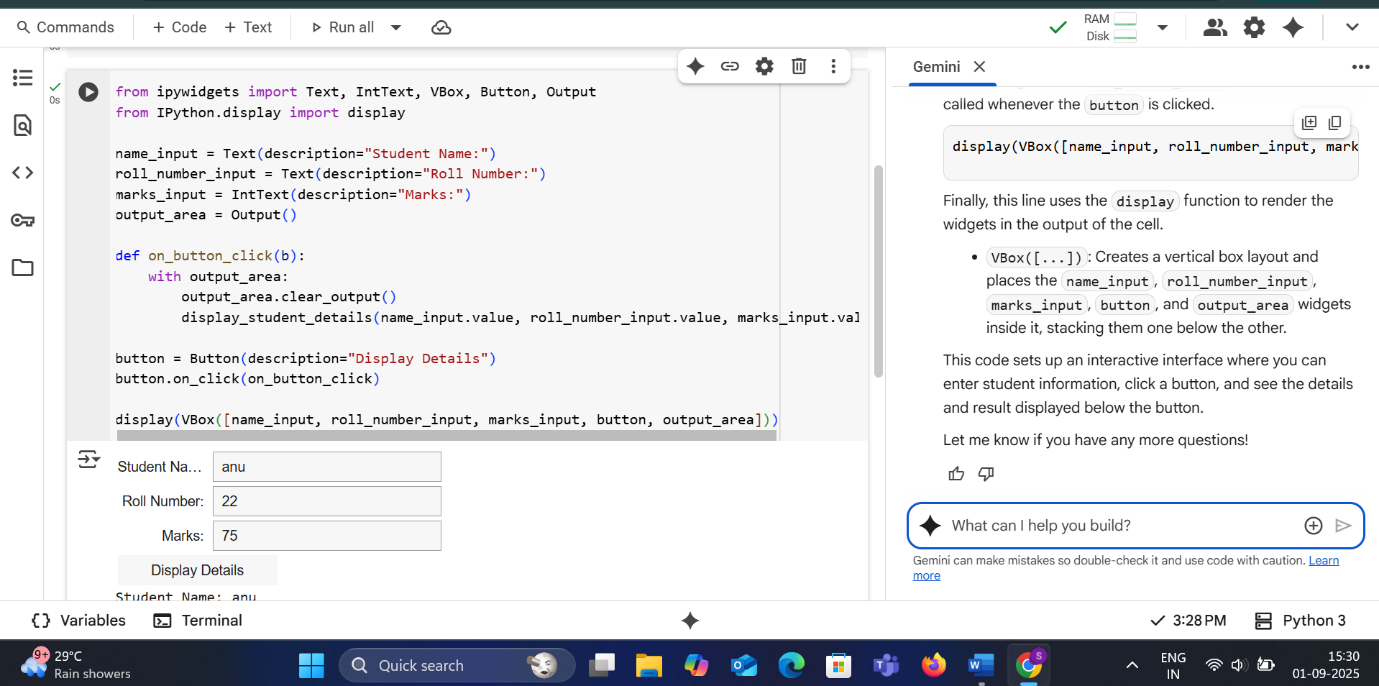
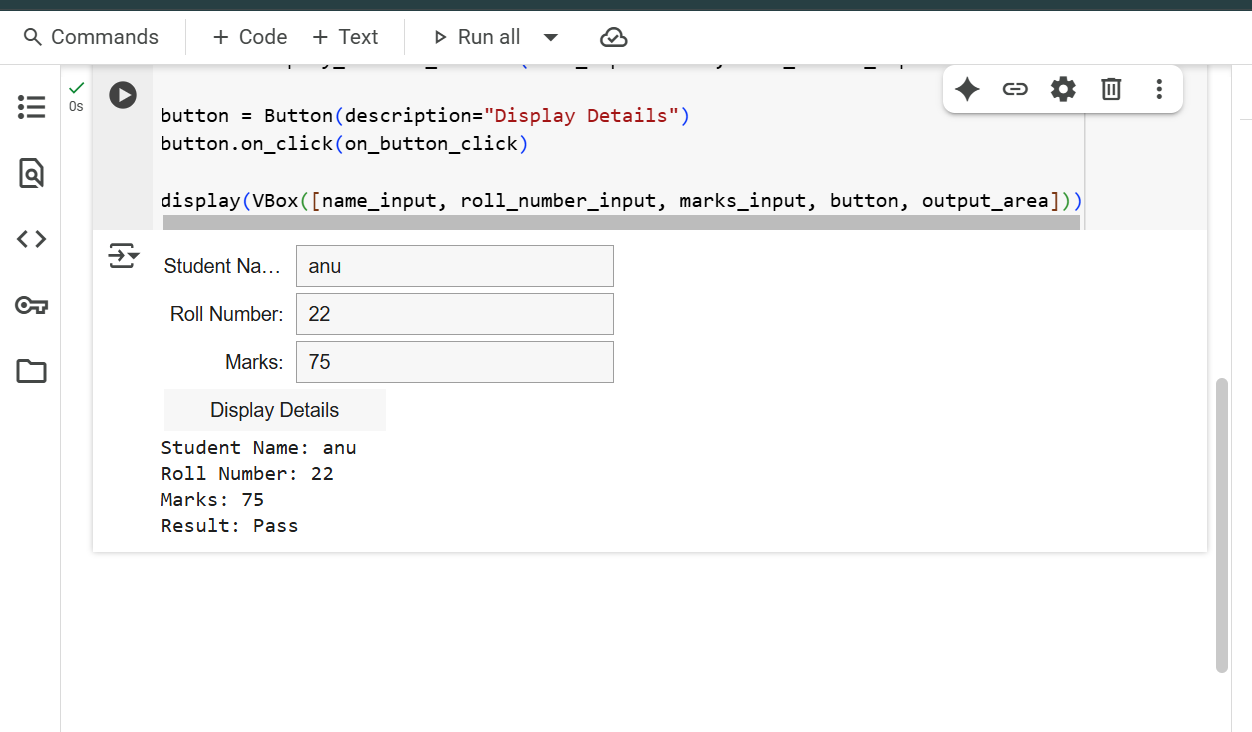
**ASSINGMENT-6**

* **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

****

**OUTPUT:**

****

**EXPLANATION:**

From ipywidgets import Text, IntText, VBox, Button, Output  
from IPython.display import display

This line imports necessary classes from the ipywidgets library and the display function from IPython.display.

* Text: Used to create a single-line text input box.
* IntText: Used to create an input box specifically for integer values.
* VBox: A layout container that arranges widgets vertically.
* Button: Creates a clickable button.
* Output: A widget to capture and display output (like print statements).
* display: A function to render IPython widgets in the output of a notebook cell.

name\_input = Text(description="Student Name:")

This line creates a Text widget for entering the student's name. The description argument provides a label next to the input box.

roll\_number\_input = Text(description="Roll Number:")

This line creates a Text widget for entering the student's roll number with a descriptive label.

marks\_input = IntText(description="Marks:")

This line creates an IntText widget for entering the student's marks. It's used because marks are typically integer values. It also has a descriptive label.

output\_area = Output()

This line creates an Output widget, which will serve as the area where the results (the student details and pass/fail status) will be displayed after the button is clicked.

def on\_button\_click(b):  
    with output\_area:  
        output\_area.clear\_output()  
        display\_student\_details(name\_input.value, roll\_number\_input.value, marks\_input.value)

This defines a function named on\_button\_click. This function will be executed when the button is clicked.

* with output\_area:: This creates a context manager that directs any output within this block to the output\_area widget.
* output\_area.clear\_output(): This clears any previous output displayed in the output\_area before displaying the new results.
* display\_student\_details(name\_input.value, roll\_number\_input.value, marks\_input.value): This line calls the display\_student\_details function (that we defined in the previous cell) with the values entered in the input boxes. .value is used to get the current content of the input widgets.

button = Button(description="Display Details")

This line creates a Button widget with the text "Display Details" on it.

button.on\_click(on\_button\_click)

This line registers the on\_button\_click function to be called whenever the button is clicked.

display(VBox([name\_input, roll\_number\_input, marks\_input, button, output\_area]))

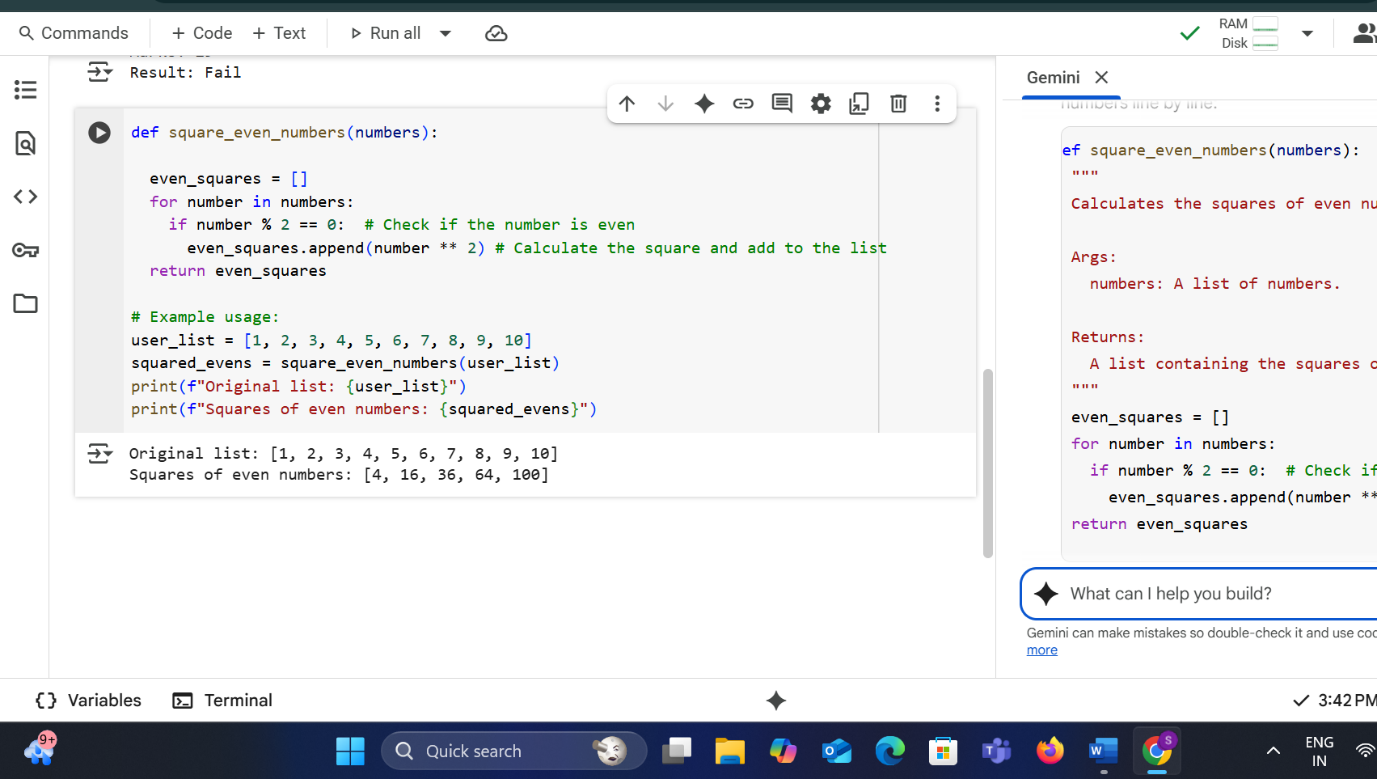
Finally, this line uses the display function to render the widgets in the output of the cell.

* VBox([...]): Creates a vertical box layout and places the name\_input, roll\_number\_input, marks\_input, button, and output\_area widgets inside it, stacking them one below the other.

This code sets up an interactive interface where you can enter student information, click a button, and see the details and result displayed below the button.

* **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.



**EXPLANATION:**

def square\_even\_numbers(numbers):

This line defines a function named square\_even\_numbers that takes one argument, numbers, which is expected to be a list.

  """  
  Calculates the squares of even numbers in a list.  
  
  Args:  
    numbers: A list of numbers.  
  
  Returns:  
    A list containing the squares of the even numbers in the input list.  
  """

This is a docstring explaining the purpose of the function, its arguments, and what it returns.

  even\_squares = []

This line initializes an empty list called even\_squares. This list will store the squares of the even numbers found in the input list.

  for number in numbers:

This is a for loop that iterates through each element in the numbers list. In each iteration, the current element is assigned to the variable number.

    if number % 2 == 0:  # Check if the number is even

Inside the loop, this if statement checks if the current number is even. The modulo operator (%) returns the remainder of a division. If a number divided by 2 has a remainder of 0, it's an even number. The comment # Check if the number is even clarifies this condition.

      even\_squares.append(number \*\* 2) # Calculate the square and add to the list

If the if condition is true (the number is even), this line is executed.

* number \*\* 2: This calculates the square of the even number.
* even\_squares.append(...): This adds the calculated square to the even\_squares list. The comment # Calculate the square and add to the list describes this action.

  return even\_squares

After the loop finishes iterating through all the numbers in the input list, this line returns the even\_squares list, which now contains the squares of all the even numbers that were in the original list.

# Example usage:

This is a comment indicating the start of example code that demonstrates how to use the function.

user\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

This line creates a list of integers called user\_list and assigns it some values.

squared\_evens = square\_even\_numbers(user\_list)

This line calls the square\_even\_numbers function with user\_list as the argument. The returned list (containing the squares of the even numbers) is stored in the variable squared\_evens.

print(f"Original list: {user\_list}")

This line prints the original list to the console using an f-string.

print(f"Squares of even numbers: {squared\_evens}")

This line prints the squared\_evens list (the result of the function call) to the console using an f-string.

This code effectively takes a list, filters out the even numbers, and provides a list of their squares.

* **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.

* **Code**

class BankAccount:

"""Represents a simple bank account."""

def \_\_init\_\_(self, account\_holder, initial\_balance=0):

"""

Initializes a new bank account.

Args:

account\_holder: The name of the account holder.

initial\_balance: The initial balance of the account (default is 0).

"""

self.account\_holder = account\_holder

self.balance = initial\_balance

print(f"Account created for {self.account\_holder} with initial balance of ${self.balance:.2f}")

def display\_details(self):

"""Displays the account holder and current balance."""

print("\nAccount Details:")

print(f"Account Holder: {self.account\_holder}")

print(f"Current Balance: ${self.balance:.2f}")

def deposit(self, amount):

"""

Deposits money into the account.

Args:

amount: The amount to deposit.

"""

if amount > 0:

self.balance += amount

print(f"\nDeposited ${amount:.2f}")

self.display\_details()

else:

print("\nInvalid deposit amount. Please enter a positive amount.")

def withdraw(self, amount):

"""

Withdraws money from the account.

Args:

amount: The amount to withdraw.

"""

if amount > 0:

if self.balance >= amount:

self.balance -= amount

print(f"\nWithdrew ${amount:.2f}")

self.display\_details()

else:

print("\nInsufficient balance.")

self.display\_details()

else:

print("\nInvalid withdrawal amount. Please enter a positive amount.")

# Example usage:

# Create an account

my\_account = BankAccount("John Doe", 1000)

# Display details

my\_account.display\_details()

# Deposit money

my\_account.deposit(500)

# Withdraw money

my\_account.withdraw(200)

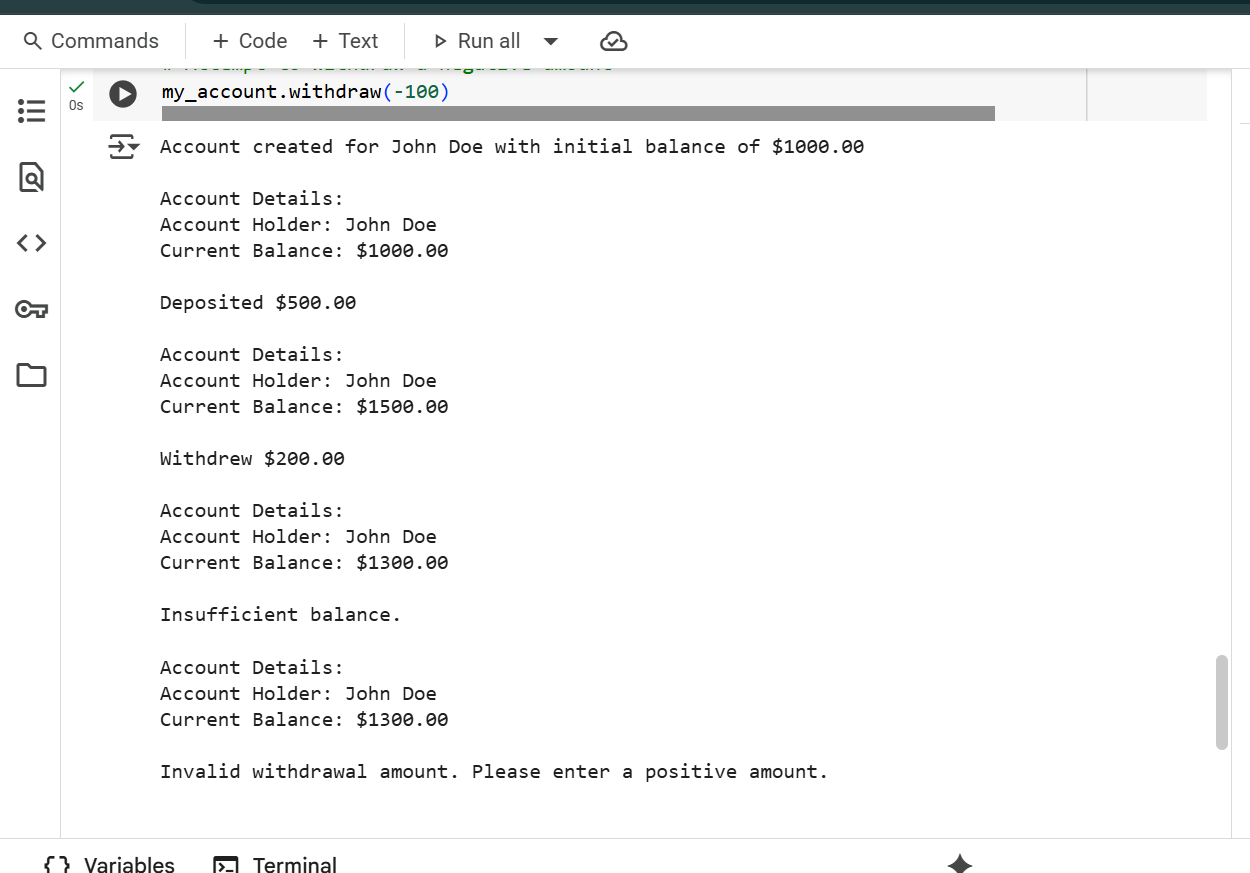
# Attempt to withdraw more than the balance

my\_account.withdraw(1500)

# Attempt to withdraw a negative amount

my\_account.withdraw(-100)

* **output**



* **EXPLANATION**  
  def square\_even\_numbers(numbers):

This line defines a function named square\_even\_numbers that takes one argument, numbers, which is expected to be a list.

  """  
  Calculates the squares of even numbers in a list.  
  
  Args:  
    numbers: A list of numbers.  
  
  Returns:  
    A list containing the squares of the even numbers in the input list.  
  """

This is a docstring explaining the purpose of the function, its arguments, and what it returns.

  even\_squares = []

This line initializes an empty list called even\_squares. This list will store the squares of the even numbers found in the input list.

  for number in numbers:

This is a for loop that iterates through each element in the numbers list. In each iteration, the current element is assigned to the variable number.

    if number % 2 == 0:  # Check if the number is even

Inside the loop, this if statement checks if the current number is even. The modulo operator (%) returns the remainder of a division. If a number divided by 2 has a remainder of 0, it's an even number. The comment # Check if the number is even clarifies this condition.

      even\_squares.append(number \*\* 2) # Calculate the square and add to the list

If the if condition is true (the number is even), this line is executed.

* number \*\* 2: This calculates the square of the even number.
* even\_squares.append(...): This adds the calculated square to the even\_squares list. The comment # Calculate the square and add to the list describes this action.

  return even\_squares

After the loop finishes iterating through all the numbers in the input list, this line returns the even\_squares list, which now contains the squares of all the even numbers that were in the original list.

# Example usage:

This is a comment indicating the start of example code that demonstrates how to use the function.

user\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

This line creates a list of integers called user\_list and assigns it some values.

squared\_evens = square\_even\_numbers(user\_list)

This line calls the square\_even\_numbers function with user\_list as the argument. The returned list (containing the squares of the even numbers) is stored in the variable squared\_evens.

print(f"Original list: {user\_list}")

This line prints the original list to the console using an f-string.

print(f"Squares of even numbers: {squared\_evens}")

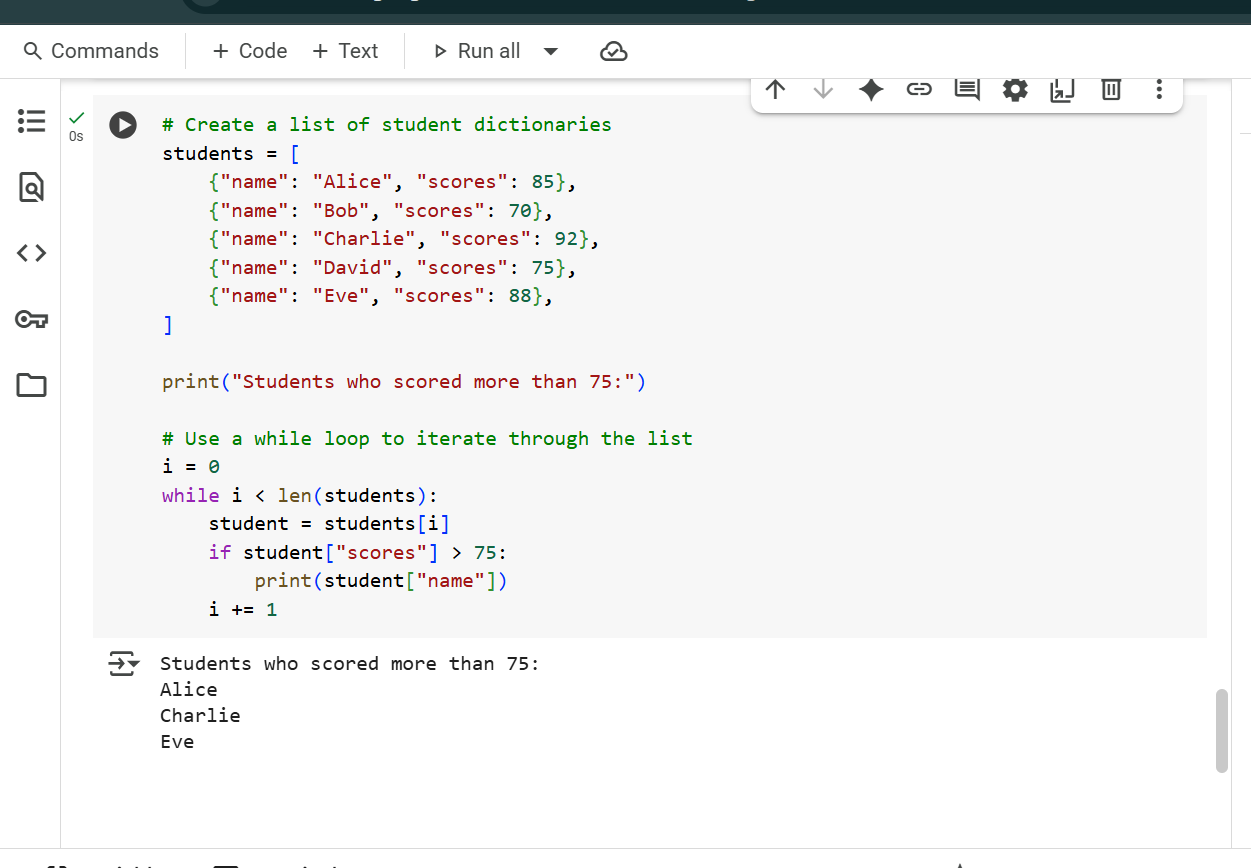
This line prints the squared\_evens list (the result of the function call) to the console using an f-string.

This code effectively takes a list, filters out the even numbers, and provides a list of their squares

* **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.

**CODE AND OUTPUT:**



**EXPLANATION:**

# Create a list of student dictionaries

This is a comment indicating that the following lines create a list containing student information.

students = [  
    {"name": "Alice", "scores": 85},  
    {"name": "Bob", "scores": 70},  
    {"name": "Charlie", "scores": 92},  
    {"name": "David", "scores": 75},  
    {"name": "Eve", "scores": 88},  
]

This block of code creates a list named students. The list contains several elements, and each element is a dictionary.

* Each dictionary represents a student.
* {"name": "Alice", "scores": 85}: This is a dictionary for Alice, with keys "name" and "scores" and their corresponding values.
* The list contains five such dictionaries, one for each student.

print("Students who scored more than 75:")

This line prints a descriptive header to the console before listing the names of the students who meet the criteria.

# Use a while loop to iterate through the list

This is a comment indicating that a while loop will be used to go through the students list.

i = 0

This line initializes a variable i to 0. This variable will be used as an index to access elements in the students list, starting from the first element (at index 0).

while i < len(students):

This is the while loop condition. The loop will continue to execute as long as the value of i is less than the total number of elements in the students list (len(students)). len(students) returns the number of items in the list.

    student = students[i]

Inside the loop, this line accesses the element at the current index i from the students list and assigns it to the variable student. Since each element is a dictionary, student will hold the dictionary for the current student in the iteration.

    if student["scores"] > 75:

This if statement checks if the value associated with the key "scores" in the current student dictionary is greater than 75.

        print(student["name"])

If the condition in the if statement is true (the student's score is greater than 75), this line prints the value associated with the key "name" from the current student dictionary. This prints the name of the student who scored more than 75.

    i += 1

This line increments the value of i by 1. This is crucial for the while loop to eventually terminate; it moves the index to the next element in the list for the next iteration.

The loop continues until i is no longer less than the length of the students list, at which point all elements have been checked.

* **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

**CODE AND OUTPUT**

class ShoppingCart:

"""Represents a simple shopping cart."""

def \_\_init\_\_(self):

"""Initializes an empty shopping cart."""

self.items = []

print("Shopping cart created.")

def add\_item(self, item):

"""

Adds an item to the shopping cart.

Args:

item: A dictionary representing the item (e.g., {'name': 'Apple', 'price': 0.5, 'quantity': 10}).

"""

self.items.append(item)

print(f"Added {item.get('quantity', 1)} x {item.get('name', 'item')} to the cart.")

def remove\_item(self, item\_name):

"""

Removes an item from the shopping cart by name.

Args:

item\_name: The name of the item to remove.

"""

initial\_item\_count = len(self.items)

self.items = [item for item in self.items if item.get('name') != item\_name]

if len(self.items) < initial\_item\_count:

print(f"Removed {item\_name} from the cart.")

else:

print(f"{item\_name} not found in the cart.")

def calculate\_total(self):

"""

Calculates the total bill with conditional discounts.

Returns:

The total bill amount after applying discounts.

"""

subtotal = 0

for item in self.items:

subtotal += item.get('price', 0) \* item.get('quantity', 1)

total = subtotal

# Apply conditional discounts

if subtotal > 100:

discount = subtotal \* 0.10 # 10% discount for orders over $100

total -= discount

print(f"\nApplied 10% discount: -${discount:.2f}")

elif subtotal > 50:

discount = subtotal \* 0.05 # 5% discount for orders over $50

total -= discount

print(f"\nApplied 5% discount: -${discount:.2f}")

print(f"\nSubtotal: ${subtotal:.2f}")

print(f"Total Bill: ${total:.2f}")

return total

# Example Usage:

my\_cart = ShoppingCart()

my\_cart.add\_item({'name': 'Laptop', 'price': 800, 'quantity': 1})

my\_cart.add\_item({'name': 'Mouse', 'price': 25, 'quantity': 2})

my\_cart.add\_item({'name': 'Keyboard', 'price': 75, 'quantity': 1})

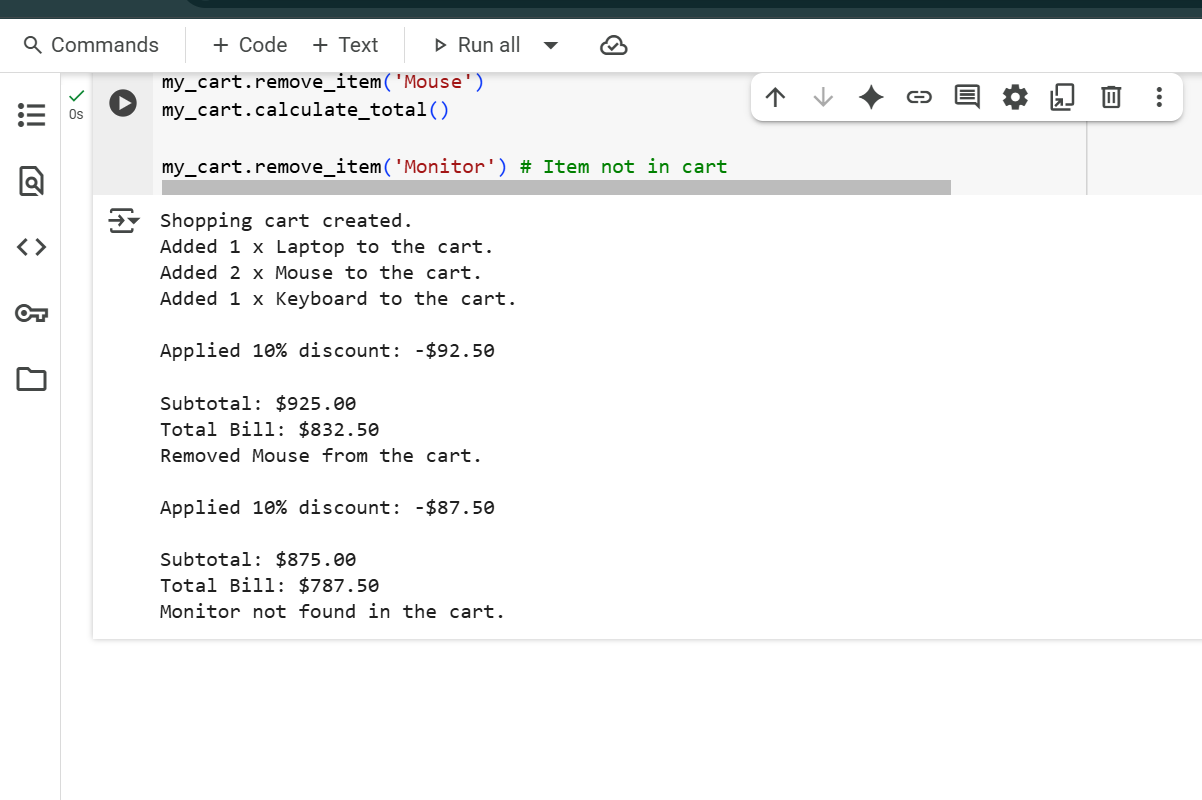
my\_cart.calculate\_total()

my\_cart.remove\_item('Mouse')

my\_cart.calculate\_total()

my\_cart.remove\_item('Monitor') # Item not in cart

* **OUTPUT**



* **EXPLANATION**

class ShoppingCart:

This line defines a new class named ShoppingCart. This class will be a blueprint for creating shopping cart objects.

    """Represents a simple shopping cart."""

This is the docstring for the ShoppingCart class, explaining its purpose.

    def \_\_init\_\_(self):

This is the initializer (constructor) method of the class. It's automatically called when you create a new ShoppingCart object.

* self: Refers to the instance of the class being created.

        """Initializes an empty shopping cart."""

This is the docstring for the \_\_init\_\_ method.

        self.items = []

This line initializes an empty list called self.items. This list will store the items added to the shopping cart. Each item will likely be represented as a dictionary.

        print("Shopping cart created.")

This line prints a confirmation message to the console when a new shopping cart object is created.

    def add\_item(self, item):

This defines a method named add\_item within the ShoppingCart class. It takes self and item as arguments.

* item: This argument is expected to be a dictionary representing the item to be added.

        """  
        Adds an item to the shopping cart.  
  
        Args:  
            item: A dictionary representing the item (e.g., {'name': 'Apple', 'price': 0.5, 'quantity': 10}).  
        """

This is the docstring for the add\_item method, explaining its purpose and arguments.

        self.items.append(item)

This line adds the item dictionary to the self.items list.

        print(f"Added {item.get('quantity', 1)} x {item.get('name', 'item')} to the cart.")

This line prints a confirmation message indicating that an item has been added. It uses the .get() method with a default value (1 for quantity, 'item' for name) to safely access keys in the item dictionary, in case they are missing.

    def remove\_item(self, item\_name):

This defines a method named remove\_item within the ShoppingCart class. It takes self and item\_name as arguments.

* item\_name: The name of the item to be removed.

        """  
        Removes an item from the shopping cart by name.

Args:  
            item\_name: The name of the item to remove.  
        """

This is the docstring for the remove\_item method.

        initial\_item\_count = len(self.items)

This line stores the initial number of items in the cart before attempting removal. This is used later to check if an item was actually removed.

        self.items = [item for item in self.items if item.get('name') != item\_name]

This is a list comprehension. It creates a new list for self.items by including only those dictionaries from the original self.items where the value associated with the key 'name' is not equal to the item\_name to be removed. This effectively removes all items with the matching name.

        if len(self.items) < initial\_item\_count:

This if statement checks if the length of the self.items list is now less than the initial\_item\_count. If it is, it means at least one item was removed.

            print(f"Removed {item\_name} from the cart.")

If an item was removed, this line prints a confirmation message.

        else:

This is the else block for the previous if statement.

            print(f"{item\_name} not found in the cart.")

If no item was removed (because the item name was not found in the cart), this line prints a message indicating that the item was not found.

    def calculate\_total(self):

This defines a method named calculate\_total within the ShoppingCart class. It takes only self as an argument.

        """  
        Calculates the total bill with conditional discounts.  
  
        Returns:  
            The total bill amount after applying discounts.  
        """

This is the docstring for the calculate\_total method.

        subtotal = 0

This line initializes a variable subtotal to 0. This will accumulate the total price of all items before discounts.

        for item in self.items:

This is a for loop that iterates through each item (dictionary) in the self.items list.

            subtotal += item.get('price', 0) \* item.get('quantity', 1)

Inside the loop, this line calculates the cost for the current item and adds it to the subtotal. It safely accesses the 'price' and 'quantity' keys using .get() with default values of 0 and 1 respectively, in case those keys are missing in an item dictionary.

        total = subtotal

This line initializes a variable total with the calculated subtotal. Discounts will be applied to this total.

        # Apply conditional discounts

This is a comment indicating the start of the discount logic.

        if subtotal > 100:

This if statement checks if the subtotal is greater than 100.

            discount = subtotal \* 0.10  # 10% discount for orders over $100

If the subtotal is greater than 100, this line calculates a 10% discount. The comment explains the discount rule.

            total -= discount

This line subtracts the calculated discount from the total.

            print(f"\nApplied 10% discount: -${discount:.2f}")

This line prints a message indicating that the 10% discount was applied and shows the discount amount formatted to two decimal places.

        elif subtotal > 50:

This is an elif (else if) statement. If the subtotal was not greater than 100, this checks if it is greater than 50.

            discount = subtotal \* 0.05  # 5% discount for orders over $50

If the subtotal is greater than 50 but not greater than 100, this line calculates a 5% discount. The comment explains this discount rule.

            total -= discount

This line subtracts the calculated 5% discount from the total.

            print(f"\nApplied 5% discount: -${discount:.2f}")

This line prints a message indicating that the 5% discount was applied and shows the discount amount formatted to two decimal places.

        print(f"\nSubtotal: ${subtotal:.2f}")

After checking for discounts, this line prints the original subtotal formatted to two decimal places.

        print(f"Total Bill: ${total:.2f}")

This line prints the final total bill amount after any applicable discounts have been applied, formatted to two decimal places.

        return total

This line returns the final calculated total bill amount.

# Example Usage:

This is a comment indicating the start of the example code for using the ShoppingCart class.

my\_cart = ShoppingCart()

This line creates a new instance (object) of the ShoppingCart class and assigns it to the variable my\_cart. This calls the \_\_init\_\_ method.

my\_cart.add\_item({'name': 'Laptop', 'price': 800, 'quantity': 1})  
my\_cart.add\_item({'name': 'Mouse', 'price': 25, 'quantity': 2})  
my\_cart.add\_item({'name': 'Keyboard', 'price': 75, 'quantity': 1})

These lines call the add\_item method on the my\_cart object to add three different items to the cart. Each item is passed as a dictionary.

my\_cart.calculate\_total()

This line calls the calculate\_total method on the my\_cart object to calculate and print the total bill with discounts for the items currently in the cart.

my\_cart.remove\_item('Mouse')

This line calls the remove\_item method to remove the item with the name 'Mouse' from the cart.

my\_cart.calculate\_total()

This line calls calculate\_total again to show the updated total after removing the 'Mouse'.

my\_cart.remove\_item('Monitor') # Item not in cart

This line attempts to remove an item named 'Monitor', which is not in the cart. The code is designed to handle this and print a "not found" message. The comment clarifies that this item is not expected to be in the cart.

This code provides a comprehensive example of a simple shopping cart with item management and a basic discount system.