Name: M.Sivateja

Hall Ticket: 2303A52145

Batch: 41

Task 1: Student Performance Evaluation System

Prompt:

Write a method to display student details

Write a method to check if marks are above class average and return a message

Code:

```
class Student:
    def __init__(self, name, roll_number, marks):
        # Initialize student attributes
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    # Method to display student details
    def display_details(self):
        print(f"Student Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")

    # Method to check performance against class average
    def check_performance(self, class_average):
        if self.marks > class_average:
            return f"{self.name} is performing above the class average."
        elif self.marks == class_average:
            return f"{self.name} is performing at the class average."
        else:
            return f"{self.name} is performing below the class average."

# --- Sample Output Generation ---

# 1. Instantiate students
students = [
    Student("Alice Smith", "S001", 85),
    Student("Bob Johnson", "S002", 72),
    Student("Charlie Brown", "S003", 90)
]

# 2. Calculate class average
total_marks = sum(s.marks for s in students)
avg_marks = total_marks / len(students)
print(f"--- Class Average: {avg_marks:.2f} ---\n")

# 3. Display details and performance for each student
for s in students:
    s.display_details()
    print(f"Status: {s.check_performance(avg_marks)}")
    print("-" * 30)
```

Output:

```
--- Class Average: 82.33 ---

Student Name: Alice Smith
Roll Number: S001
Marks: 85
Status: Alice Smith is performing above the class average.
------------------------------
Student Name: Bob Johnson
Roll Number: S002
Marks: 72
Status: Bob Johnson is performing below the class average.
------------------------------
Student Name: Charlie Brown
Roll Number: S003
Marks: 90
Status: Charlie Brown is performing above the class average.
------------------------------
```

Explanation:

The class stores basic student details as attributes while leaving the method logic incomplete. The comments guide GitHub Copilot to generate methods for showing student information and checking performance using the class average.

Task 2: Data Processing in a Monitoring System

Prompt:

Identify even sensor readings from a list, calculate the square of each even number, and print the result in a readable format inside a for loop.

Code:

```python
# List of sensor readings
sensor_readings = [12, 7, 20, 15, 8, 3, 22]

# Iterate over the sensor readings
for reading in sensor_readings:
    # GitHub Copilot Prompt:
    # 1. Identify even numbers using the modulus operator
    # 2. Calculate the square of the even numbers
    # 3. Print the result in a readable format
    if reading % 2 == 0:
        square = reading ** 2
        print(f"Sensor Reading: {reading} is Even. Squared result: {square}")
```

Output:

```
Sensor Reading: 12 is Even. Squared result: 144
Sensor Reading: 20 is Even. Squared result: 400
Sensor Reading: 8 is Even. Squared result: 64
Sensor Reading: 22 is Even. Squared result: 484
```

Explanation:

The loop processes each sensor reading individually, filters the even values, calculates their squares, and displays the results clearly. This demonstrates simple data processing using conditional checks and iteration.

3: Banking Transac on Simula on

Prompt:

Create a Python class named BankAccount with a ributes account_holder and balance. Add methods to deposit money and withdraw money, and ensure withdrawals are prevented when the balance is insufficient.

Code:

```
class BankAccount:
    def __init__(self, account_holder, balance=0.0):
        # Initialize account attributes
        self.account_holder = account_holder
        self.balance = balance

    # Method to deposit money
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Successfully deposited ${amount:.2f}. New balance: ${self.balance:.2f}")
        else:
            print("Deposit amount must be positive.")

    # Method to withdraw money with balance check
    def withdraw(self, amount):
        # Check if amount is valid and if there is enough balance
        if amount <= 0:
            print("Withdrawal amount must be positive.")
        elif amount > self.balance:
            print(f"Insufficient funds! Attempted to withdraw ${amount:.2f}, but current balance is ${self.balance:.2
        else:
            self.balance -= amount
            print(f"Successfully withdrew ${amount:.2f}. Remaining balance: ${self.balance:.2f}")

# --- Sample Output Generation ---
```

```
# 1. Create a bank account
my_account = BankAccount("John Doe", 500.0)
print(f"Account created for {my_account.account_holder} with balance ${my_account.balance:.2f}\n")

# 2. Perform a deposit
my_account.deposit(200.0)

# 3. Attempt a valid withdrawal
my_account.withdraw(150.0)

# 4. Attempt an invalid withdrawal (insufficient funds)
my_account.withdraw(1000.0)

# 5. Final Balance check
print(f"\nFinal Balance for {my_account.account_holder}: ${my_account.balance:.2f}")
```

Output:

```
Account created for John Doe with balance $500.00

Successfully deposited $200.00. New balance: $700.00
Successfully withdrew $150.00. Remaining balance: $550.00
Insufficient funds! Attempted to withdraw $1000.00, but current balance is $550.00.

Final Balance for John Doe: $550.00
```

Explanation:

The prompt guides Copilot to create a BankAccount class with essential attributes and methods for handling transactions. It supports safe operations by allowing deposits and stopping withdrawals when the balance is not enough.

Task 4: Student Scholarship Eligibility Check

Prompt:

Create a while loop that iterates through a list of student dictionaries and takes the scholarship cutoff score as user input.Print the names of students whose score is greater than the entered cutoff value.

Code:

```
# List of student dictionaries
student_records = [
    {"name": "Alice", "score": 82},
    {"name": "Bob", "score": 68},
    {"name": "Charlie", "score": 90},
    {"name": "David", "score": 74},
    {"name": "Eve", "score": 79}
]

# Initialize index for iteration
i = 0

print("--- Students Eligible for Scholarship (Score > 75) ---")

# GitHub Copilot Prompt:
# 1. Use a while loop to iterate through the student_records list
# 2. Check if the student's score is greater than 75
# 3. Print the name of the eligible student
# 4. Increment the index to avoid an infinite loop
while i < len(student_records):
    student = student_records[i]
    if student['score'] > 75:
        print(f"Eligible: {student['name']} (Score: {student['score']})")
    i += 1
```

Output:

```
--- Students Eligible for Scholarship (Score > 75) ---
Eligible: Alice (Score: 82)
Eligible: Charlie (Score: 90)
Eligible: Eve (Score: 79)
```

Explanation:

The program keeps student details in a list of dictionaries and asks the user to enter a cutoff score. It then uses a while loop to check each record and displays the names of students who scored higher than the given value.

Task 5: Online Shopping Cart Module

Prompt:

Create a Python class named ShoppingCart with an empty list to store cart items (name, price, quantity).

Add methods to add and remove items, calculate the total bill using a loop, and apply a discount conditionally when the total exceeds a specified amount.

Code:

```python
class ShoppingCart:
    def __init__(self):
        # Initialize an empty list to store items (name, price, quantity)
        self.items = []

    # Method to add an item to the cart
    def add_item(self, name, price, quantity):
        item = {"name": name, "price": price, "quantity": quantity}
        self.items.append(item)
        print(f"Added {quantity} x {name} to the cart.")

    # Method to remove an item from the cart by name
    def remove_item(self, name):
        for item in self.items:
            if item["name"].lower() == name.lower():
                self.items.remove(item)
                print(f"Removed {name} from the cart.")
                return
        print(f"Item {name} not found in cart.")

    # Method to calculate the total bill using a loop
    def calculate_total(self):
        total = 0
        for item in self.items:
            total += item["price"] * item["quantity"]
        return total

    # Method to apply conditional discounts
    def apply_discount(self, total):
        # GitHub Copilot Prompt: Apply a 10% discount if total exceeds $100
        if total > 100:
            discount = total * 0.10
            print(f"Discount Applied (10%): -${discount:.2f}")
            return total - discount
        return total

    def display_cart(self):
        print("\n--- Current Shopping Cart ---")
        if not self.items:
            print("Cart is empty.")
            return
        for item in self.items:
            print(f"{item['name']} - ${item['price']} x {item['quantity']}")

        raw_total = self.calculate_total()
        print(f"Subtotal: ${raw_total:.2f}")
        final_total = self.apply_discount(raw_total)
        print(f"Final Total: ${final_total:.2f}\n")
```

```python
# --- Sample Output Generation ---

# 1. Initialize cart and add items
my_cart = ShoppingCart()
my_cart.add_item("Laptop", 800.00, 1)
my_cart.add_item("Mouse", 25.00, 2)
my_cart.add_item("Sticker", 2.00, 5)

# 2. Display cart and total
my_cart.display_cart()

# 3. Remove an item and re-calculate
my_cart.remove_item("Mouse")
my_cart.display_cart()
```

Output:

```
Added 1 x Laptop to the cart.
Added 2 x Mouse to the cart.
Added 5 x Sticker to the cart.

--- Current Shopping Cart ---
Laptop - $800.0 x 1
Mouse - $25.0 x 2
Sticker - $2.0 x 5
Subtotal: $860.00
Discount Applied (10%): -$86.00
Final Total: $774.00

Removed Mouse from the cart.

--- Current Shopping Cart ---
Laptop - $800.0 x 1
Sticker - $2.0 x 5
Subtotal: $810.00
Discount Applied (10%): -$81.00
Final Total: $729.00
```

Explanation:

The prompt helps Copilot create a ShoppingCart class that uses a list to manage items. It handles adding and removing products, calculates the total using a loop, and applies discounts through conditional checks.