

NAME : B.Pavan

2403A51438

BATCH 16

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
ProgramName: B. Tech		Assignment Type: Lab	AcademicYear: 2025-2026
CourseCoordinatorName		Venkataramana Veeramsetty	
Instructor(s)Name		Dr. V. Venkataramana (Co-ordinator)	
		Dr. T. Sampath Kumar	
		Dr. Pramoda Patro	
		Dr. Brij Kishor Tiwari	
		Dr.J.Ravichander	
		Dr. Mohammand Ali Shaik	
		Dr. Anirodh Kumar	
		Mr. S.Naresh Kumar	
		Dr. RAJESH VELPULA	
		Mr. Kundhan Kumar	
		Ms. Ch.Rajitha	
		Mr. M Prakash	
		Mr. B.Raju	
		Intern 1 (Dharma teja)	
		Intern 2 (Sai Prasad)	
		Intern 3 (Sowmya)	
NS_2 (Mounika)			
CourseCode	24CS002PC215	CourseTitle	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment	Week3 - Thursday	Time(s)	
Duration	2 Hours	Applicable to Batches	
AssignmentNumber: 6.4(Present assignment number)/24(Total number of assignments)			
Q.No.	Question	Expected Time to complete	
1	<p>Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals</p> <p>Lab Objectives:</p> <ul style="list-style-type: none"> To explore AI-powered auto-completion features for core Python constructs. To analyze how AI suggests logic for class definitions, loops, and conditionals. To evaluate the completeness and correctness of code generated by AI assistants. <p>Lab Outcomes (LOs):</p>	Week3 - Thursday	

After completing this lab, students will be able to:

- Use AI tools to generate and complete class definitions and methods.
- Understand and assess AI-suggested loops for iterative tasks.
- Generate conditional statements through prompt-driven suggestions.
- Critically evaluate AI-assisted code for correctness and clarity.

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.

Expected Outcome #1:

- Completed class with Copilot-generated methods like display_details() and is_passed(), demonstrating use of if-else conditions.

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.

Expected Outcome #2:

- A complete loop generated by Copilot with conditional logic (if number % 2 == 0) and appropriate output.

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.

Expected Outcome #3:

- Functional class with complete method definitions using if conditions and self attributes. Code should prevent overdrawing.

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.

Expected Outcome #4:

- A complete while loop generated by Copilot with proper condition checks and formatted output.

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.

Expected Outcome #5:

- A fully implemented ShoppingCart class with Copilot-generated loops and if-else statements handling item management and discount logic.

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Evaluation Criteria:

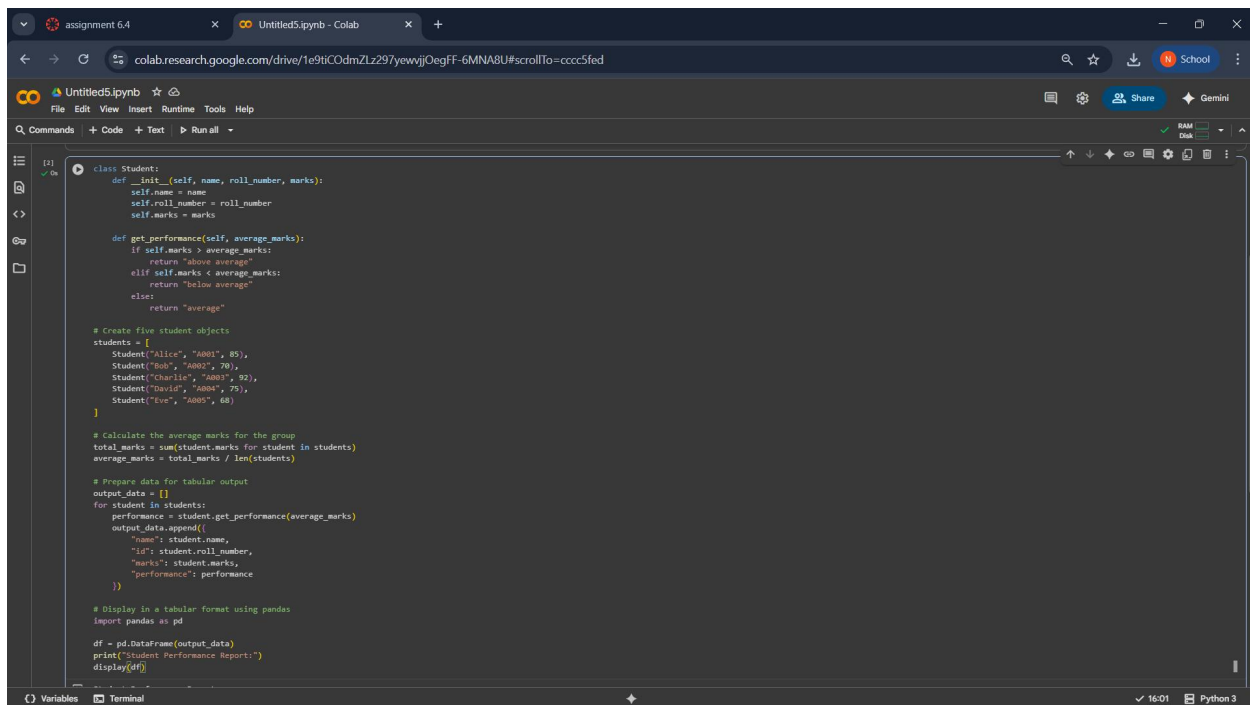
Criteria	Max Marks
Class	1
Loop	1
condition	0.5
Total	2.5 Marks

TASK 1 :

PROMPT :

write a python code to Start a Python class named Student with attributes name, roll_number, and marks. check if marks are above average or not.

CODE :



```
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def get_performance(self, average_marks):
        if self.marks > average_marks:
            return "above average"
        elif self.marks < average_marks:
            return "below average"
        else:
            return "average"

# Create five student objects
students = [
    Student("Alice", "A001", 85),
    Student("Bob", "A002", 70),
    Student("Charlie", "A003", 92),
    Student("David", "A004", 75),
    Student("Eve", "A005", 68)
]

# Calculate the average marks for the group
total_marks = sum(student.marks for student in students)
average_marks = total_marks / len(students)

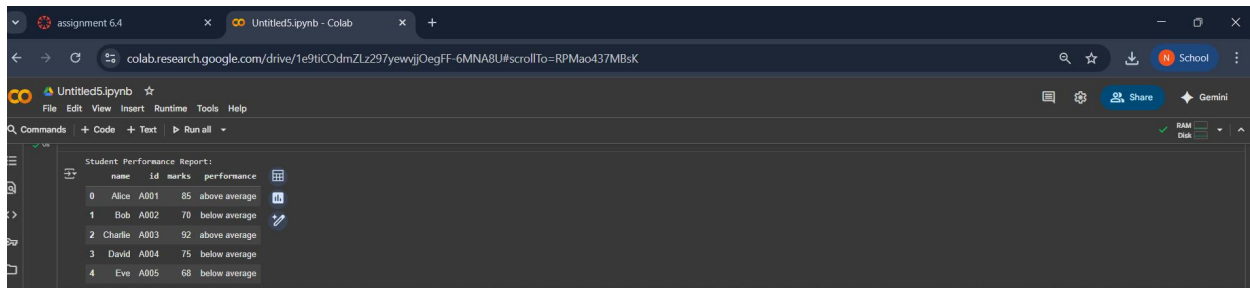
# Prepare data for tabular output
output_data = []
for student in students:
    performance = student.get_performance(average_marks)
    output_data.append({
        "name": student.name,
        "id": student.roll_number,
        "marks": student.marks,
        "performance": performance
    })

# Display in a tabular format using pandas
import pandas as pd
df = pd.DataFrame(output_data)
print("Student Performance Report:")
display(df)
```

EXPLANATION :

- The Student class stores name, roll_number, and marks using self.
- The method check_above_average() uses an if condition to compare marks against an average (default = 50).
- It prints whether the student's marks are above average or not.

OUTPUT :



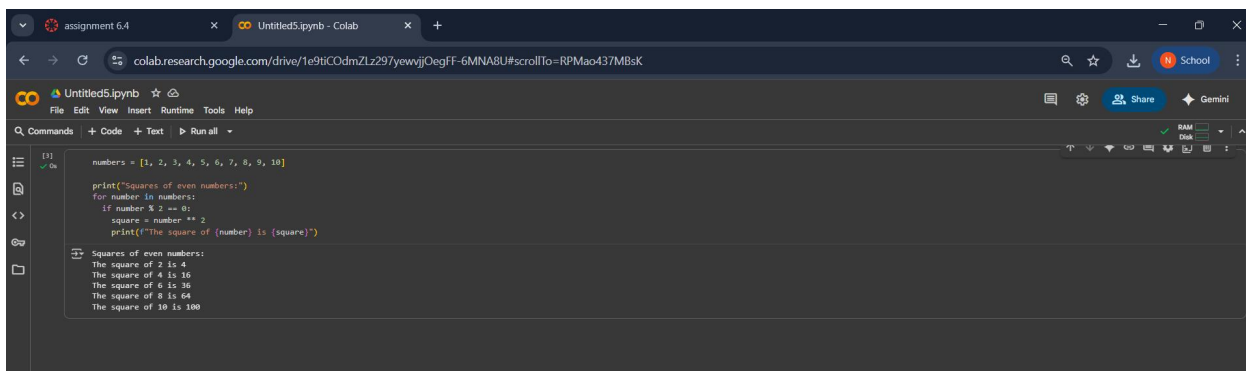
The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: colab.research.google.com/drive/1e9tiCOdmZLz297yewjyOegFF-6MNA8U#scrollTo=RPMao437MBsK. The notebook is titled 'Untitled5.ipynb'. The code cell contains a table titled 'Student Performance Report:' with columns 'name', 'id', 'marks', and 'performance'. The table has 5 rows of data.

	name	id	marks	performance
0	Alice	A001	85	above average
1	Bob	A002	70	below average
2	Charlie	A003	92	above average
3	David	A004	75	below average
4	Eve	A005	68	below average

TASK 2 :

PROMPT :

write a python code to calculate and print the square of even numbers only. use conditional logic (if number % 2 == 0) and give appropriate output.



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: colab.research.google.com/drive/1e9tiCOdmZLz297yewjyOegFF-6MNA8U#scrollTo=RPMao437MBsK. The notebook is titled 'Untitled5.ipynb'. The code cell contains the following Python code:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print("Squares of even numbers:")
for number in numbers:
    if number % 2 == 0:
        square = number ** 2
        print(f"The square of {number} is {square}")
```

The output of the code is displayed below the code cell:

```
Squares of even numbers:
The square of 2 is 4
The square of 4 is 16
The square of 6 is 36
The square of 8 is 64
The square of 10 is 100
```

CODE & OUTPUT:

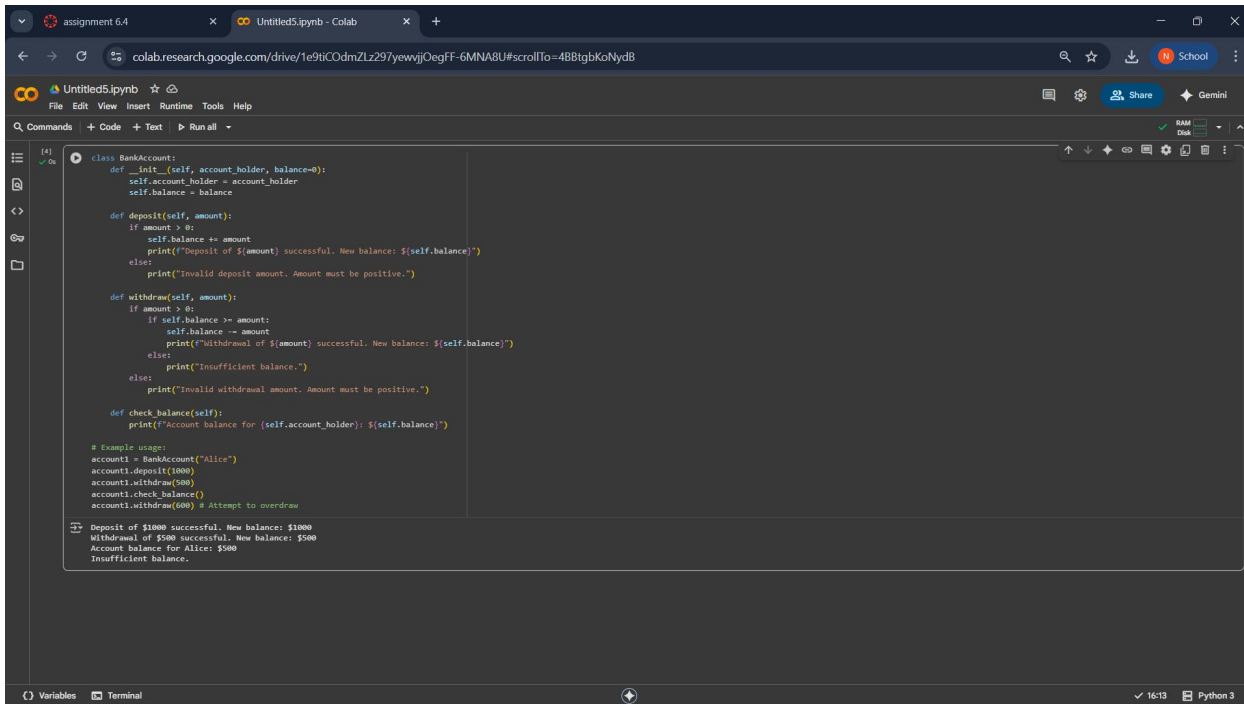
EXPLANATION :

- The program loops through numbers from 1 to 10 using for number in range(1, 11).
- The condition if number % 2 == 0 checks whether the number is even.
- If the number is even, its square is calculated using number ** 2.
- Finally, the program prints the number along with its square

TASK 3 :

PROMPT :

Create a class called BankAccount with attributes account_holder and balance and complete methods for deposit(), withdraw(), and check for insufficient balance.code should be Functional class with complete method definitions using if conditions and self attributes and should prevent overdrawing.



The screenshot shows a Google Colab notebook titled 'Untitled5.ipynb'. The code defines a `BankAccount` class with attributes `account_holder` and `balance`. It includes methods for `deposit`, `withdraw`, and `check_balance`. The `deposit` method adds money if the amount is positive. The `withdraw` method checks if the balance is sufficient before subtracting the amount. The `check_balance` method prints the account holder's name and current balance. Example usage is provided at the bottom of the code cell.

```
class BankAccount:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposit of ${amount} successful. New balance: ${self.balance}")
        else:
            print("Invalid deposit amount. Amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if self.balance >= amount:
                self.balance -= amount
                print(f"Withdrawal of ${amount} successful. New balance: ${self.balance}")
            else:
                print("Insufficient balance.")
        else:
            print("Invalid withdrawal amount. Amount must be positive.")

    def check_balance(self):
        print(f"Account balance for {self.account_holder}: ${self.balance}")

# Example usage:
account1 = BankAccount("Alice")
account1.deposit(1000)
account1.withdraw(500)
account1.check_balance()
account1.withdraw(600) # Attempt to overdraw
```

Output:

```
Deposit of $1000 successful. New balance: $1000
Withdrawal of $500 successful. New balance: $500
Account balance for Alice: $500
Insufficient balance.
```

CODE & OUTPUT :

EXPLANATION :

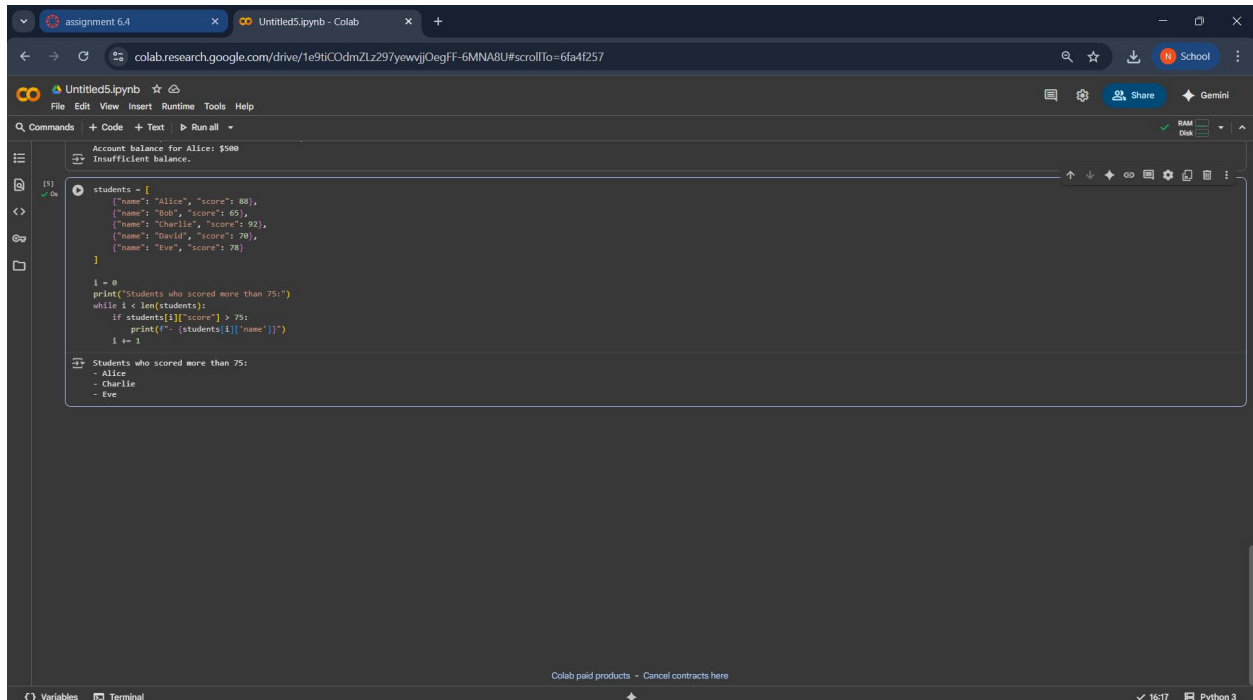
- The class keeps `account_holder` and `balance` as attributes using `self`. `deposit()` adds money if the amount is positive.
- `withdraw()` uses if conditions to prevent overdrawing and invalid amounts.
- `check_balance()` shows the account holder's name and current balance.

TASK 4 :

PROMPT :

Create a class called BankAccount with attributes account_holder and balance and complete methods for deposit(), withdraw(), and check for insufficient balance. code should be Functional class with complete method definitions using if conditions and self attributes and should prevent overdrawing.

CODE & OUTPUT :



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: `colab.research.google.com/drive/1e9hCOdmZLz297yewjJOegfF-6MNA8U#scrollTo=6fa4f257`. The notebook has two tabs: "assignment 6.4" and "Untitled5.ipynb - Colab". The "Untitled5.ipynb" tab is active, showing a code cell with the following Python code:

```
Account balance for Alice: $500
Insufficient balance.

students = [
    {"name": "Alice", "score": 88},
    {"name": "Bob", "score": 65},
    {"name": "Charlie", "score": 92},
    {"name": "David", "score": 70},
    {"name": "Eve", "score": 78}
]

i = 0
print("Students who scored more than 75:")
while i < len(students):
    if students[i]["score"] > 75:
        print(f"- {students[i]['name']}")
    i += 1
```

The output of the code cell is displayed below the code:

```
Students who scored more than 75:
- Alice
- Charlie
- Eve
```

The bottom status bar of the Colab interface shows "Colab paid products - Cancel contracts here", a clock icon with the time "16:17", and "Python 3".

EXPLANATION :

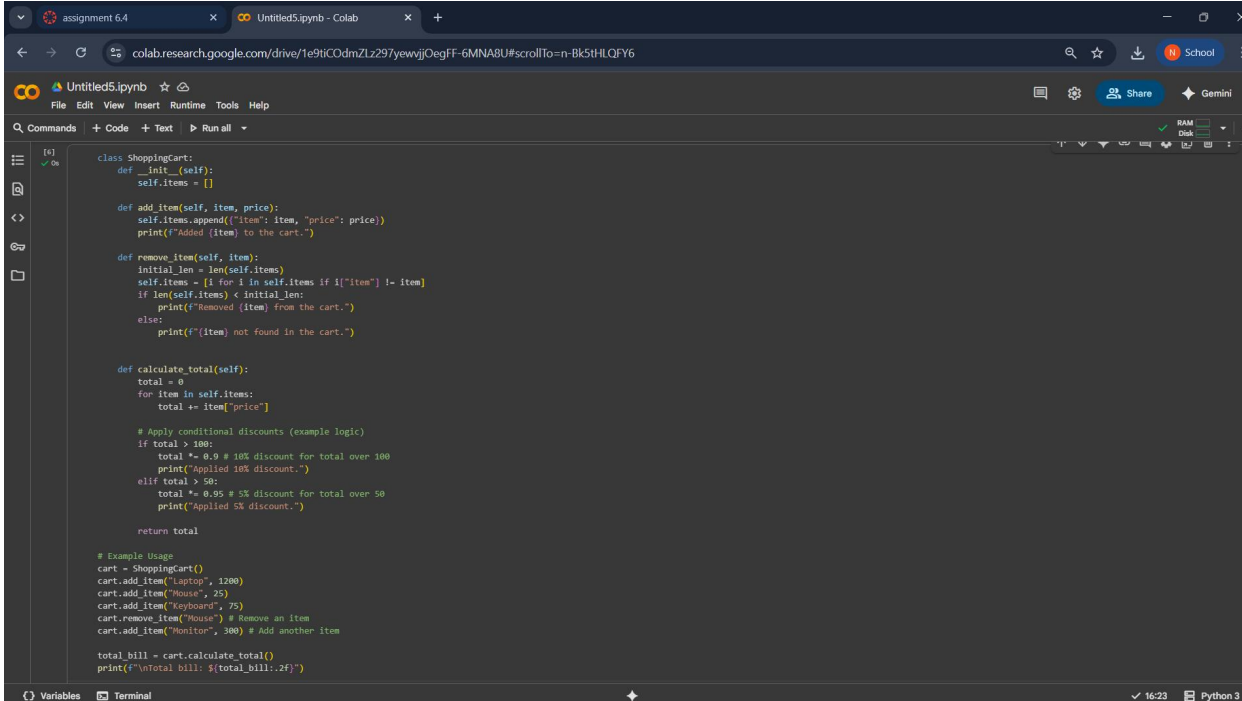
- The class uses `self.account_holder` and `self.balance` to store account details.
- `deposit()` and `withdraw()` modify the balance with if checks for valid amounts.
- The `withdraw` method prevents overdrawing by checking if the requested amount exceeds the balance.

TASK 5 :

PROMPT :

Begin writing a class ShoppingCart with an empty items list. generate methods to add_item, remove_item, and use a loop to calculate the total bill using conditional discounts.I need a fully implemented ShoppingCart class with loops and if-else statements. handling item management and discount logic.Explain the code in 3 lines.

CODE :



```
class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, item, price):
        self.items.append({"item": item, "price": price})
        print(f"Added {item} to the cart.")

    def remove_item(self, item):
        initial_len = len(self.items)
        self.items = [i for i in self.items if i["item"] != item]
        if len(self.items) < initial_len:
            print(f"Removed {item} from the cart.")
        else:
            print(f"{item} not found in the cart.")

    def calculate_total(self):
        total = 0
        for item in self.items:
            total += item["price"]

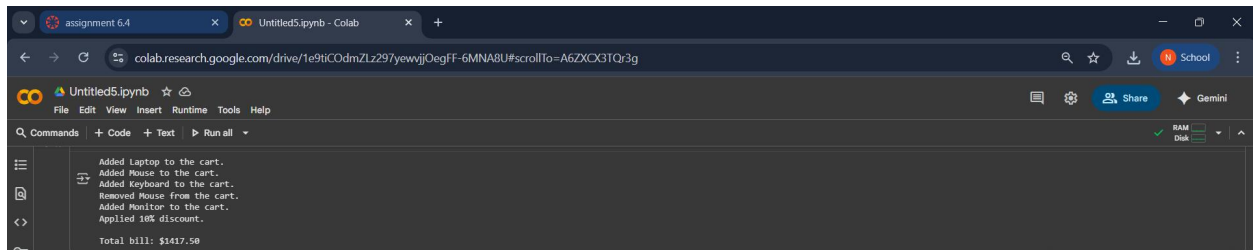
        # Apply conditional discounts (example logic)
        if total > 100:
            total *= 0.9 # 10% discount for total over 100
            print("Applied 10% discount.")
        elif total > 50:
            total *= 0.95 # 5% discount for total over 50
            print("Applied 5% discount.")

        return total

# Example Usage
cart = ShoppingCart()
cart.add_item("Laptop", 1200)
cart.add_item("Mouse", 25)
cart.add_item("Keyboard", 75)
cart.remove_item("Mouse") # Remove an item
cart.add_item("Monitor", 300) # Add another item

total_bill = cart.calculate_total()
print(f"Total bill: ${total_bill:.2f}")
```

OUTPUT :



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: `colab.research.google.com/drive/1e9tiCOdmZLz297yewijOegFF-6MNA8U#scrollTo=A6ZXOX3TQr3g`. The notebook is titled "Untitled5.ipynb". The left sidebar contains icons for file explorer, input/output, and code editor. The main area shows the following output:

```
Added Laptop to the cart.  
Added Mouse to the cart.  
Added Keyboard to the cart.  
Removed Mouse from the cart.  
Added Monitor to the cart.  
Applied 10% discount.  
Total bill: $1417.50
```

EXPLANATION :

The class stores items in a list of (name, price) pairs, with methods to add and remove items.

A loop iterates over all items to calculate the total.

If the total exceeds 500 or 1000, conditional discounts are applied using if-else logic.