AI ASSISTED CODING

Assignment-6.4:

***Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals***

**2403A52107**

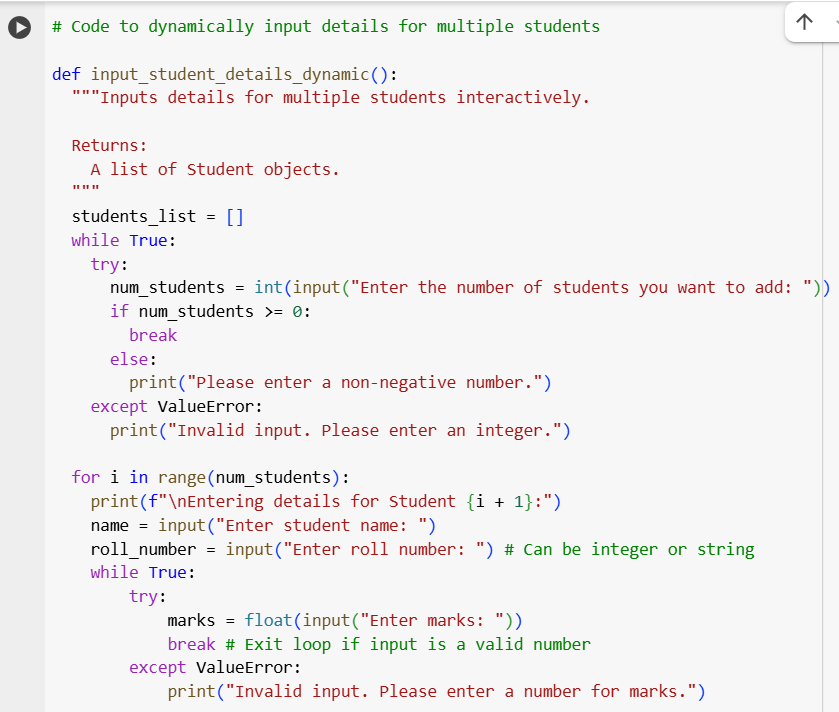
**Pavani Voddepalli**

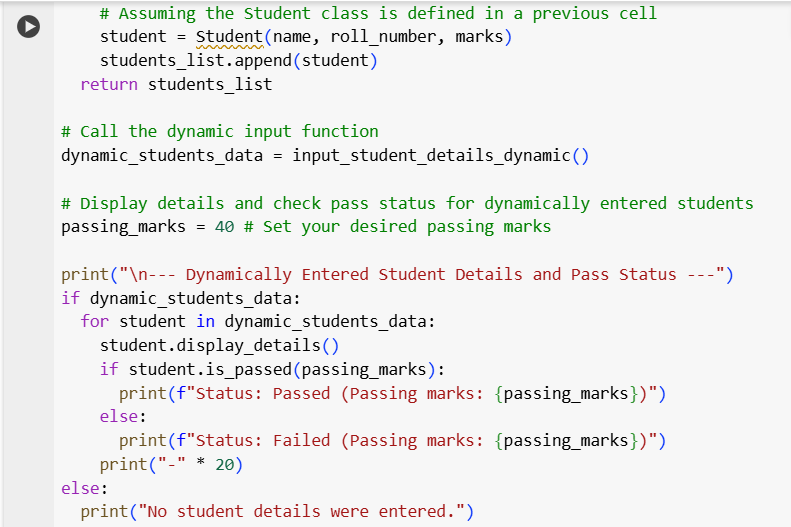
**Batch : 24BTCAIAI05**

**TASK 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.

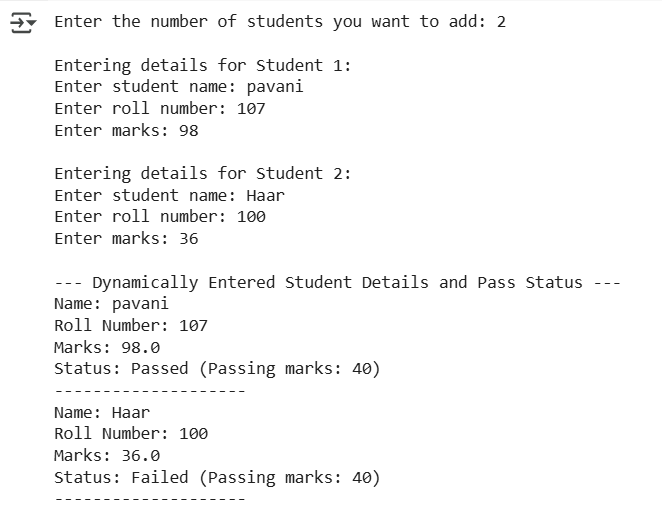
**PROMPT** : Generate a Python program dynamically using class named Student with attributes name, roll\_number, and marks with methods for displaying details and checking if marks are above average : display\_details() and is\_passed(), demonstrating use of if-else conditions .

**CODE :**





**OUTPUT :**



**EXPLANATION :**

This program allows you to **input details for students dynamically** during the execution of the code.

It first **asks you how many students** you want to add. It includes a check to make sure you enter a valid number.

Then, for the number of students you specified, it **goes through a process for each student**:

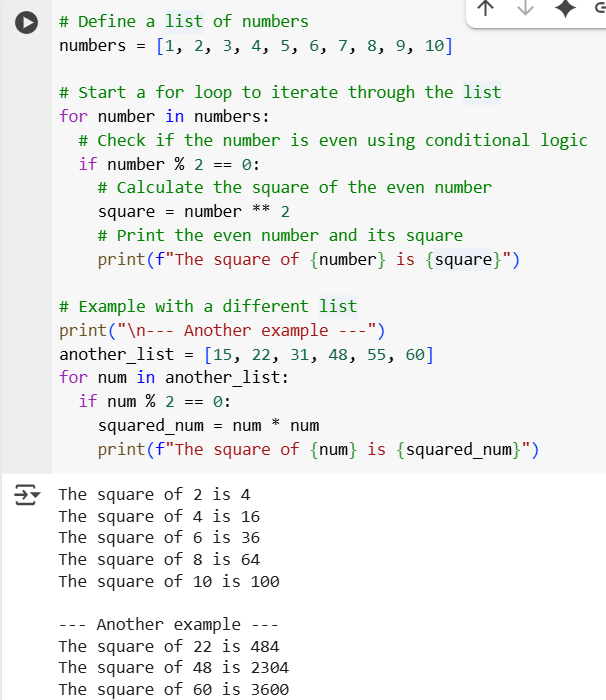
* It **prompts you to enter the student's name**.
* It **prompts you to enter the student's roll number**.
* It **prompts you to enter the student's marks**, and it will keep asking until you enter a valid number.
* Using the information you provide, it **creates a representation of that student** (a Student object).
* It **keeps track of all the students** you've entered.

After you've entered details for all the students, the program then **shows you the information for each student** you added. For each student, it also **checks if their marks are above a certain passing mark** and tells you if they passed or failed.

TASK 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.

PROMPT : Generate a Python program calculating first two lines of a for loop to iterate through a list of numbers and print the square of even numbers only , with conditional logic.

**CODE & OUTPUT :**



**EXPLANATION :**

This code program iterates through a list of numbers and prints the square of only the even numbers found in that list.

Here's a breakdown of what it does:

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

This line creates a Python list named numbers containing integers from 1 to 10. This is the list the program will process.

# Start a for loop to iterate through the list  
for number in numbers:  
  # ... rest of the loop ...

This is a for loop that begins the process of going through each item in the numbers list one by one. In each turn of the loop, the current number from the list is temporarily assigned to the variable number.

  # Check if the number is even using conditional logic  
  if number % 2 == 0:  
    # ... actions for even numbers ...

Inside the loop, this if statement checks if the current number is even. The modulo operator (%) calculates the remainder when number is divided by 2. If the remainder is 0 (number % 2 == 0), it means the number is perfectly divisible by 2, and thus it's an even number. This is the conditional logic that determines whether the following actions will be executed.

    # Calculate the square of the even number  
    square = number \*\* 2  
    # Print the even number and its square  
    print(f"The square of {number} is {square}")

These lines are executed *only if* the if condition above is true (i.e., the number is even).

* square = number \*\* 2: This calculates the square of the even number by raising it to the power of 2 (\*\* 2) and stores the result in the square variable.
* print(f"The square of {number} is {square}"): This line prints a formatted string to the console, showing the even number and its calculated square.

# Example with a different list  
# ... similar loop and logic ...

This part repeats the same process but with a different list of numbers (another\_list) to show that the function works for other inputs as well. It uses slightly different variable names (num and squared\_num), but the logic remains the same.

**TASK 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

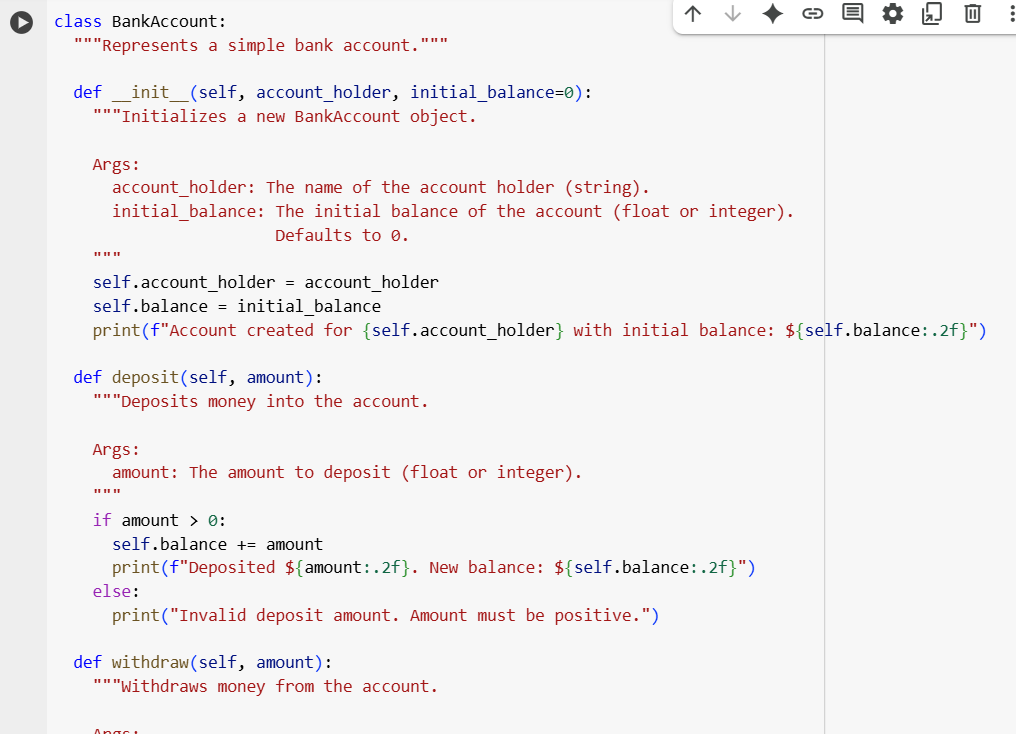
**PROMPT :** Write a python program to create a class called BankAccount with attributes account\_holder and balance.Include the following methods:

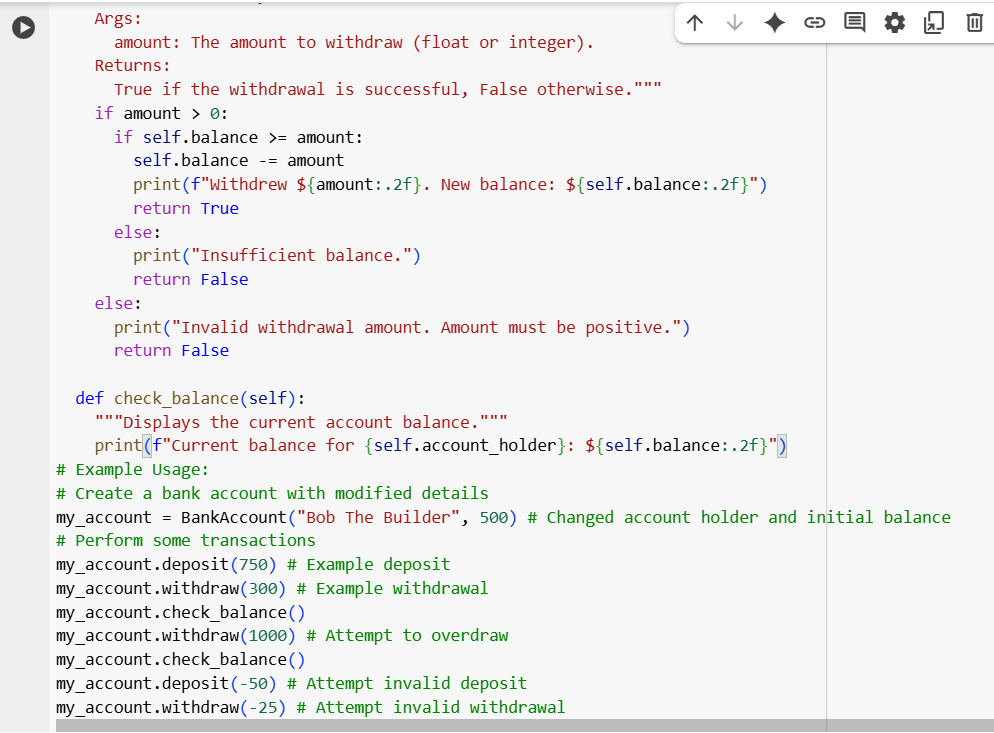
- deposit(amount): to add funds to the balance. It should only allow positive amounts.

- withdraw(amount): to subtract funds from the balance. It should only allow positive amounts and prevent withdrawing more than the current balance (insufficient funds).

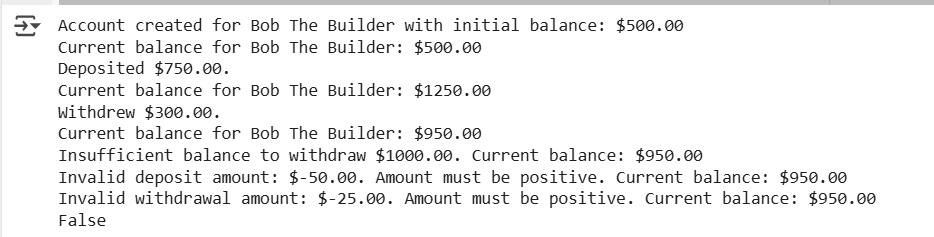
- check\_balance(): to display the current balance and account holder.

Demonstrate the usage of these methods with an example. The example should include creating an account, performing valid and invalid deposits and withdrawals, and show the balance after each transaction, including clear messages for invalid operations indicating the attempted amount and current balance.

**CODE :** 



**OUTPUT :**



**EXPLANATION :**

This code defines a Python class named BankAccount which models a simple bank account. It includes attributes to store information about the account and methods to perform common banking operations like depositing, withdrawing, and checking the balance.

Here's a breakdown of the key parts:

class BankAccount:  
  """Represents a simple bank account."""  
  # ... rest of the class ...

This line defines the BankAccount class. The docstring provides a brief description of what the class represents.

  def \_\_init\_\_(self, account\_holder, initial\_balance=0):  
    """Initializes a new BankAccount object.  
    # ... docstring ...  
    """  
    self.account\_holder = account\_holder  
    self.balance = initial\_balance  
    print(f"Account created for {self.account\_holder} with initial balance: ${self.balance:.2f}")

This is the **constructor** method (\_\_init\_\_). It's called automatically when you create a new BankAccount object.

self: This refers to the instance of the class being created. It's used to access and set the attributes of that specific object.

account\_holder: This argument takes the name of the person who owns the account.

initial\_balance=0: This argument takes the starting amount of money in the account. It has a default value of 0 if no initial balance is provided.

Inside the method, self.account\_holder = account\_holder and self.balance = initial\_balance assign the provided values to the instance's attributes.

A confirmation message is printed showing that the account has been created with the initial balance.

  def deposit(self, amount):  
    """Deposits money into the account.  
    # ... docstring ...  
    """  
    if amount > 0:  
      self.balance += amount  
      print(f"Deposited ${amount:.2f}.")  
      self.check\_balance() # Show balance after deposit  
    else:  
      print(f"Invalid deposit amount: ${amount:.2f}. Amount must be positive. Current balance: ${self.balance:.2f}") # Modified message

This is the deposit method.

self: Refers to the specific bank account instance.

amount: The amount of money to deposit.

if amount > 0:: It checks if the deposit amount is a positive number.

If positive, self.balance += amount adds the amount to the account's balance. A success message and the new balance are printed (by calling self.check\_balance()).

If not positive, it prints an error message indicating that the amount must be positive and shows the attempted amount and current balance.

  def withdraw(self, amount):  
    """Withdraws money from the account.  
    # ... docstring ...  
    """  
    if amount > 0:  
      if self.balance >= amount:  
        self.balance -= amount  
        print(f"Withdrew ${amount:.2f}.")  
        self.check\_balance() # Show balance after successful withdrawal  
        return True  
      else:  
        print(f"Insufficient balance to withdraw ${amount:.2f}. Current balance: ${self.balance:.2f}") # Modified message  
        # self.check\_balance() # Already included in the message  
        return False  
    else:  
      print(f"Invalid withdrawal amount: ${amount:.2f}. Amount must be positive. Current balance: ${self.balance:.2f}") # Modified message  
      # self.check\_balance() # Already included in the message  
      return False

This is the withdraw method.

self: Refers to the specific bank account instance.

amount: The amount of money to withdraw.

if amount > 0:: It first checks if the withdrawal amount is positive.

If positive, it then checks if self.balance >= amount: to see if there are enough funds in the account.

If there are sufficient funds, self.balance -= amount subtracts the amount from the balance. A success message, the new balance, and True are printed.

If there are insufficient funds, it prints an "Insufficient balance" error message showing the attempted amount and current balance, and returns False.

If the initial withdrawal amount was not positive, it prints an "Invalid withdrawal amount" error message showing the attempted amount and current balance, and returns False.

  def check\_balance(self):  
    """Displays the current account balance."""  
    print(f"Current balance for {self.account\_holder}: ${self.balance:.2f}")

This is the check\_balance method.

self: Refers to the specific bank account instance.

It simply prints the current balance and the account\_holder's name for that instance.

# Example Usage:  
# Create a bank account with modified details  
# ... perform transactions ...

This section demonstrates how to create an instance of the BankAccount class (my\_account = BankAccount(...)) and then call the methods (deposit, withdraw, check\_balance) on that instance to perform transactions and display the results. It includes examples of valid operations and attempts at invalid operations (overdrawing, negative amounts) to show how the error handling works and how the balance is displayed after each step.

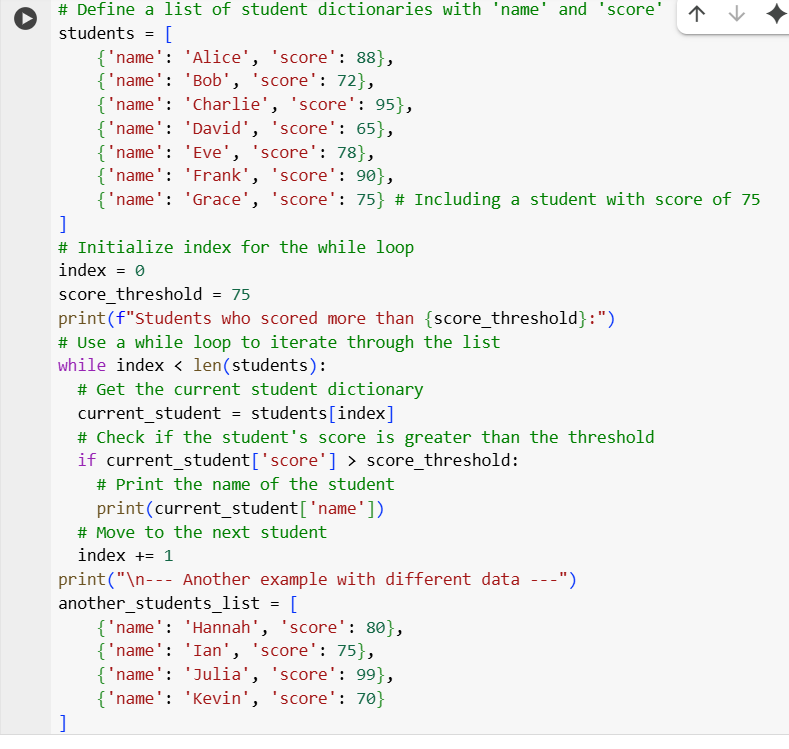
**TASK 4 :** Define a list of student dictionaries with keys name and score. Ask Copilot to write a while loop to print the names of students who scored more than 75.

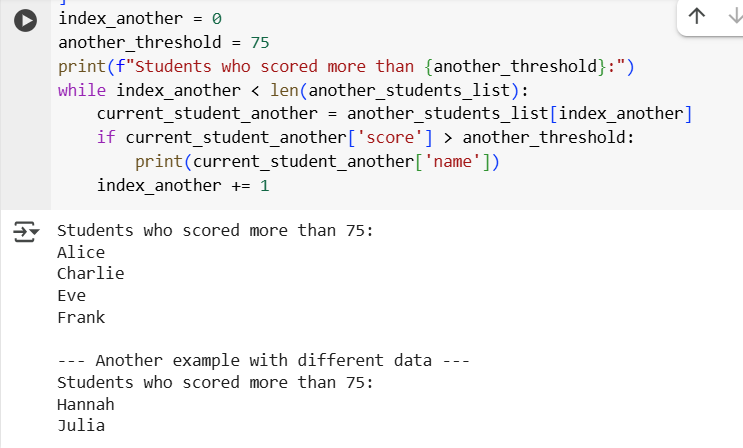
**PROMPT :** write a program in python to define list of students dictionaries with keys and score using while loop to print the names of students and who scored more than 75.

**CODE**

**&**

**OUTPUT :**





**EXPLANATION :**

This code program works with a list of student records, where each record is stored as a dictionary. It uses a while loop to go through each student and identify those who have a score above a specific threshold.

Here's a breakdown of what the code does:

# Define a list of student dictionaries with 'name' and 'score'  
students = [  
    {'name': 'Alice', 'score': 88},  
    # ... other student dictionaries ...  
]

This line creates a list named students. Each item in this list is a dictionary. Each dictionary represents a student and has two key-value pairs: 'name' storing the student's name and 'score' storing their score. This list is the data the program will process.

# Initialize index for the while loop  
index = 0  
score\_threshold = 75  
  
print(f"Students who scored more than {score\_threshold}:")

These lines set up for the while loop.

* index = 0: An integer variable index is initialized to 0. This variable will be used to keep track of the current position (index) in the students list as the while loop progresses.
* score\_threshold = 75: This variable sets the minimum score that a student needs to have to be considered for printing their name.
* A header message is printed to indicate what the output will show.

# Use a while loop to iterate through the list  
while index < len(students):  
  # ... loop body ...

This is the while loop.

* while index < len(students):: The loop will continue to execute as long as the value of index is less than the total number of items in the students list (len(students)). This ensures that the loop processes every student in the list.

  # Get the current student dictionary  
  current\_student = students[index]  
  
  # Check if the student's score is greater than the threshold  
  if current\_student['score'] > score\_threshold:  
    # Print the name of the student  
    print(current\_student['name'])

These lines are inside the while loop and are executed for each student.

* current\_student = students[index]: This line accesses the dictionary at the current index in the students list and assigns it to the variable current\_student.
* if current\_student['score'] > score\_threshold:: This if statement checks the value associated with the key 'score' in the current\_student dictionary. It compares this score to the score\_threshold.
* If the student's score is indeed greater than score\_threshold, the code inside the if block is executed: print(current\_student['name']). This line accesses the value associated with the key 'name' in the current\_student dictionary and prints it to the console.

  # Move to the next student  
  index += 1

This line is also inside the while loop, but outside the if block. index += 1 increments the value of index by 1. This is crucial for the while loop to eventually terminate and move to the next student in the list in the next iteration. If this line were missing, the loop would run indefinitely (an infinite loop).

print("\n--- Another example with different data ---")  
# ... similar loop and logic with different variables ...

This part is similar to the first part but demonstrates the same logic with a different list of student dictionaries, reinforcing how the code can be used with different data.

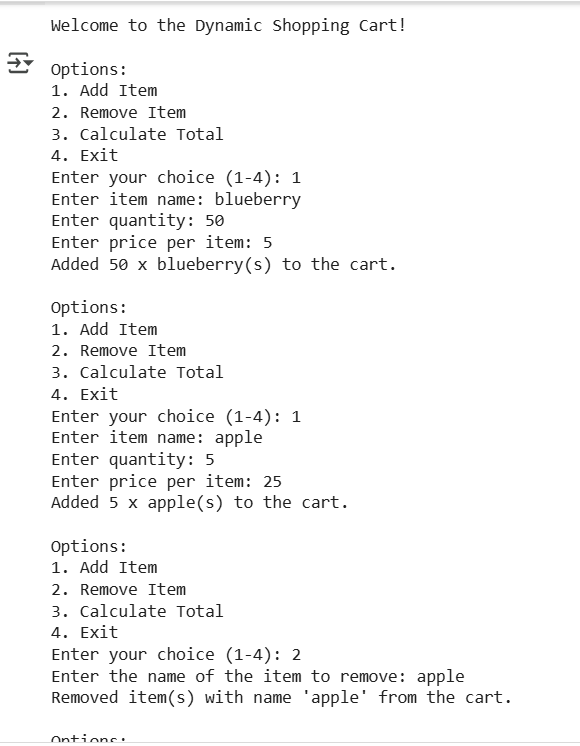
**TASK 5 :** Begin writing a class ShoppingCart with an empty items list. Prompt Copilot to generate methods to add\_item, remove\_item, and use a loop to calculate the total bill using conditional discounts.

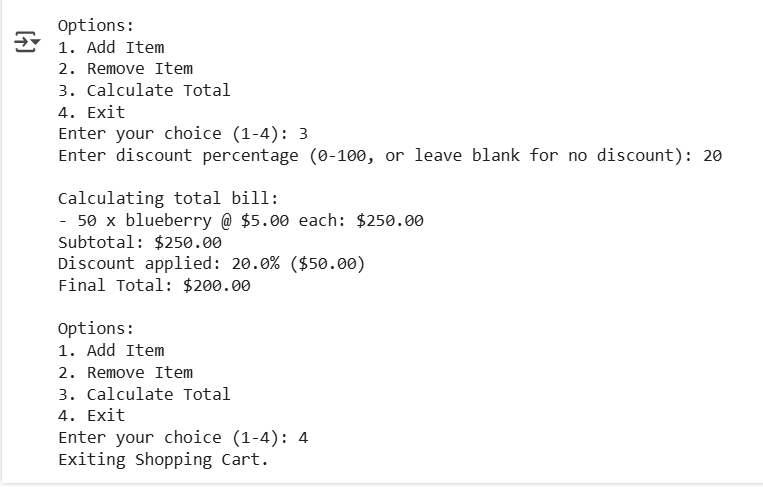
**PROMPT :** Write a python program dynamically with a class ShoppingCart with an empty items list using the methods to add\_item, remove\_item also use a loop to calculate the total bill using conditional discounts implementing if- else statements handling item management and discount logic

**CODE & OUTPUT :**









**EXPLANATION :**

This code provides a **dynamic text-based user interface** that allows you to interact with a ShoppingCart object. It assumes that the ShoppingCart class (defined in cell xQaZOaU18ncd) is already available in the environment.

Here's a breakdown of what the code does:

# Assume the ShoppingCart class from cell xQaZOaU18ncd is defined and available  
  
def run\_dynamic\_shopping\_cart():  
  """Runs a dynamic text-based interface for the ShoppingCart."""  
  my\_cart = ShoppingCart()  
  print("Welcome to the Dynamic Shopping Cart!")  
  
  # ... rest of the function ...

This defines the function run\_dynamic\_shopping\_cart which sets up and runs the interactive interface.

* my\_cart = ShoppingCart(): It creates a new instance of the ShoppingCart class. This is the shopping cart you will be interacting with.
* A welcome message is printed.

  while True:  
    print("\nOptions:")  
    print("1. Add Item")  
    print("2. Remove Item")  
    print("3. Calculate Total")  
    print("4. Exit")  
  
    choice = input("Enter your choice (1-4): ")  
  
    # ... handling user choice ...

This is the main loop that keeps the interface running until you choose to exit.

* while True:: This creates an infinite loop, meaning the interface will continue to display options and ask for input until explicitly told to stop.
* Inside the loop, it **displays the available options** to the user: Add Item, Remove Item, Calculate Total, and Exit.
* choice = input("Enter your choice (1-4): "): It prompts the user to enter their choice and stores the input in the choice variable.

    if choice == '1':  
      # ... code to get item details and add to cart ...  
    elif choice == '2':  
      # ... code to get item name and remove from cart ...  
    elif choice == '3':  
      # ... code to get discount percentage and calculate total ...  
    elif choice == '4':  
      print("Exiting Shopping Cart.")  
      break # Exit the while loop  
    else:  
      print("Invalid choice. Please enter a number between 1 and 4.")

This block of if-elif-else statements handles the user's input based on their choice.

* **If choice is '1' (Add Item):** It prompts the user for the item name, quantity, and price. It includes while loops and try-except blocks to ensure that the user enters valid numbers for quantity (positive integer) and price (non-negative number). Once valid input is received, it calls the my\_cart.add\_item() method with the entered details.
* **If choice is '2' (Remove Item):** It prompts the user for the name of the item to remove and calls the my\_cart.remove\_item() method.
* **If choice is '3' (Calculate Total):** It prompts the user for a discount percentage (allowing them to leave it blank for no discount). It includes a while loop and try-except block for valid numerical input within the 0-100 range. It then calls the my\_cart.calculate\_total() method with the specified discount percentage.
* **If choice is '4' (Exit):** It prints an exit message and uses the break statement to exit the while True loop, ending the program.
* **If choice is anything else:** It prints an "Invalid choice" message.

# Run the dynamic interface  
run\_dynamic\_shopping\_cart()

This line at the end of the code block calls the run\_dynamic\_shopping\_cart() function to start the interactive interface when the cell is executed.