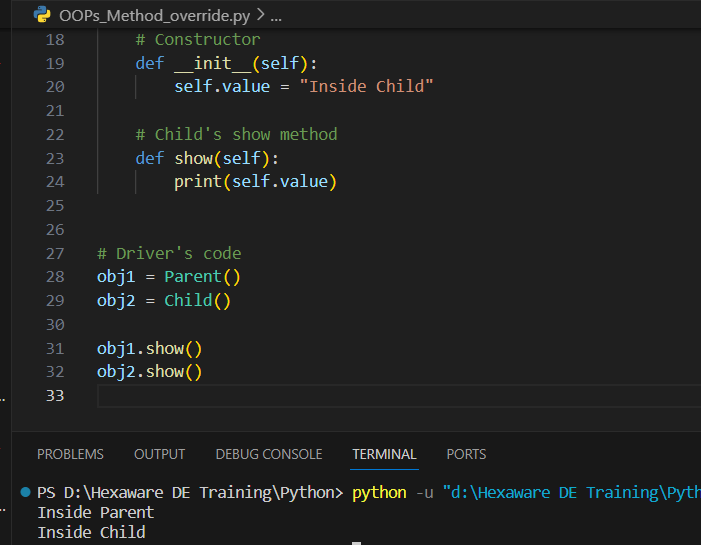


**Method Overriding:**

In method overriding you can reimplement the methods of parent class inside child class wherein the methods of child class and parent class have the similarity in the following features:

same name, parameters or signatures and same return type.

Eg:



**File handling:**

Python file handling refers to the process of reading and writing files. It's a way for programs to save and retrieve data. Here's a simple explanation of file handling in Python, broken down into key points:

**Opening a File:** Python uses the built-in open() function to access a file. This function requires the file path and a mode (like read, write, append) as arguments.

**File Modes:**

Read ('r'): Opens a file for reading. This is the default mode.

Write ('w'): Opens a file for writing. Creates a new file if it does not exist or truncates the file if it exists.

Append ('a'): Opens a file for appending at the end without truncating it. Creates a new file if it does not exist.

Read and Write ('r+'): Opens a file for both reading and writing.

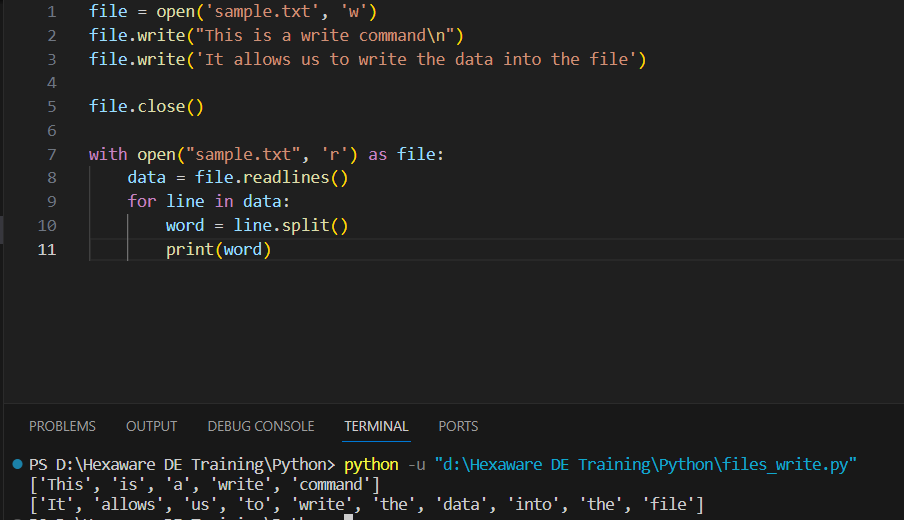
Binary Mode: Add 'b' to the mode (like 'rb', 'wb') for binary files.

Reading from a File: Methods like read(), readline(), or readlines() are used to read data from a file.

**Writing to a File:** Methods like write() or writelines() are used to write data to a file.

**Closing a File:** After completing operations on a file, it should be closed using the close() method to free up system resources.

Here is an example for file handling:



**Modules:**

A Python module is essentially a file that contains Python code, including definitions and statements. Within a module, one can define various elements like functions, classes, and variables, as well as include code that can be executed directly. By organizing related code into modules, it enhances the clarity and usability of the code, as well as provides a logical structure to the codebase.

**Creating a Module:**

Any Python file with a .py extension can be a module. You just need to write Python code in it.

**Using a Module:**

To use the code in a module, you import it into your Python program using the import statement.

Once imported, you can call functions, classes, and variables defined in that module.

Here is a simple example of python modules. Here we have imported math module and use its methods in our file as shown below.

