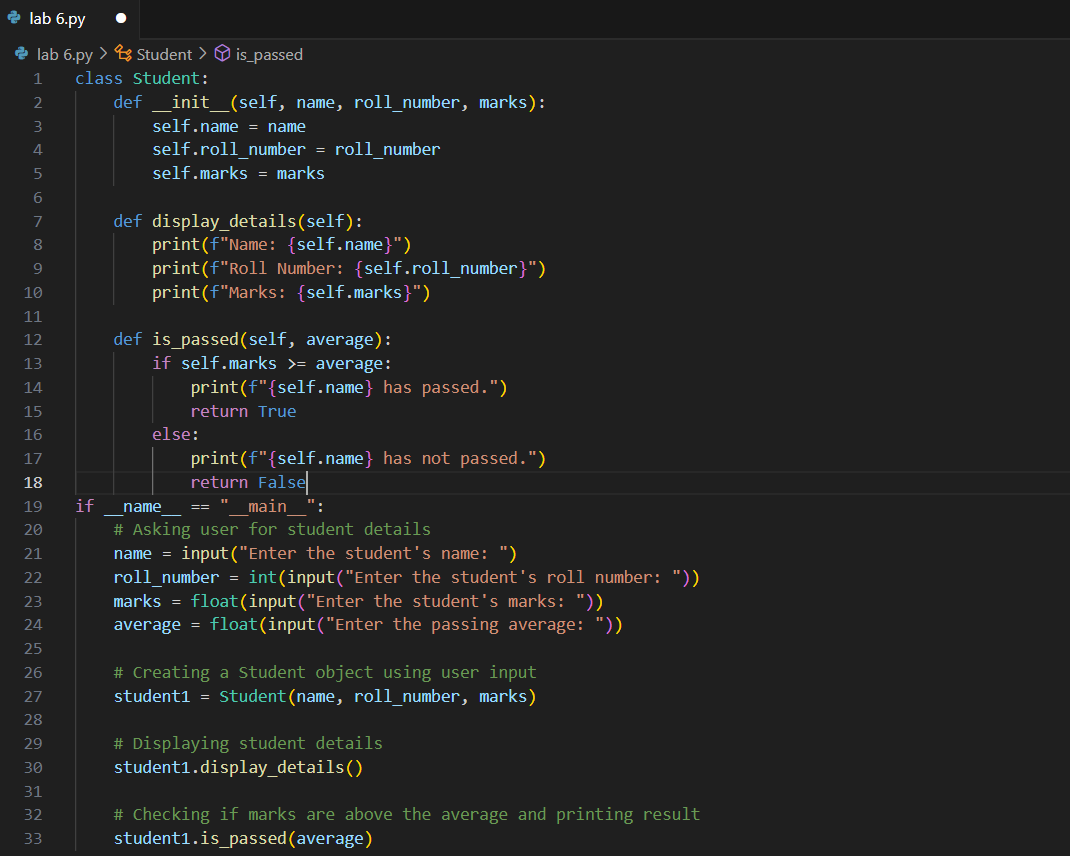
# LAB ASSIGNMENT-6

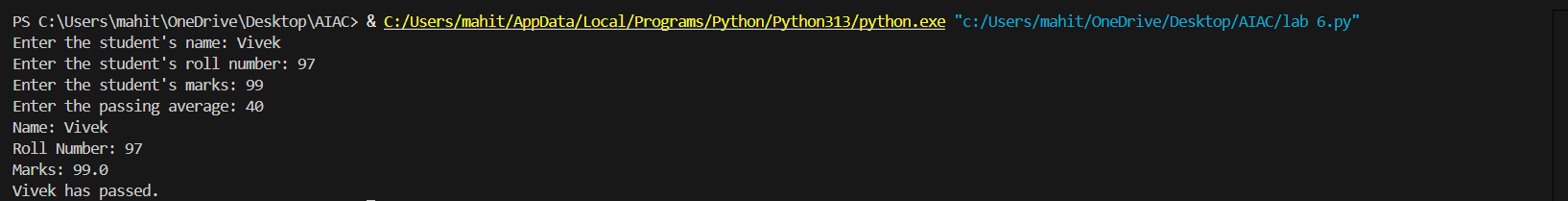
TASK-1:

Prompt:

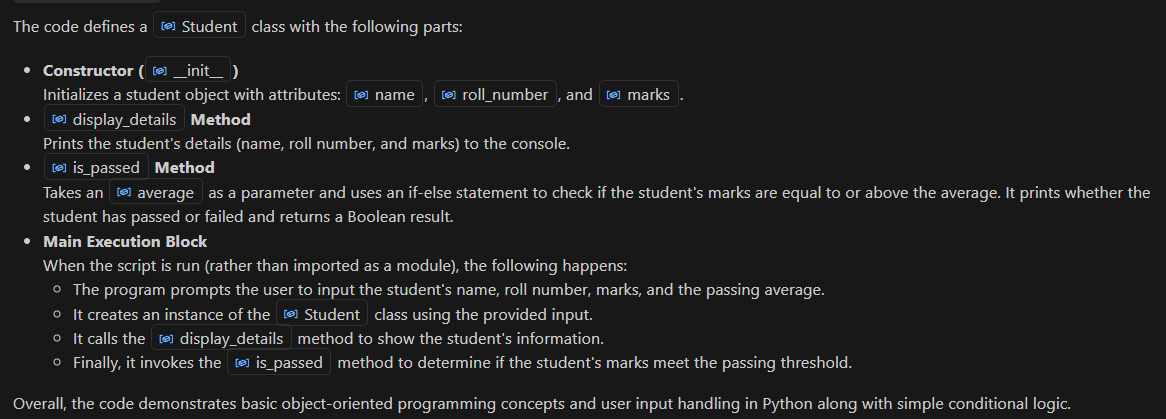
Generate a python code to ask input to user using a Class named Student with attributes name, roll\_number, and marks to complete methods for displaying details and checking if marks are above average. Use classes like display\_details() and is\_passed(), demonstrating use of if-else conditions.

Code and Output:





Code Explanation:



TASK-2:

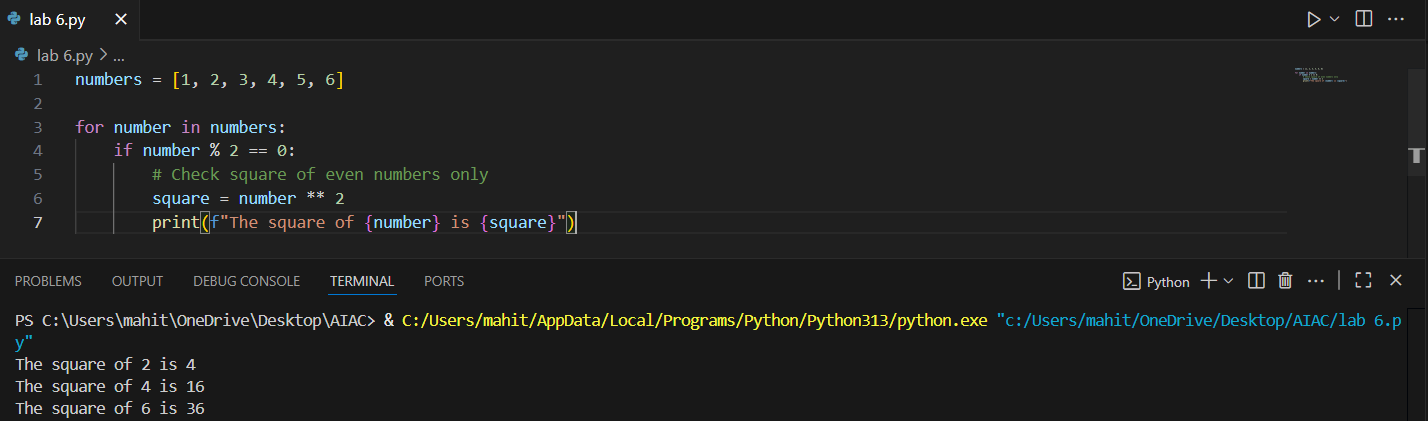
Prompt:

for number in numbers:

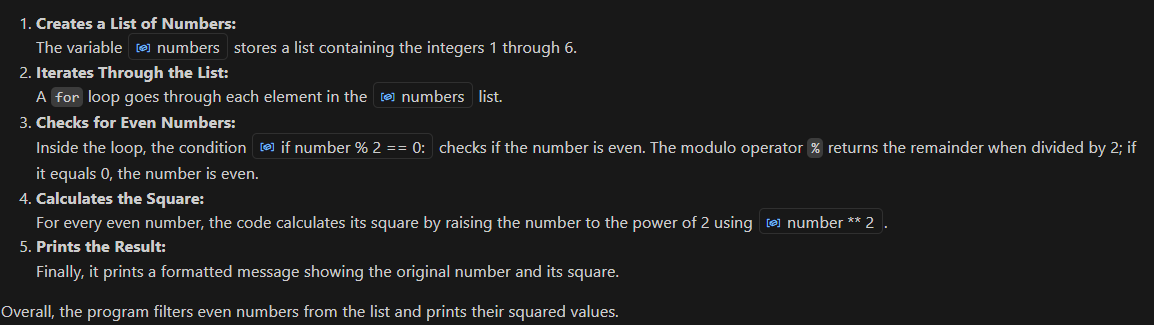
if number % 2 == 0:

#check square of even numbers only

Code and Output:



Code Explanation:



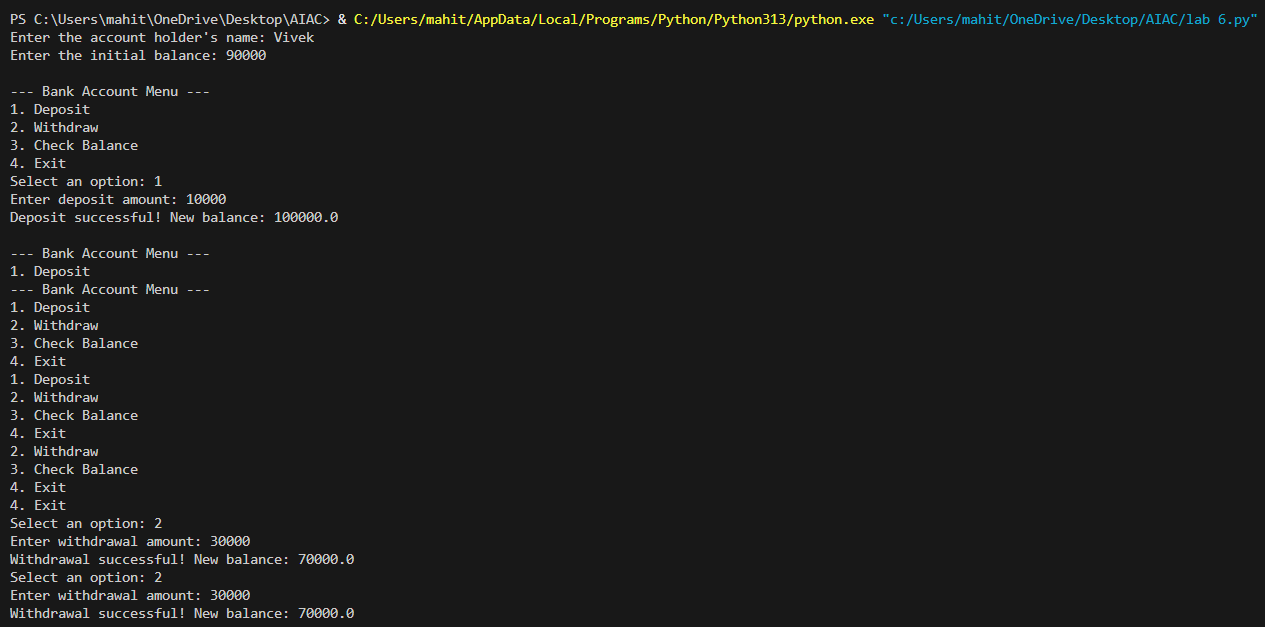
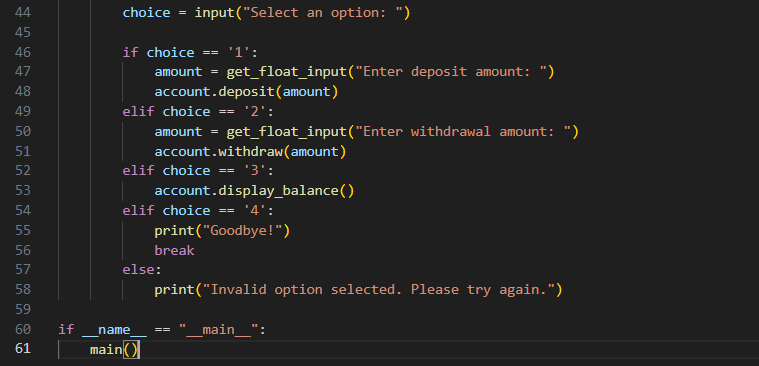
TASK-3:

Prompt:

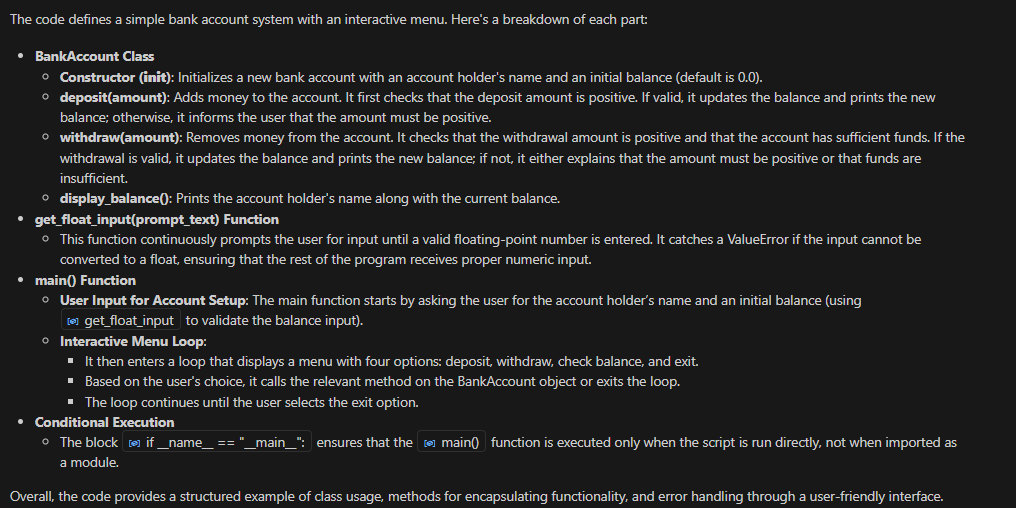
Create a class called BankAccount with attributes account\_holder and balance. Give a python code to ask input to user also complete methods for deposit(), withdraw(), and check for insufficient balance.

Code and Output:





Code Explanation:



TASK-4:

Prompt:

students = [

{"name": "Tony Stark", "score": 97},

{"name": "Steve Rogers", "score": 88},

{"name": "Thor", "score": 82},

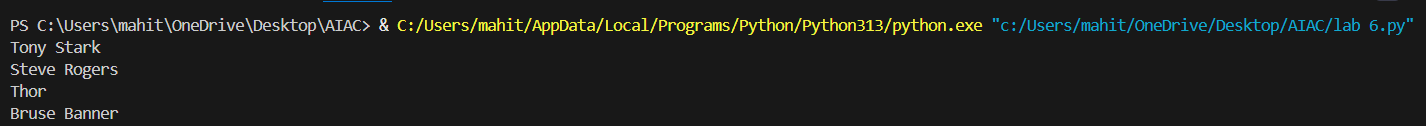
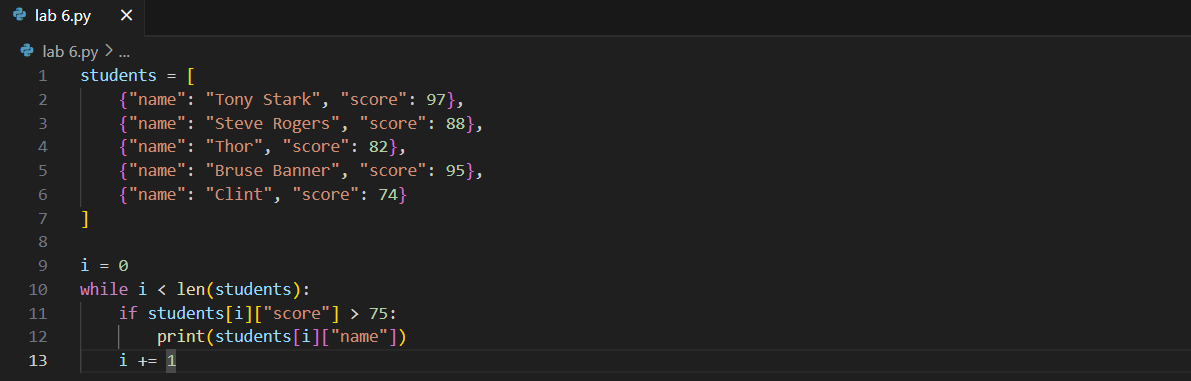
{"name": "Bruse Banner", "score": 95},

{"name": "Clint", "score": 74}

]

Here, is a list of student dictionaries. Now, give a python code using while loop to print the names of students who scored more than 75.

Code and Output:



Code Explanation:

**🔹 Code Purpose**

* Prints the names of students who scored **more than 75**.

**🔹 Data Structure**

* students is a **list of dictionaries**.Each dictionary contains:
  + "name" – student's name and "score" – student's numeric score.

**🔹 Loop Logic**

* i = 0: initializes the counter.
* while i < len(students): loops through all students by index.
  + Checks if the student's "score" is **greater than 75**. If true, prints the student's "name".
* i += 1: moves to the next student in the list.

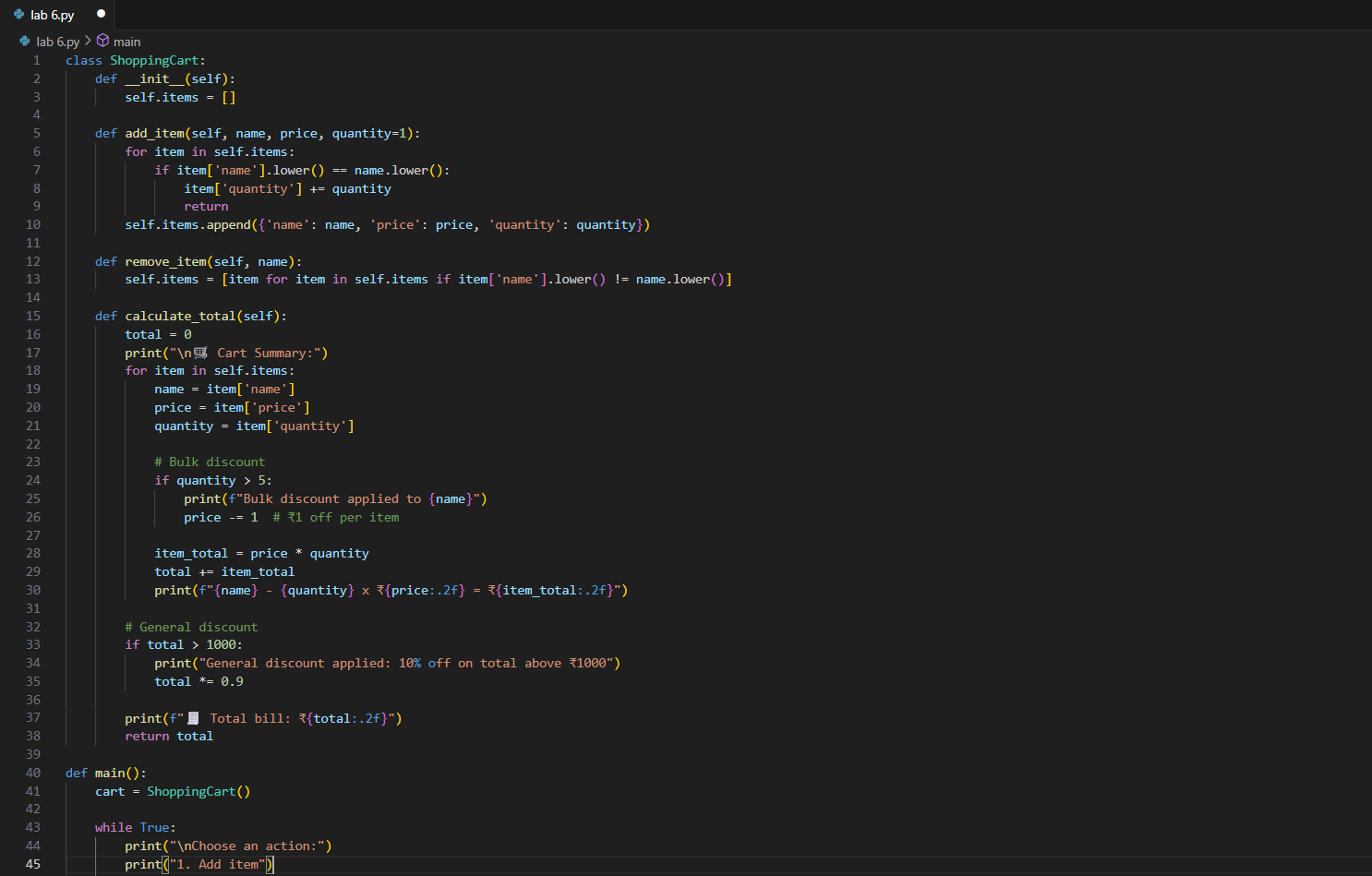
**🔹 Condition**

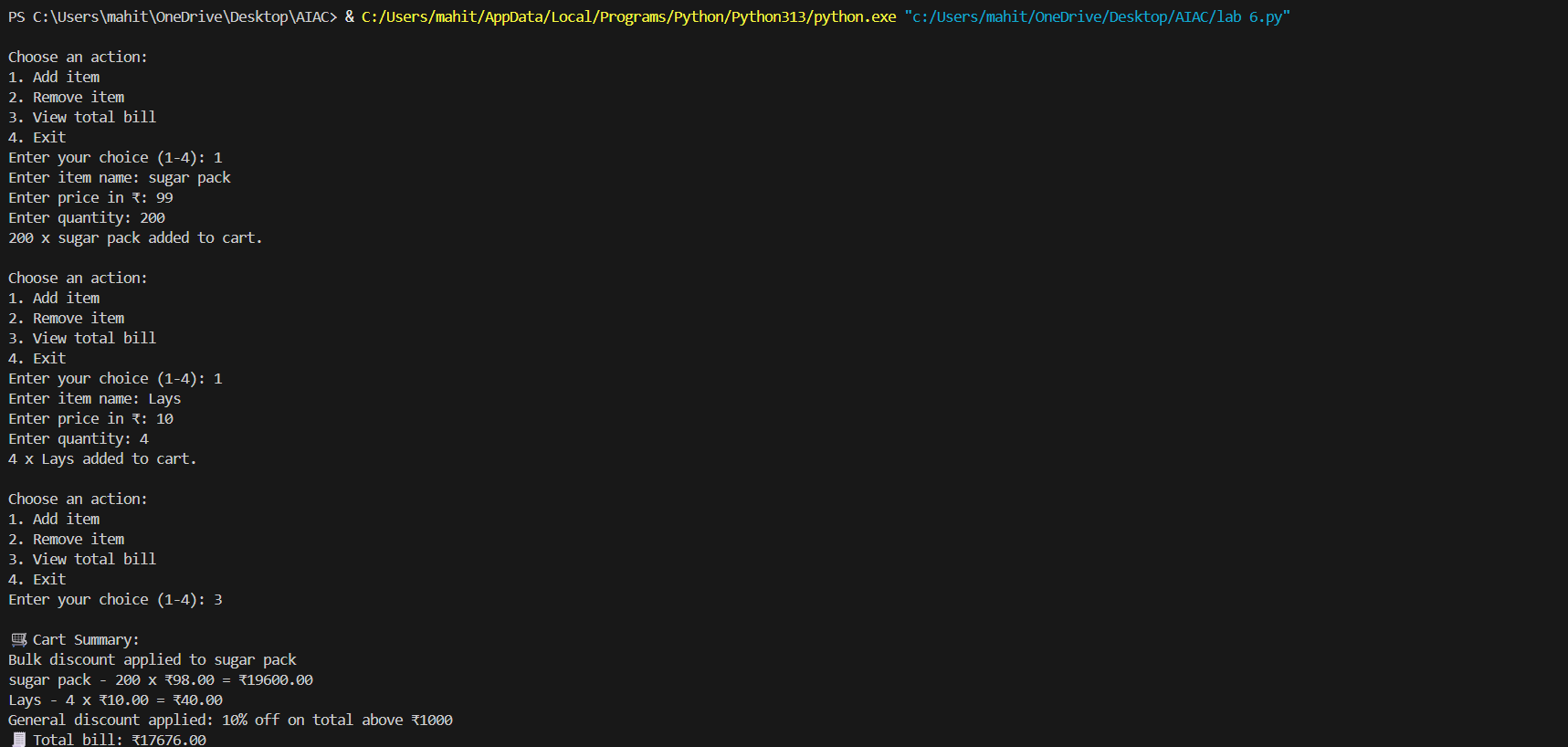
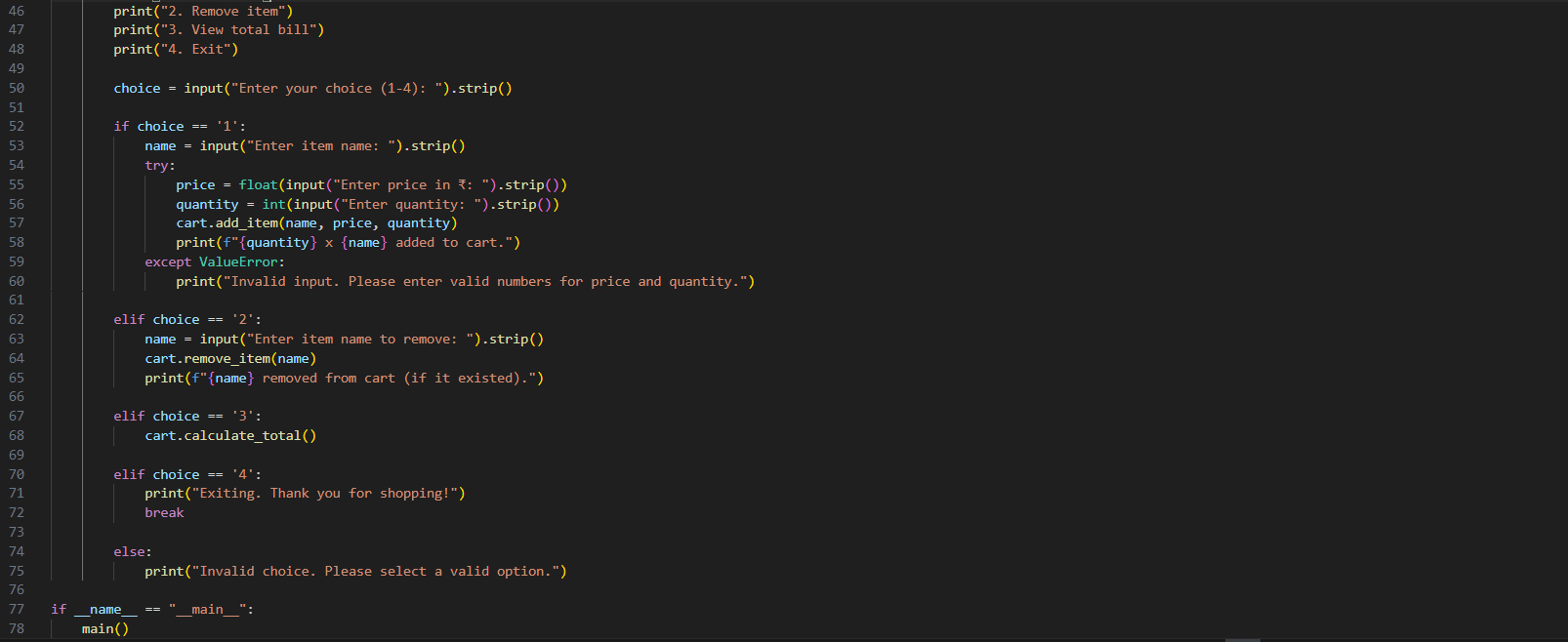
* Only students with score > 75 are printed.

TASK-5:

Prompt:

Give a python code to ask input to user also generate methods to add\_item, remove\_item, and use a loop to calculate the total bill using conditional discounts. Begin writing a class ShoppingCart with an empty items list.

Code and Output:



Code Explanation:

**ShoppingCart Class**

* **Constructor**

**(\_\_int\_\_)**  
Initializes a new shopping cart with an empty list of items (self.items = []).

* **add\_item(name, price, quantity=1)**
* **Method**  
  Adds an item to the cart.
  + If the item already exists (case-insensitive comparison), it increases its quantity instead of adding a duplicate.
  + Otherwise, it appends a new dictionary with name, price, and quantity to the list.
* **remove\_item(name)**
* **Method**  
  Removes all entries matching the given item name (case-insensitive) from the cart.
  + It uses a list comprehension to filter out items whose name matches the provided name.
* **calculate\_total()**
* **Method**  
  Calculates and displays the total bill with discounts:
  + Iterates through each item and prints details (name, quantity, price).
  + **Bulk Discount:** If quantity > 5, applies a ₹1 discount per item and displays a message.
  + Calculates item\_total for each item and adds it to total.
  + **General Discount:** If the total exceeds ₹1000, applies an additional 10% discount and displays a message.
  + Prints the final bill total and returns the amount.

**main() Function**

* **Cart**
* **Initialization:**  
  Creates a ShoppingCart object to manage items.
* **Interactive Menu**
* **Loop:**  
  Continuously displays options until the user exits:
  1. **Add Item:**
     + Prompts for name, price, and quantity.
     + Adds the item using cart.add\_item().
     + Handles invalid numeric input using try-except.
  2. **Remove Item:**
     + Prompts for an item name and removes it using cart.remove\_item().
  3. **View Total Bill:**
     + Calls cart.calculate\_total() to display a detailed summary with discounts.
  4. **Exit:**
     + Breaks out of the loop and ends the program.
  5. Handles invalid menu choices gracefully.