Python for Computer Science and Data Science 2 (CSE 3652)

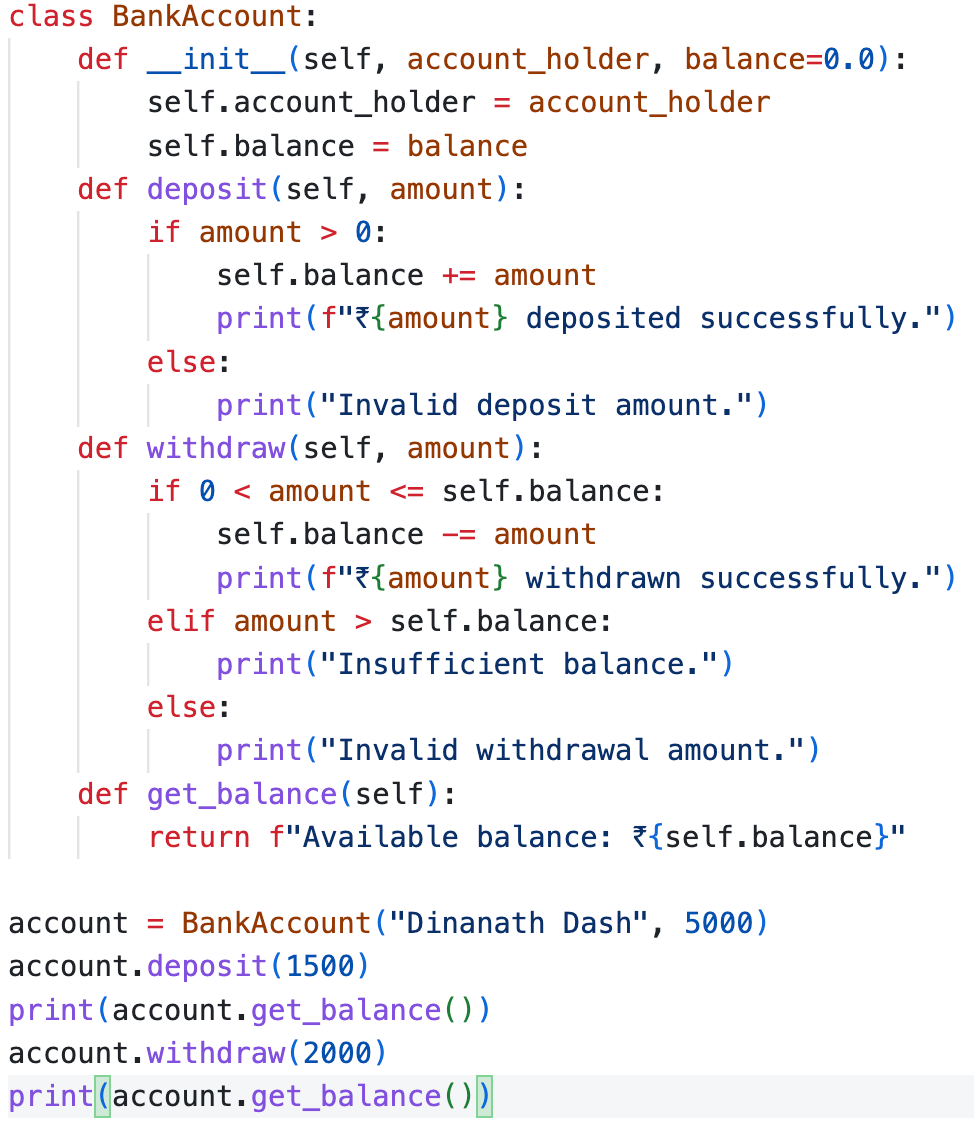
**Minor Assignment-1: Object-Oriented Programming (OOP)**

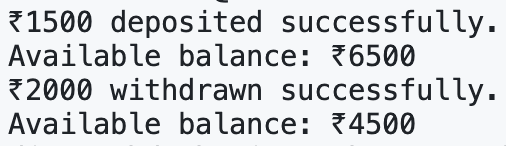
1. What is the significance of classes in Python programming, and how do they contribute to object- oriented programming?

**Ans:-** Classes in Python are fundamental to object-oriented programming (OOP), enabling the creation of reusable and modular code. Here’s why they are significant:

1. Encapsulation - Classes allow bundling of data (attributes) and behavior (methods) together. Access control can be implemented using public, protected, and private members.
2. Reusability and Modularity - Code can be organized into self-contained objects, making it easier to maintain and extend. A class can be instantiated multiple times to create objects with shared structure but unique data.
3. Abstraction - Unnecessary details can be hidden while exposing only the necessary functionalities. Helps in simplifying complex implementations.
4. Inheritance - Allows new classes (child classes) to inherit attributes and methods from existing classes (parent classes). Promotes code reuse and enables hierarchical relationships.
5. Polymorphism - Different classes can define methods with the same name but different implementations. Enables flexibility in designing extensible code.
6. Create a custom Python class for managing a bank account with basic functionalities like deposit and withdrawal?

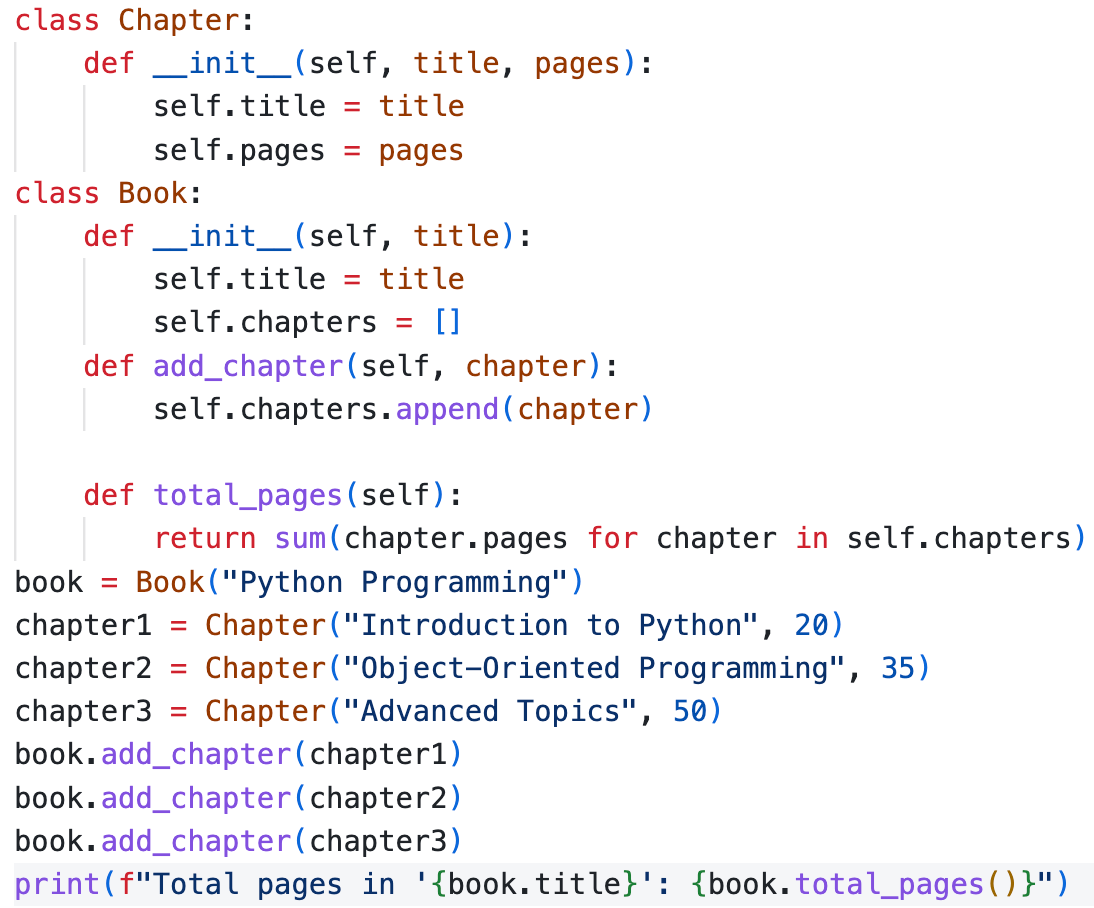
**Ans:-**



Output:- 

1. Create a Book class that contains multiple Chapters, where each Chapter has a title and page count. Write code to initialize a Book object with three chapters and display the total page count of the book.

**Ans:-**

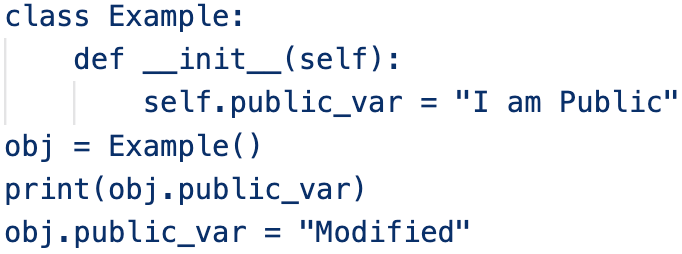


Output:- Screenshot 2025-02-10 at 7.31.30 PM

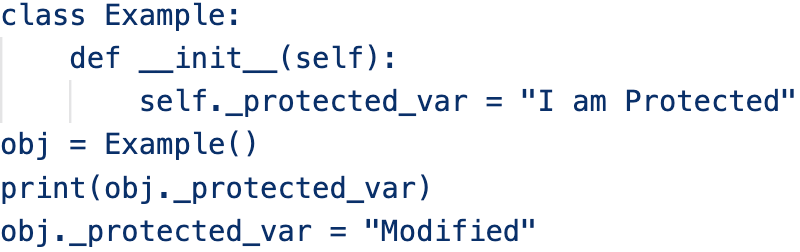
1. How does Python enforce access control to class attributes, and what is the difference between public, protected, and private attributes?

**Ans:-** Python enforces access control through naming conventions rather than strict access modifiers like in Java or C++. It uses prefixes (\_ and \_\_) to indicate different levels of attribute visibility.

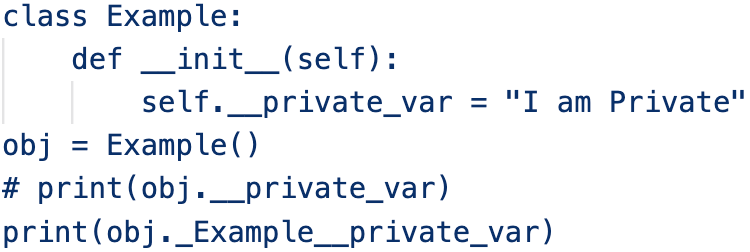
1. Public Attributes (name): Accessible from anywhere (inside and outside the class). No restrictions on modification.

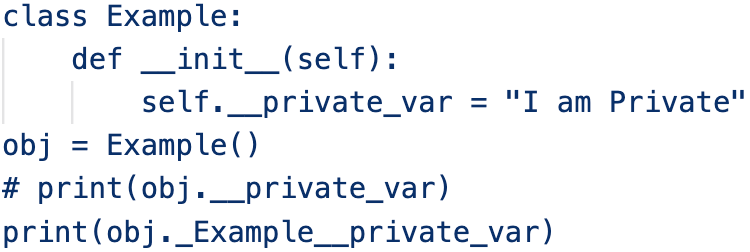


1. Protected Attributes (\_name): Indicated with a single underscore (\_). Not enforced by Python but treated as a convention: "Use this carefully." Can still be accessed but should be considered internal to the class or its sub-classes.



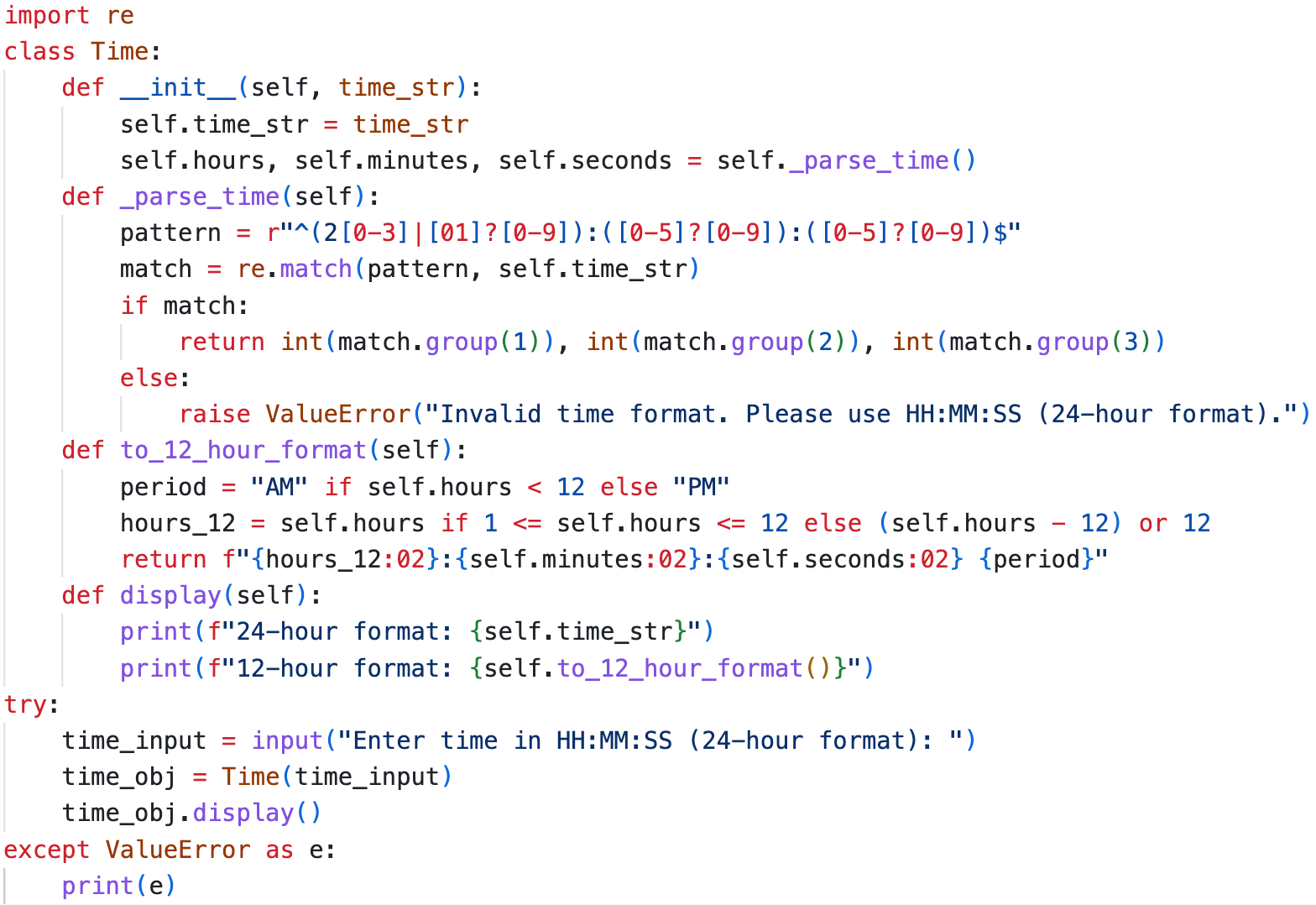
1. Private Attributes (\_\_name): Indicated with double underscores (\_\_). Python performs name mangling, renaming \_\_var to \_ClassName\_\_var, making it harder to access directly. Cannot be accessed directly outside the class but can still be accessed using name mangling.

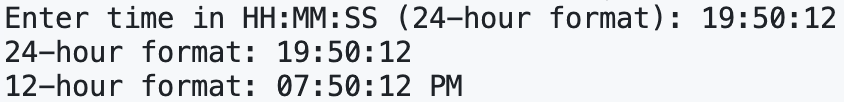




1. Write a Python program using a Time class to input a given time in 24-hour format and convert it to a 12-hour format with AM/PM. The program should also validate time strings to ensure they are in the correct HH:MM:SS format. Implement a method to check if the time is valid and return an appropriate message.

**Ans:-**

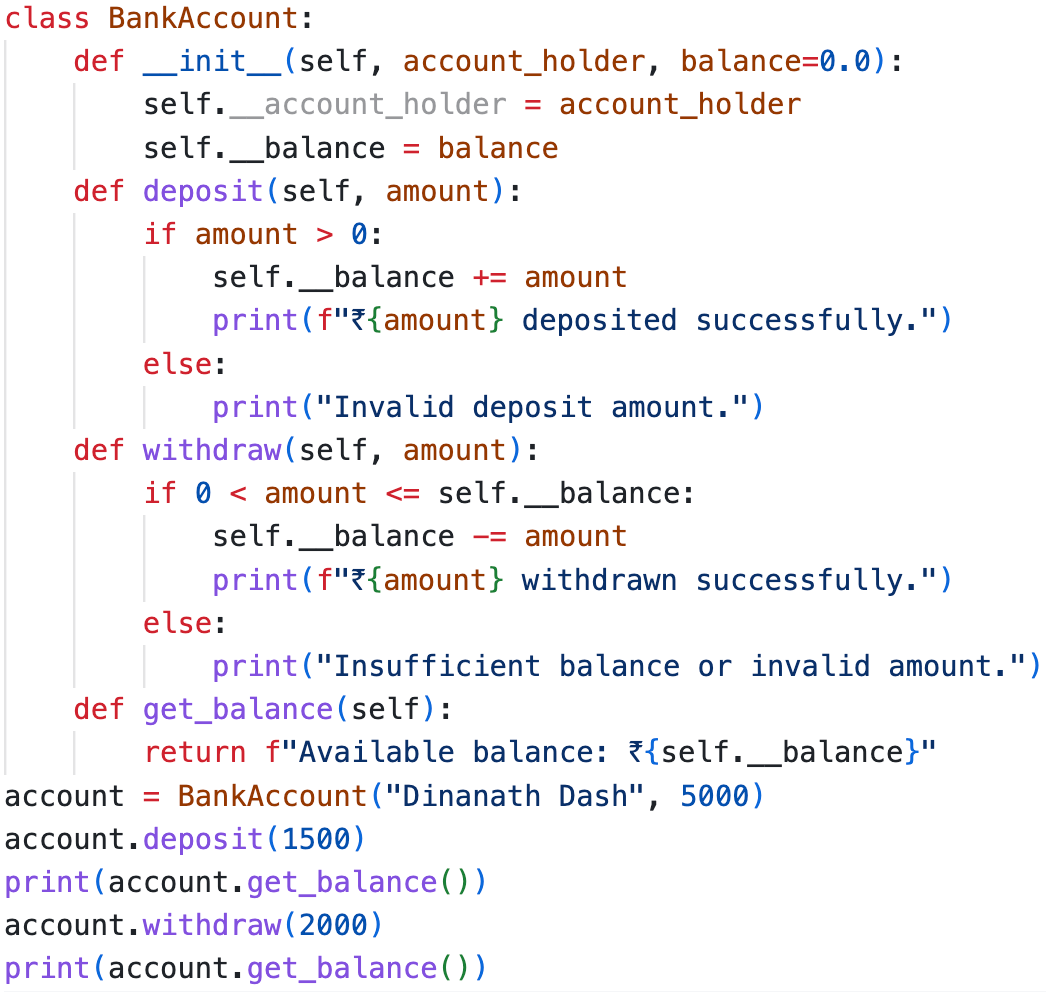
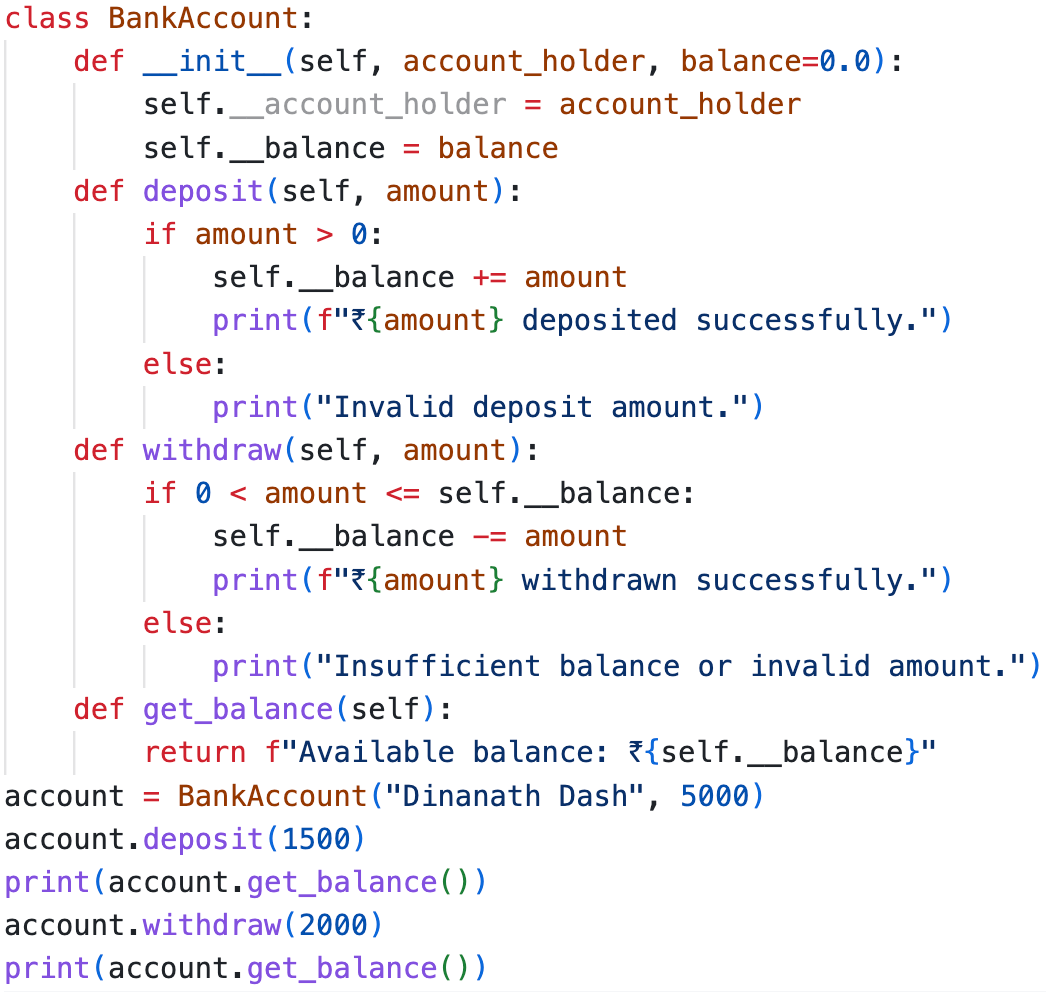


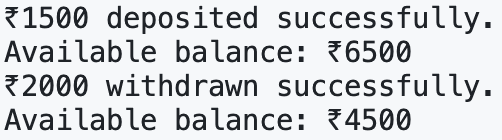
Output:- 

1. Write a Python program that uses private attributes for creating a BankAccount class. Implement methods to deposit, withdraw, and display the balance, ensuring direct access to the balance attribute is restricted. Explain why using private attributes can help improve data security and prevent accidental modifications.

**Ans:-** Why Use Private Attributes?

1. Encapsulation- Prevents direct modification of sensitive attributes.
2. Data Security- Ensures controlled access via methods, reducing risks of unintended changes.
3. Prevents Accidental Modification- Direct access to \_\_balance is restricted, avoiding unintended overwrites.



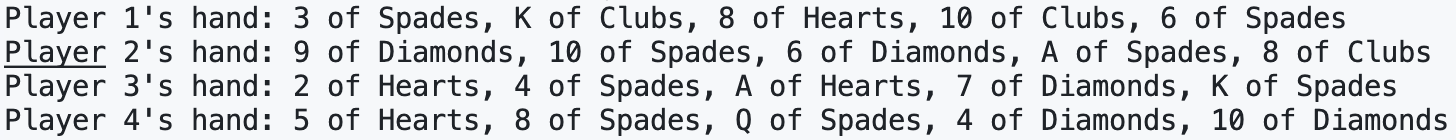
Output:- 

1. Write a Python program to simulate a card game using object-oriented principles. The program should include a Card class to represent individual playing cards, a Deck class to represent a deck of cards, and a Player class to represent players receiving cards. Implement a shuffle method in the Deck class to shuffle the cards and a deal method to distribute cards to players. Display each player’s hand after dealing.

**Ans:-**



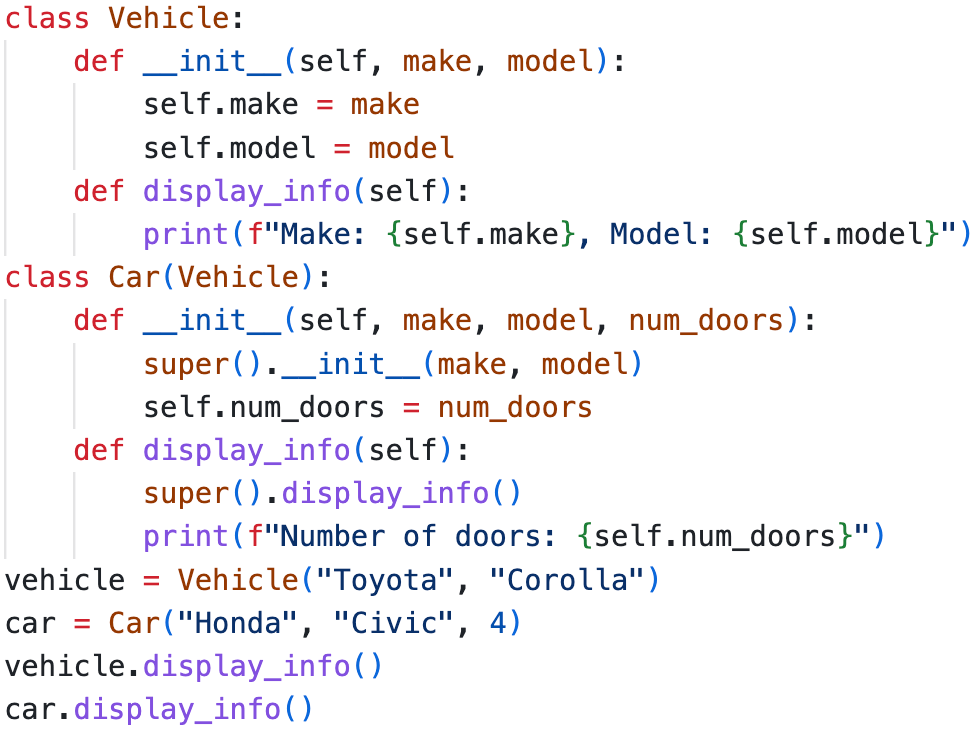
Output:-

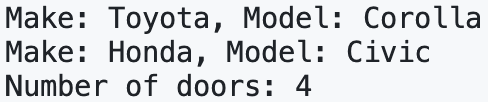


1. Write a Python program that defines a base class Vehicle with attributes make and model, and a method display info(). Create a subclass Car that inherits from Vehicle and adds an additional at- tribute num doors. Instantiate both Vehicle and Car objects, call their display info() methods, and explain how the subclass inherits and extends the functionality of the base class.

**Ans:-** How Inheritance Works Here:

1. The Car subclass inherits attributes (make, model) and methods (display\_info()) from the Vehicle base class.
2. The super().\_\_init\_\_() call allows Car to initialize inherited attributes from Vehicle.
3. The display\_info() method in Car extends the base class method by adding num\_doors.

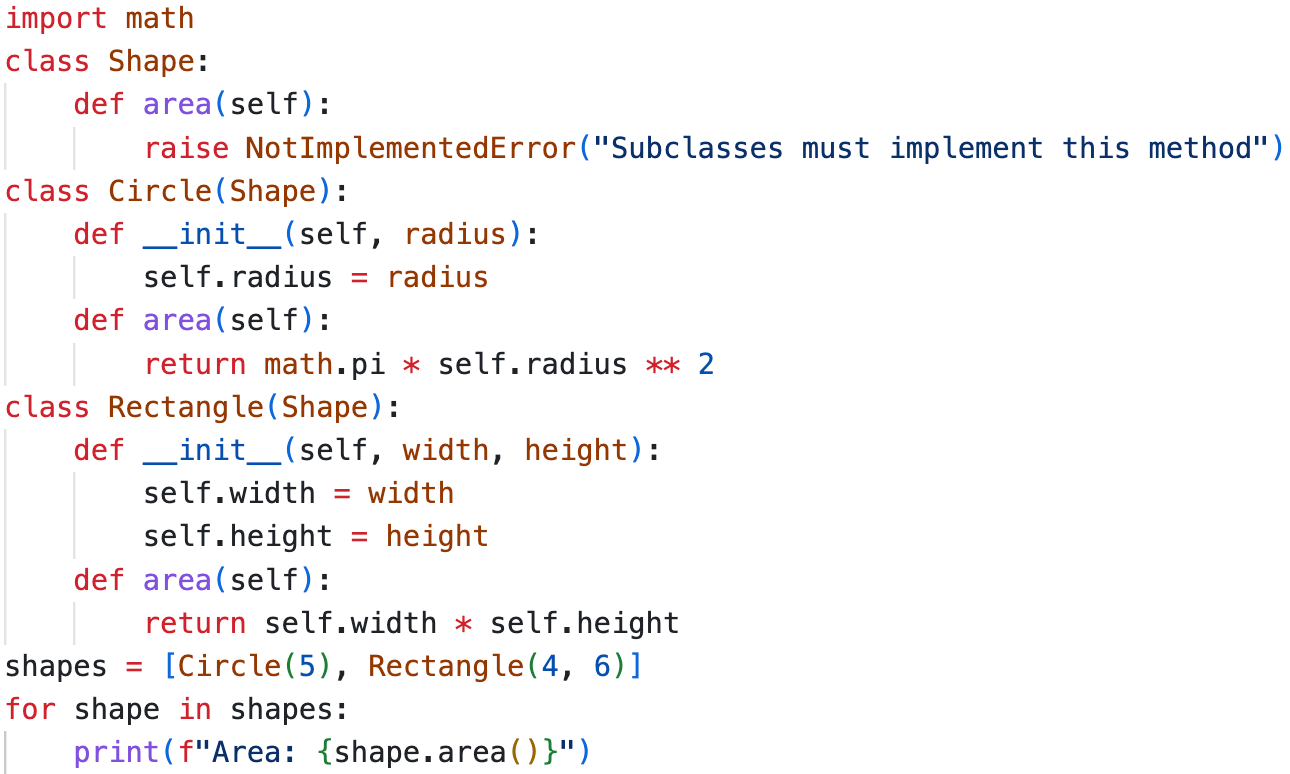


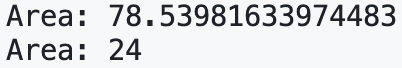
Output:- 

1. Write a Python program demonstrating polymorphism by creating a base class Shape with a method area(), and two subclasses Circle and Rectangle that override the area() method. Instantiate objects of both subclasses and call the area() method. Explain how polymorphism simplifies working with different shapes in an inheritance hierarchy.

**Ans:-** How Polymorphism Works Here:

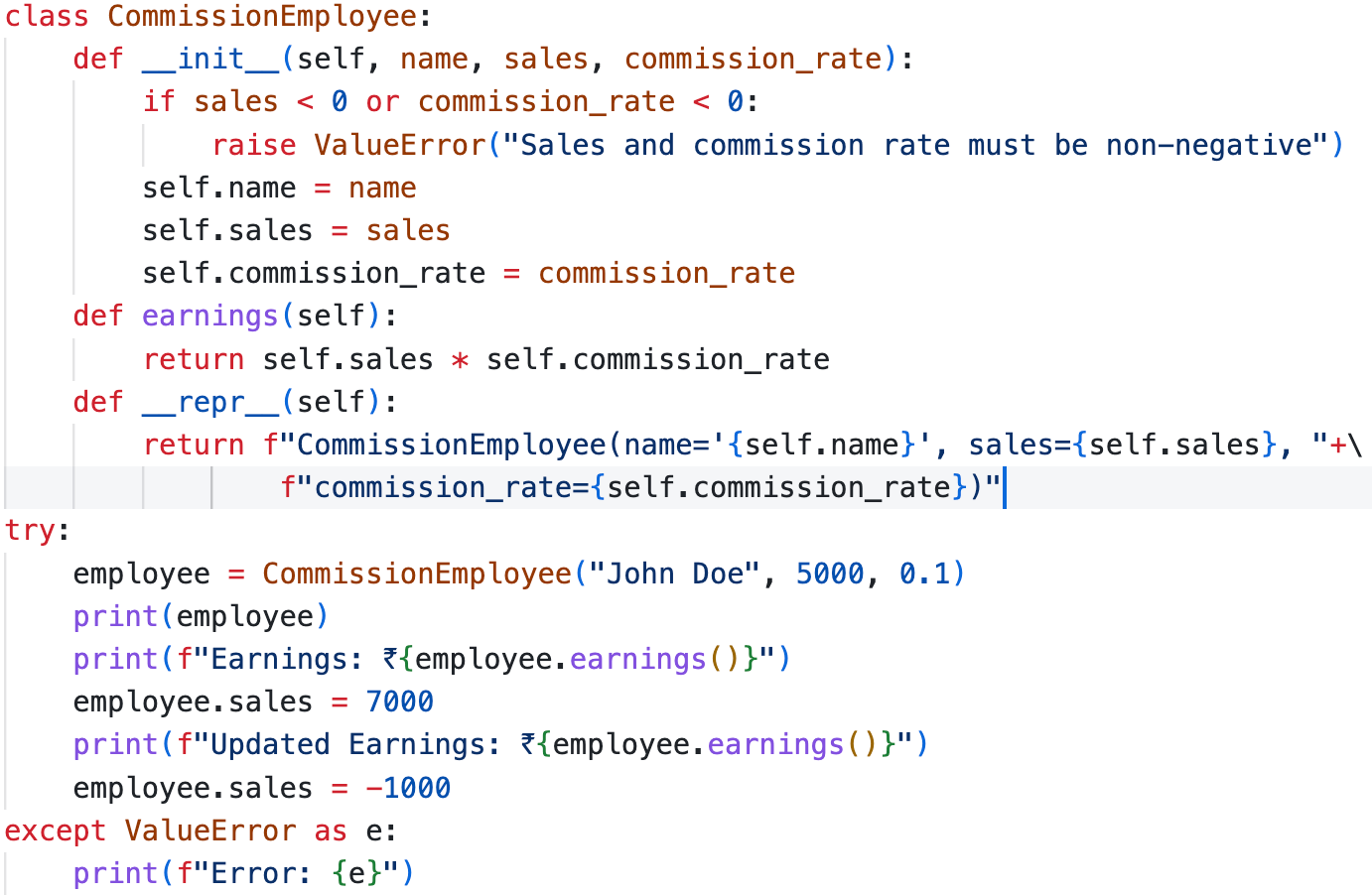
1. The Shape base class defines a common interface (area()).
2. Circle and Rectangle override area() to provide specific implementations.
3. Using polymorphism, both objects can be processed through the same loop, simplifying handling different shapes dynamically.

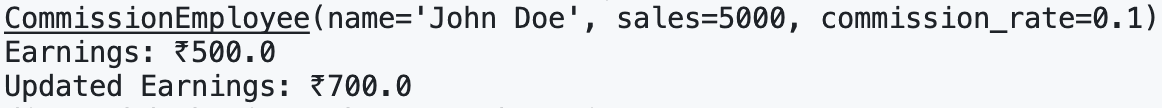


Output:- 

1. Implement the CommissionEmployee class with init, earnings, and repr methods. Include properties for personal details and sales data. Create a test script to instantiate the object, display earnings, modify sales data, and handle data validation errors for negative values.

**Ans:-**

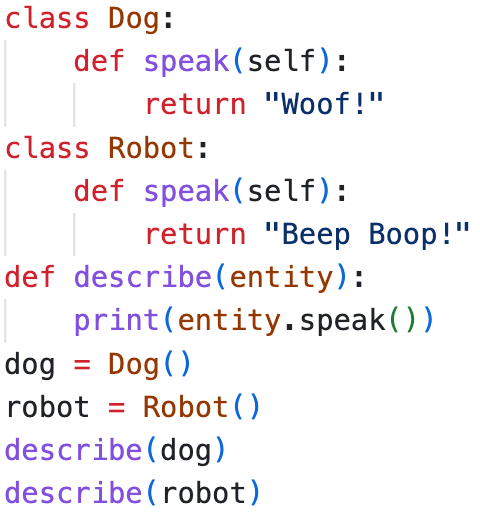
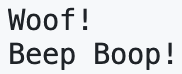


Output:- 

1. What is duck typing in Python? Write a Python program demonstrating duck typing by creating a function describe() that accepts any object with a speak() method. Implement two classes, Dog and Robot, each with a speak() method. Pass instances of both classes to the describe() function and explain how duck typing allows the function to work without checking the object’s type.

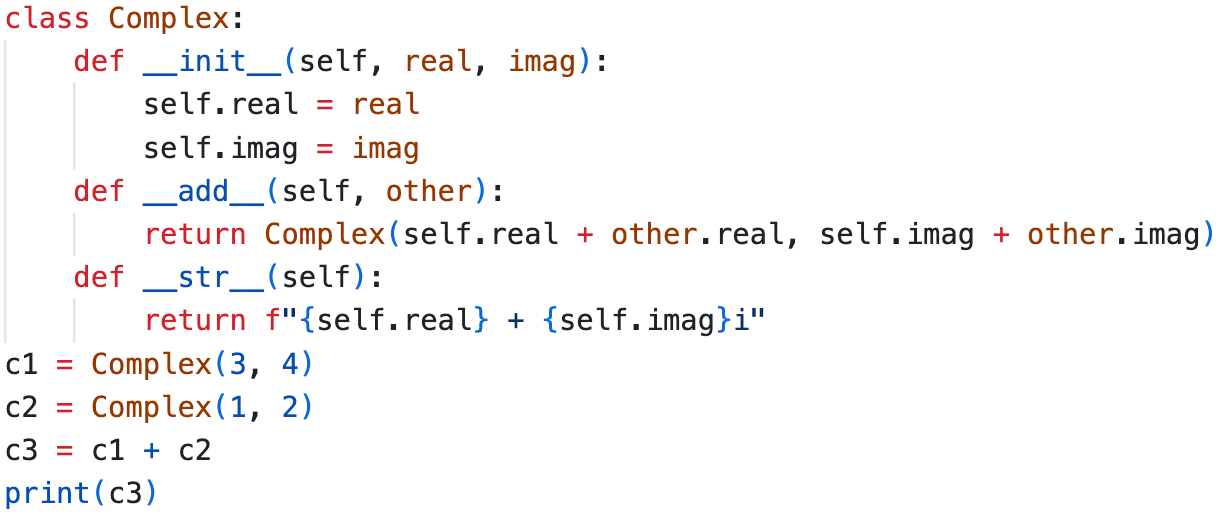
**Ans:-** How Duck Typing Works Here:

1. The describe() function does not check the type of the object.
2. As long as the object has a speak() method, it works.
3. Both Dog and Robot have speak(), so they are compatible.
4. Python’s duck typing allows flexibility without enforcing explicit type checks.

 Output:- 

1. WAP to overload the + operator to perform addition of two complex numbers using a custom Complex class?

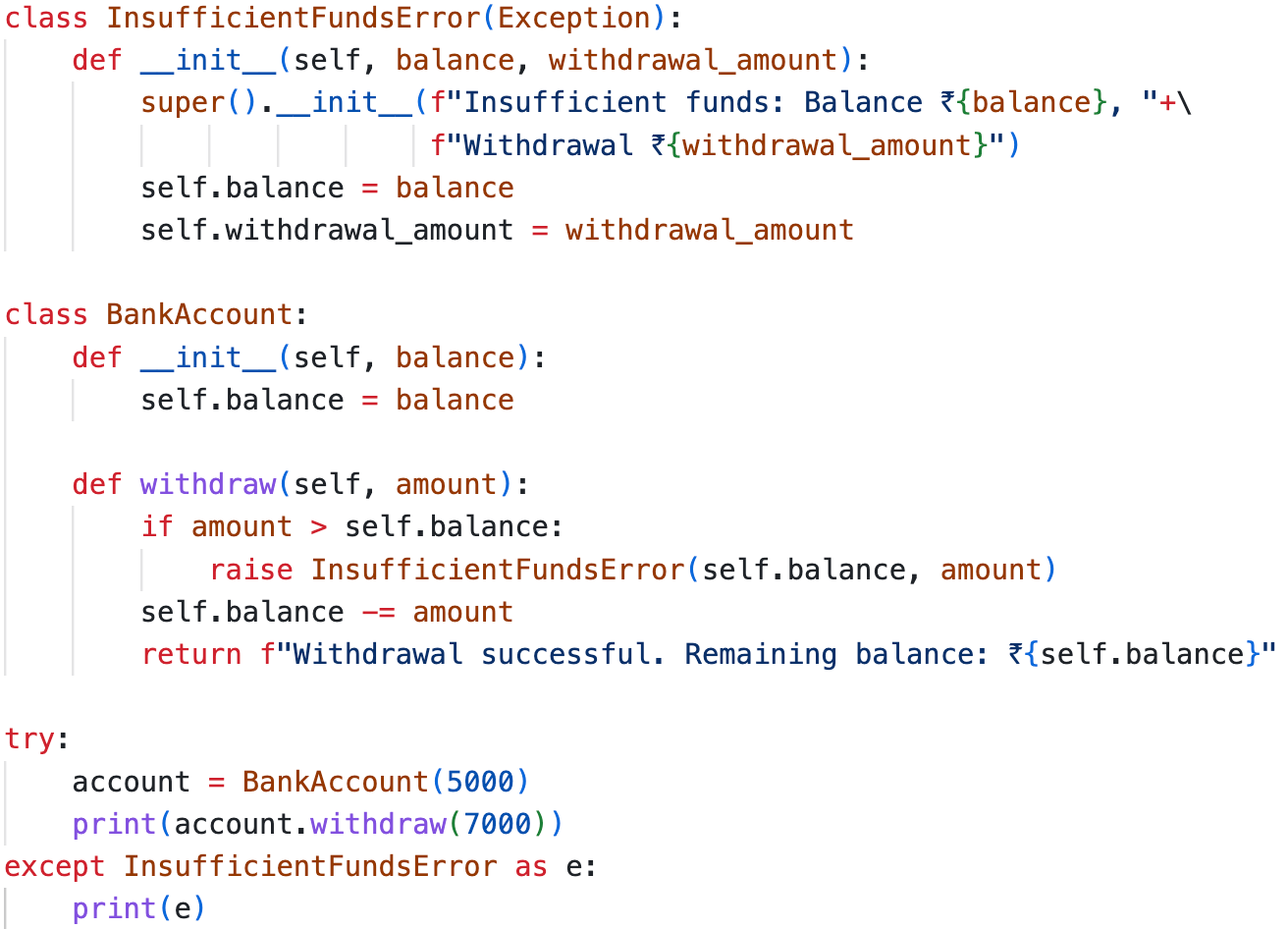
**Ans:-**



Output:- Screenshot 2025-02-10 at 10.16.21 PM

1. WAP to create a custom exception class in Python that displays the balance and withdrawal amount when an error occurs due to insufficient funds?

**Ans:-**



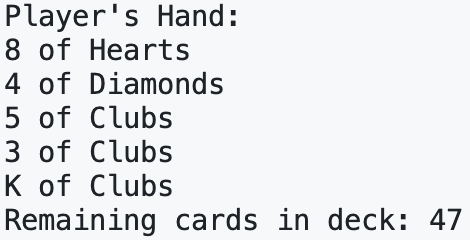
Output:- Screenshot 2025-02-10 at 10.20.58 PM

1. Write a Python program using the Card data class to simulate dealing 5 cards to a player from a shuffled deck of standard playing cards. The program should print the player’s hand and the number of remaining cards in the deck after the deal.

**Ans:-**



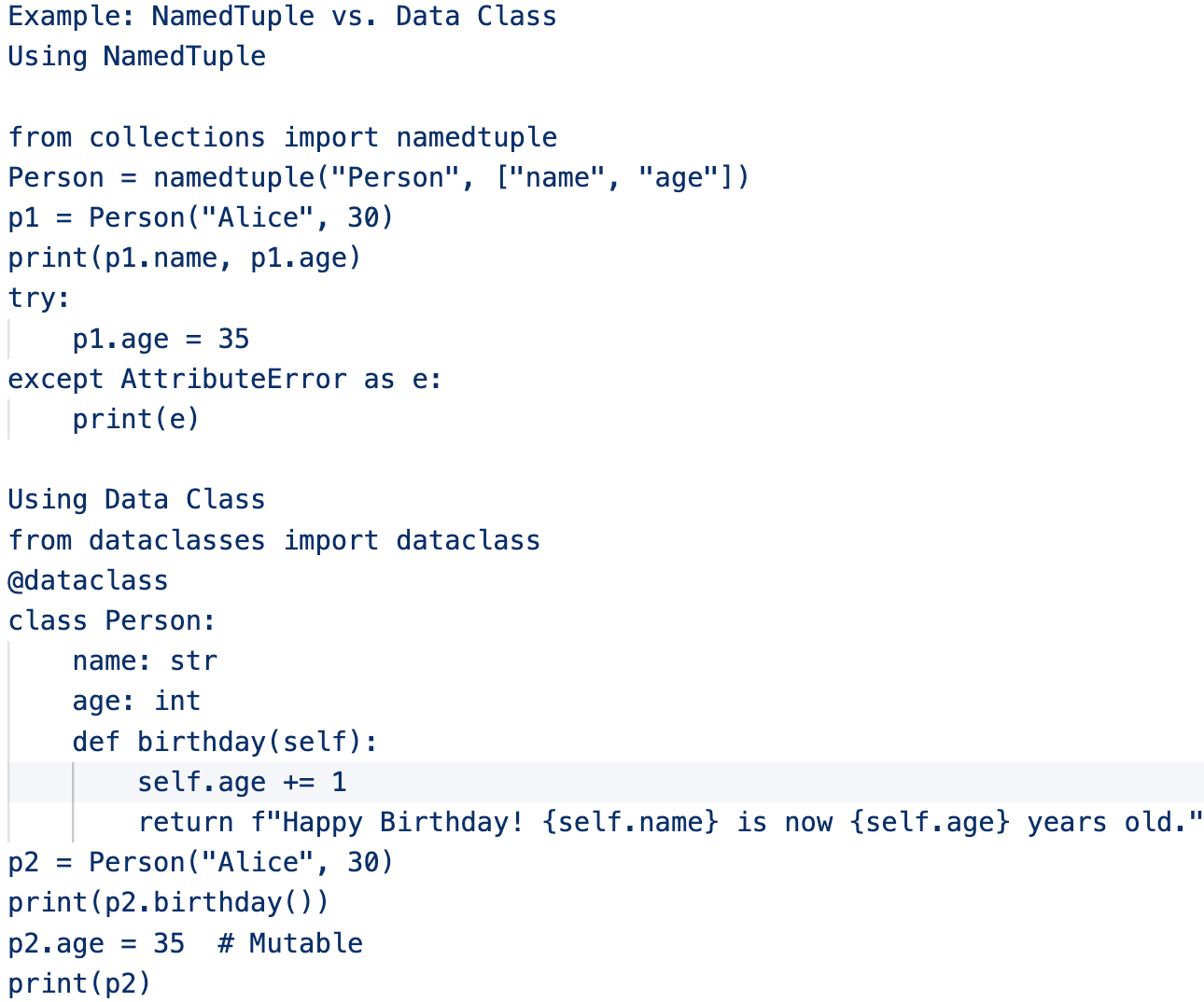


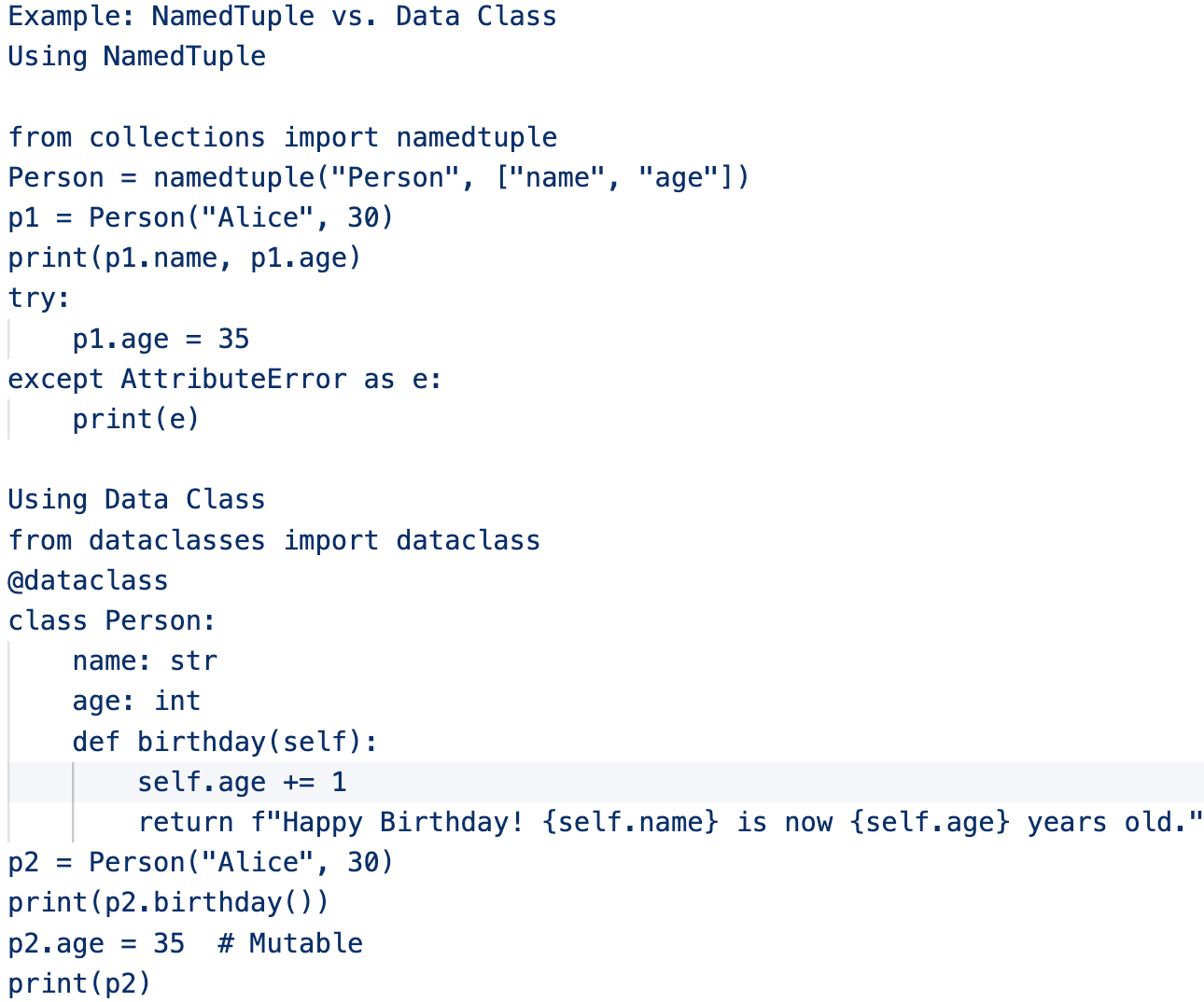
Output:- 

1. How do Python data classes provide advantages over named tuples in terms of flexibility and functionality? Give an example using python code.

**Ans:-** Advantages of Data Classes Over Named Tuples

| **Feature** | **Named Tuple** | **Data Class** |
| --- | --- | --- |
| Immutability | Immutable by default | Mutable by default |
| Methods | Cannot have methods | Can define methods |
| Inheritance | Limited support | Fully supports inheritance |
| Default Values | Requires \_replace() | Supports default values |
| Type Annotations | Optional | Strongly supported |
| Flexibility | Less flexible | More flexible with additional functionality |



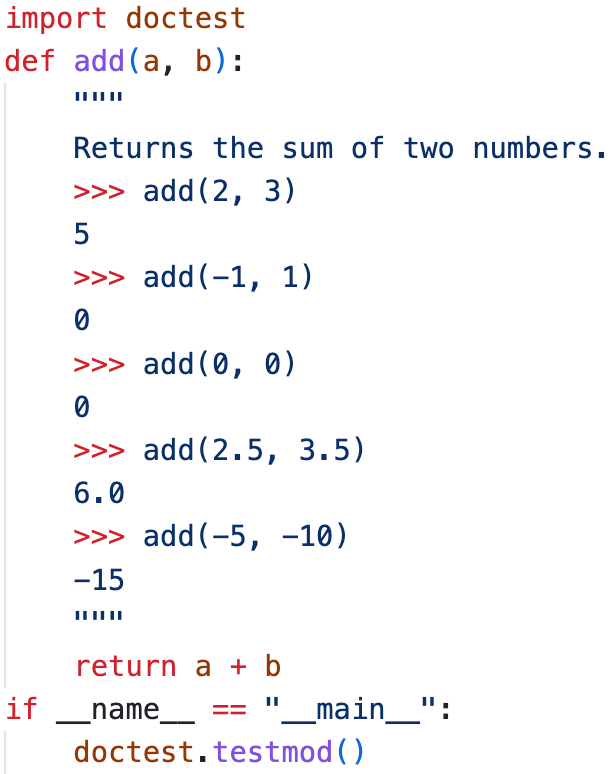


Key Takeaways:

1. Data classes allow mutability, while named tuples are immutable.
2. Data classes support methods and custom logic, enhancing functionality.
3. Data classes allow default values, type hints, and inheritance.
4. Write a Python program that demonstrates unit testing directly within a function’s docstring using the doctest module. Create a function add(a, b) that returns the sum of two numbers and includes multiple test cases in its docstring. Implement a way to automatically run the tests when the script is executed.

**Ans:-** How It Works:

1. Docstring contains test cases that specify expected outputs.
2. doctest.testmod() automatically runs all embedded test cases when executed.
3. If all tests pass, no output is shown; otherwise, failures are displayed.



1. Scope Resolution: object’s namespace → class namespace → global namespace → built-in namespace.

*species = “Global Species”*

*class Animal:*

*species = “Class Species”*

*def \_\_init\_\_ (self, species):*

*self.species = species*

*def display\_species(self):*

*print(“Instance species:’, self.species)*

*print(“Class species:”, Animal.species)*

*print(“Global species:”, globals()[‘species’])*

*a = Animal(“Instance Species”)*

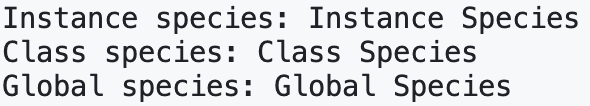
*a.display\_species()*

What will be the output when the above program is executed? Explain the scope resolution process step by step.

**Ans:-** Scope Resolution Process:

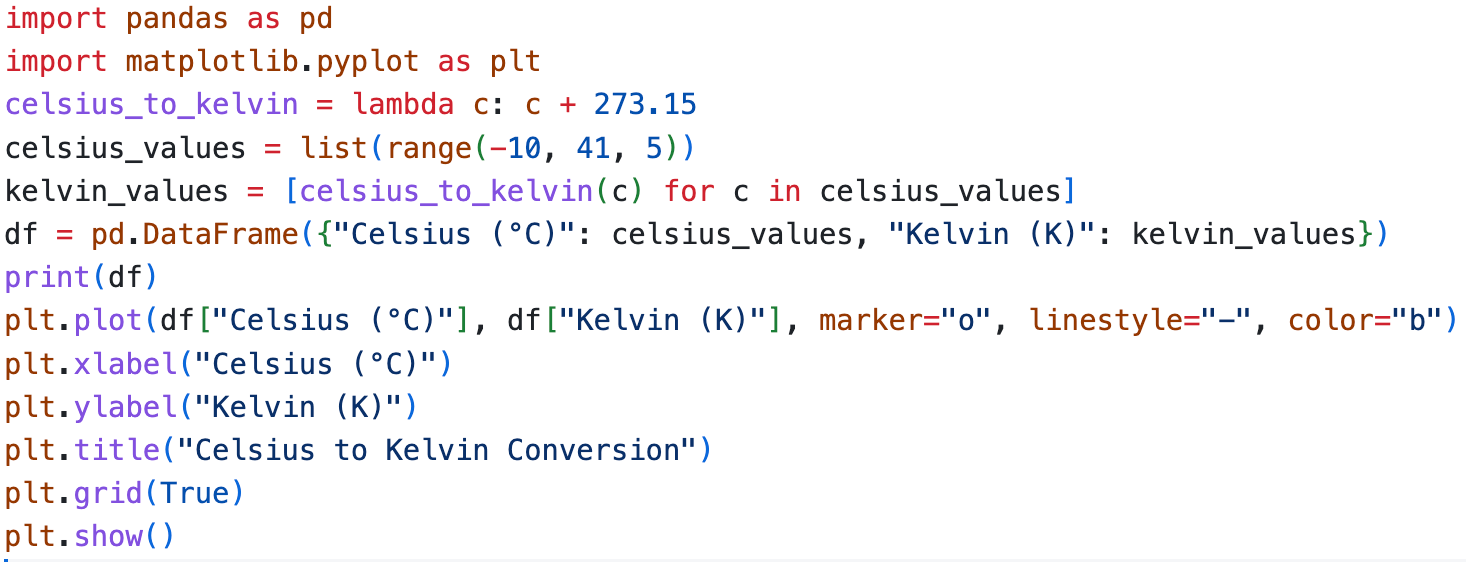
Python follows a LEGB (Local → Enclosing → Global → Built-in) resolution order:

1. Instance Namespace (self.species): self.species = "Instance Species" sets the instance-specific species attribute. When self.species is accessed, Python first checks the object's namespace and finds "Instance Species".
2. Class Namespace (Animal.species): Animal.species = "Class Species" is a class-level attribute. If self.species did not exist, it would fall back to Animal.species. Animal.species is directly accessed in display\_species(), printing "Class Species".
3. Global Namespace (species): species = "Global Species" is a global variable. globals()['species'] retrieves it explicitly, printing "Global Species".

Output:- 

1. Write a Python program using a lambda function to convert temperatures from Celsius to Kelvin, store the data in a tabular format using pandas, and visualize the data using a plot.

**Ans:-**



Output:-

