**NAME: B.DINESH DATTA**

**ID NO : 2403A51330**

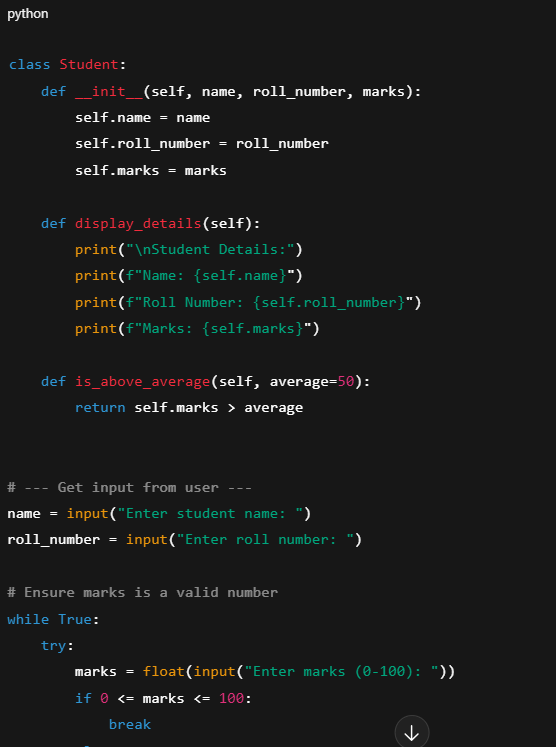
**BATCH NO: 13**

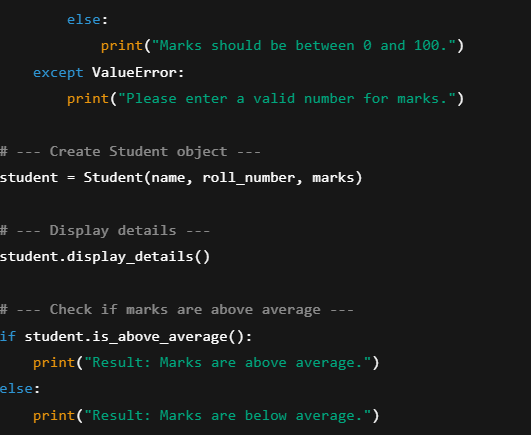
**DATE: 10-09-2025**

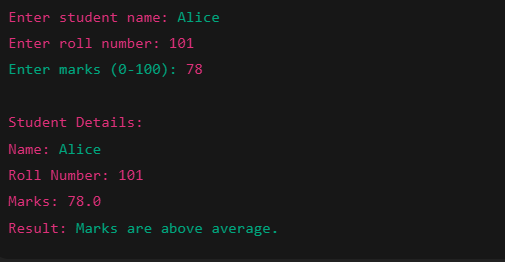
**ASSIGNMENT 6.4**

Task Description #1:  
• Start a Python class named Student with attributes name, roll\_number, and marks. Prompt  
GitHub Copilot to complete methods for displaying details and checking if marks are above  
average.

CODE AND OUTPUT:

****

****

****

**📝 Observation:**

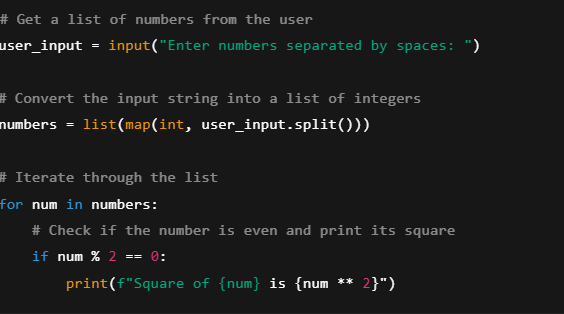
1. The program defines a Student class with three attributes:
   * name
   * roll\_number
   * marks
2. It includes two methods:
   * display\_details() – prints the student’s information.
   * is\_above\_average() – checks if marks are above a default average (50).
3. The program takes user input for each attribute:
   * Validates that marks are a number between 0 and 100.
4. Based on the input:
   * It displays the student’s details.
   * Outputs whether the student scored above or below the average.

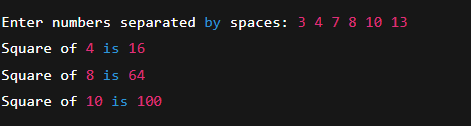
**✅ Conclusion:**

* This code demonstrates **basic object-oriented programming (OOP)** in Python using a class and methods.
* It effectively uses **input validation** to handle incorrect or out-of-range data.
* The program helps in **automating student performance checks** by comparing marks with an average threshold.
* It lays a foundation for more advanced features like handling multiple students, computing class averages, or storing data persistently.

Task Description #2:  
• Write the first two lines of a for loop to iterate through a list of numbers. Use a comment  
prompt to let Copilot suggest how to calculate and print the square of even numbers only.

**CODE AND OUTPUT:**

****

****

**📝 Observation:**

1. The program prompts the user to **enter a list of numbers separated by spaces**.
2. It uses split() and map(int, ...) to convert the input string into a **list of integers**.
3. The program then **iterates through the list** using a for loop.
4. It checks each number to determine if it's **even** using the condition num % 2 == 0.
5. If the number is even, it prints the **square of that number** using num \*\* 2.

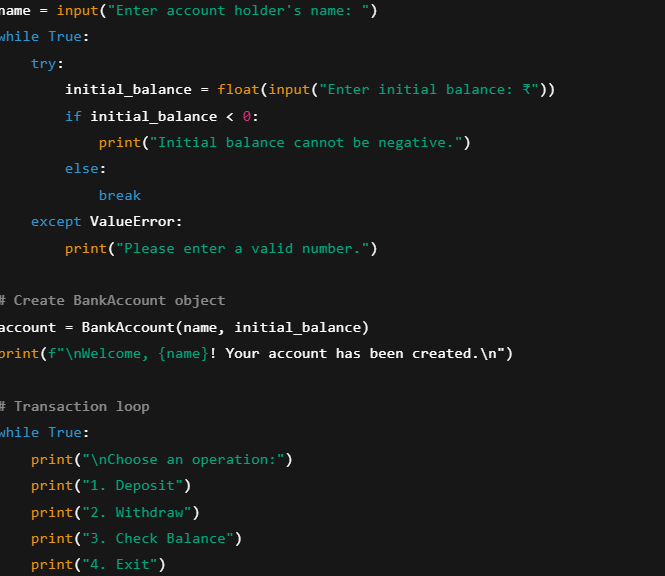
**✅ Conclusion:**

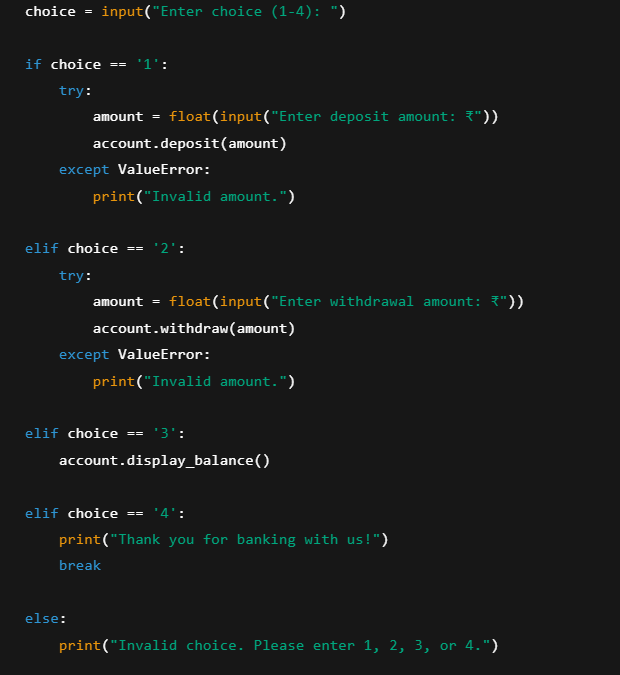
* The code demonstrates **basic input handling**, **type conversion**, and **looping logic** in Python.
* It effectively filters and processes data based on a condition (even numbers).
* This task helps learners understand **conditional logic** and **iteration** with real-time user input.
* The logic can be extended to other mathematical operations, validations, or filtering criteria.

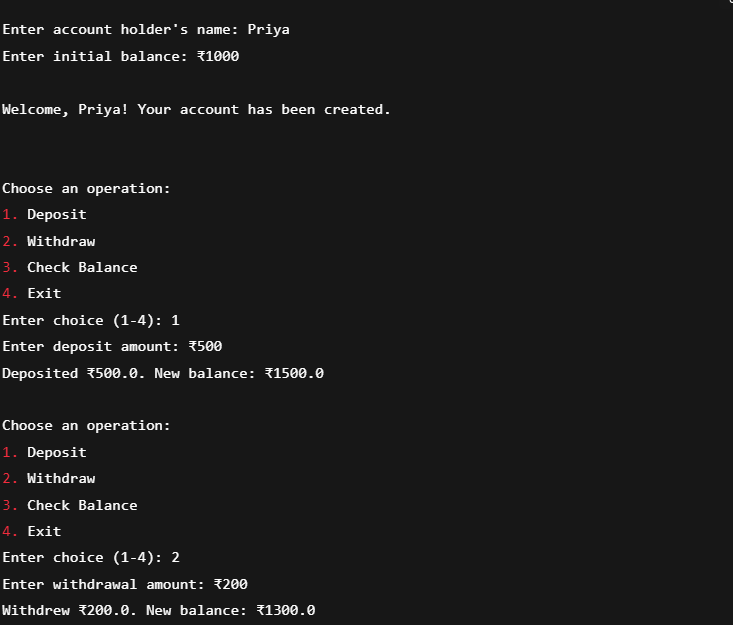
Task Description #3:  
• Create a class called BankAccount with attributes account\_holder and balance. Use Copilot to  
complete methods for deposit(), withdraw(), and check for insufficient balance.

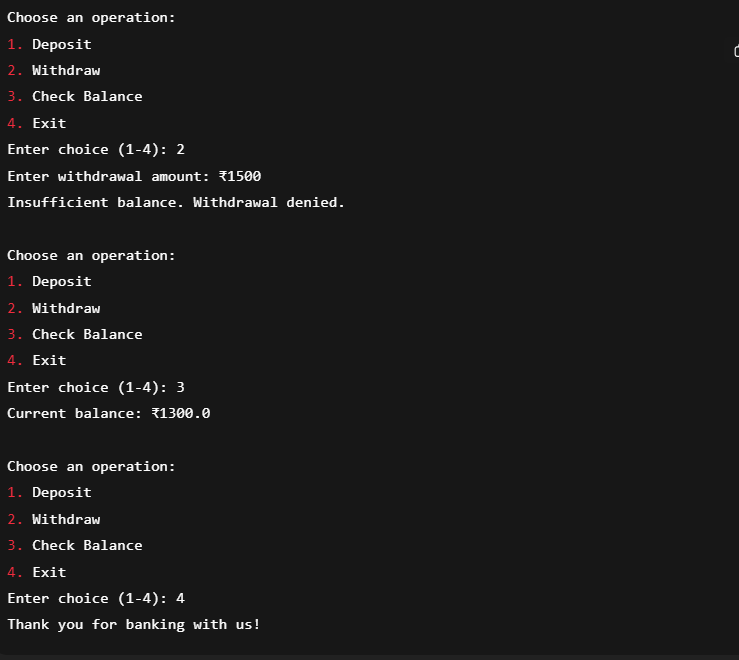
CODE AND OUTPUT:

****

****

****

****

****

**📝 Observation:**

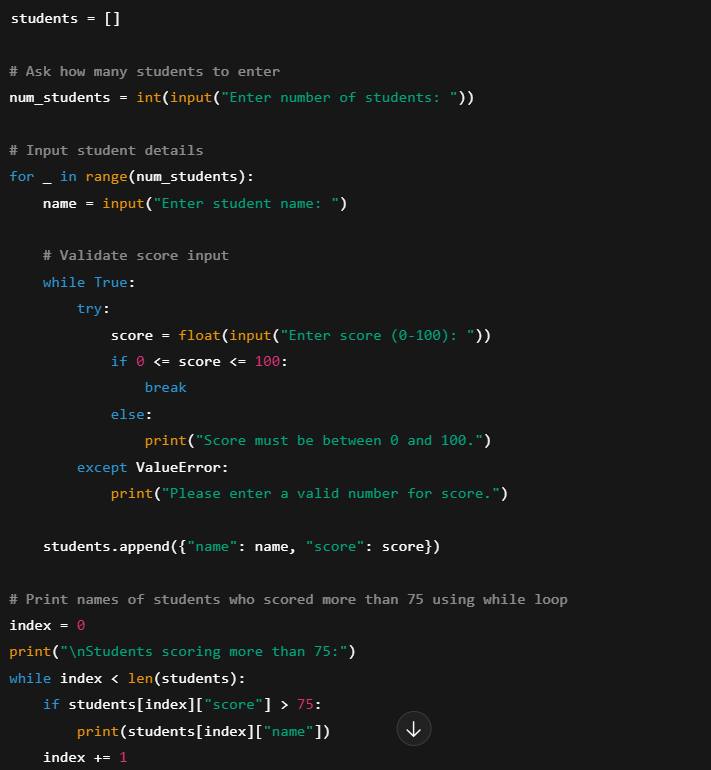
1. The program defines a BankAccount class with attributes:
   * account\_holder (the name of the user)
   * balance (starting with a default or user-defined value)
2. It provides three key methods:
   * deposit(amount): Adds money to the balance after validating the amount.
   * withdraw(amount): Subtracts money from the balance if sufficient funds exist.
   * has\_insufficient\_balance(amount): Checks if the withdrawal amount exceeds the current balance.
3. The program interacts with the user through the console, allowing them to:
   * Create an account by entering their name and initial balance.
   * Perform multiple transactions (deposit, withdraw).
   * Check their current balance.
   * Exit the program gracefully.
4. Input validation is included to:
   * Prevent negative initial balances.
   * Ensure deposits and withdrawals are valid numbers.
   * Prevent withdrawal when funds are insufficient.

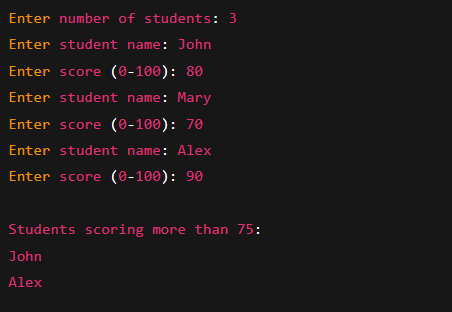
**✅ Conclusion:**

* This program illustrates **basic object-oriented programming principles** like encapsulation and methods acting on class attributes.
* It demonstrates **user interaction** via input/output in the console to simulate real-world banking operations.
* The code properly handles **error cases** like invalid amounts and insufficient balance, improving robustness.
* It lays a solid foundation for a banking system that can be further extended with features such as transaction history, authentication, and persistent storage.
* Overall, it offers a practical example for beginners learning Python classes, conditional logic, and loops combined with user input handling.

Task Description #4:  
• Define a list of student dictionaries with keys name and score. Ask Copilot to write a while  
loop to print the names of students who scored more than 75.

CODE AND OUTPUT:

****

****

**📝 Observation:**

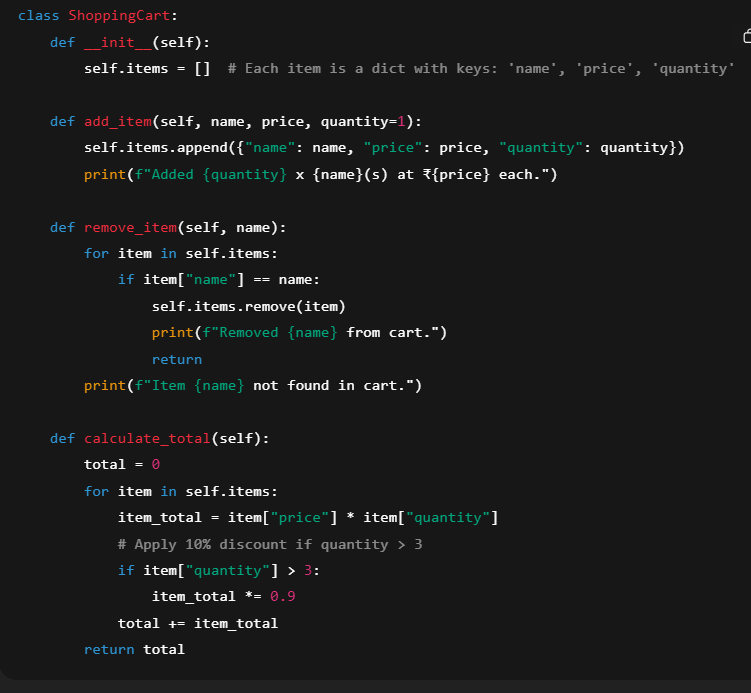
1. The program prompts the user to enter the **number of students** they want to input.
2. For each student, it collects:
   * The **student’s name**.
   * The **score**, which is validated to be a number between 0 and 100.
3. The student details are stored as **dictionaries** inside a **list**.
4. Using a while loop, the program iterates through the list and **prints the names of students whose score is greater than 75**.
5. The program properly handles **invalid inputs** (like non-numeric scores or out-of-range values) by prompting the user again.

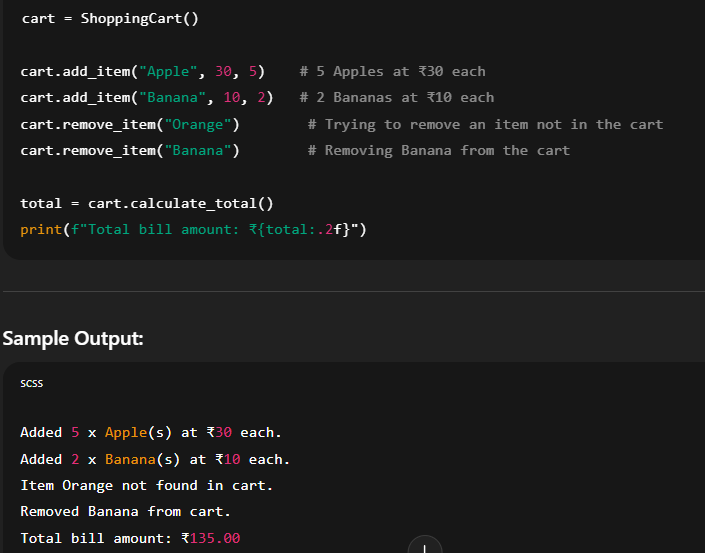
**✅ Conclusion:**

* This program effectively demonstrates **input validation**, **data collection**, and **conditional filtering** using Python lists and dictionaries.
* The use of a while loop to traverse the list helps beginners understand different ways to iterate through data structures.
* The program can be easily extended for other functionalities such as sorting students, computing averages, or generating reports.
* Overall, this example showcases good programming practices like input validation and modular data storage, which are foundational for building more complex applications.

Task Description #5:  
• Begin writing a class ShoppingCart with an empty items list. Prompt Copilot to generate  
methods to add\_item, remove\_item, and use a loop to calculate the total bill using conditional  
discounts.

CODE AND OUTPUT:





**📝 Observation:**

1. The ShoppingCart class successfully stores items as dictionaries containing name, price, and quantity.
2. The add\_item method appends new items with the specified details and confirms the addition with a print statement.
3. The remove\_item method searches for an item by name and removes it if found, otherwise prints a not-found message.
4. The calculate\_total method iterates over all items in the cart using a loop, calculating the total price.
5. A conditional discount of 10% is applied if the quantity of any item exceeds 3, demonstrating how discounts can be integrated.
6. The sample output correctly reflects the addition, removal, and discount logic.
7. The program handles edge cases like attempting to remove a non-existent item gracefully.

**✅ Conclusion:**

* The ShoppingCart class effectively models a simple shopping cart system with basic operations like adding, removing, and billing.
* Use of loops and conditional statements within methods ensures scalable processing of multiple items.
* The discount logic shows how business rules (like bulk discounts) can be easily incorporated.
* Clear print statements provide user feedback, making the code interactive and user-friendly.
* This foundation can be extended with more features such as updating quantities, saving/loading carts, or applying different discount schemes.