AI Assisted Coding-Lab 6.4

Shivani Pambidi

2303A52172

Batch – 41

Question 1:

TASK 1 : Student Performance Evaluation System PROMPT:

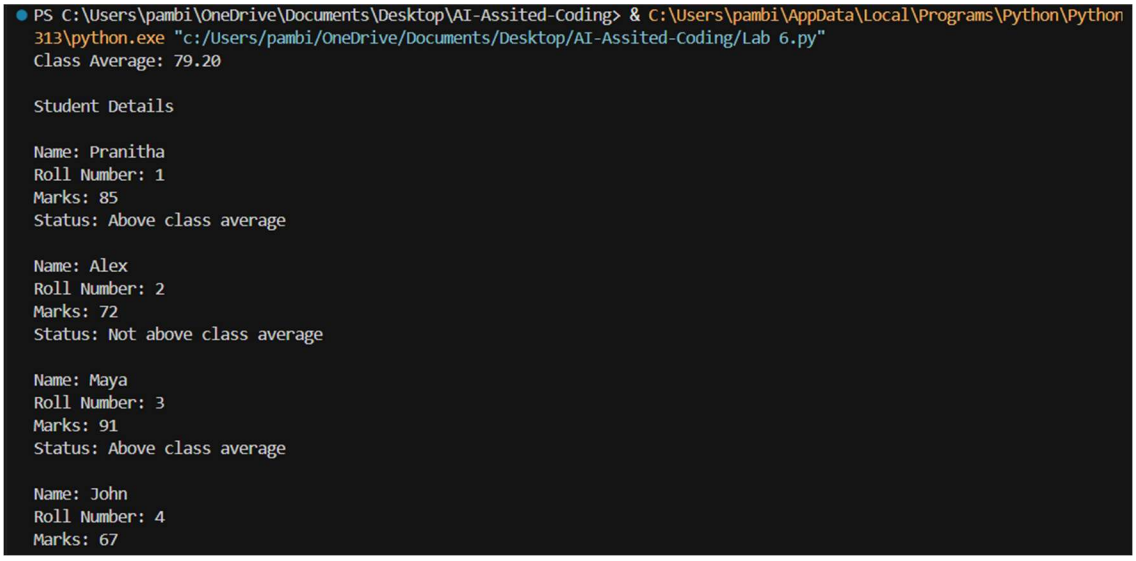
Create a method to display student details (name, roll number, marks).Create a method to compare student marks with class average.If marks are greater than average, return "Above Average".Otherwise, return "Below Average"

CODE:

```
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks
    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")
    def is_above_class_average(self, class_average):
        if self.marks > class_average:
            return True
        else:
            return False
# List of 10 students
students = [
    Student("Pranitha", 1, 85),
    Student("Alex", 2, 72),
    Student("Maya", 3, 91),
    Student("John", 4, 67),
    Student("Sara", 5, 78),
    Student("David", 6, 88),
    Student("Emma", 7, 95),
    Student("Chris", 8, 60),
    Student("Sophia", 9, 82),
    Student("Liam", 10, 74),
]
avg = sum(s.marks for s in students) / len(students)
print(f"Class Average: {avg:.2f}\n")
```

```
    Student("Maya", 3, 91),
    Student("John", 4, 67),
    Student("Sara", 5, 78),
    Student("David", 6, 88),
    Student("Emma", 7, 95),
    Student("Chris", 8, 60),
    Student("Sophia", 9, 82),
    Student("Liam", 10, 74),
]
avg = sum(s.marks for s in students) / len(students)
print(f"Class Average: {avg:.2f}\n")
print("Student Details\n")
for s in students:
    s.display_details()
    if s.is_above_class_average(avg):
        print("Status: Above class average\n")
    else:
        print("Status: Not above class average\n")
```

OUTPUT:



JUSTIFICATION :

This system helps to check student performance automatically in a fair way. Student details are stored properly using a class. Each student's data is handled separately to avoid mistakes. The program uses simple if–else conditions to compare marks with the class average. GitHub Copilot helps in writing the code faster and more easily. Overall, it saves time and reduces errors.

Question 2:

TASK 2: Data Processing in a Monitoring System

PROMPT:

Loop through the sensor readings list.Check if the reading is even using modulus operator.If even, calculate the square of the number.Print the even number and its square in a readable format.

CODE:

```
sensor_readings = [10, 15, 22, 33, 40, 55]
for reading in sensor_readings:
    # Check if the reading is even
    if reading % 2 == 0:
        square = reading * reading
        print(f"Even Reading: {reading}, Square: {square}")
```

OUTPUT:

```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Programs\Python\Python
313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab 6.py"
Even Reading: 10, Square: 100
Even Reading: 22, Square: 484
Even Reading: 40, Square: 1600
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding>
```

JUSTIFICATION:

This task helps process only useful sensor data by selecting even numbers. A for loop is used to go through the readings easily. The modulus operator identifies even values correctly. Conditional statements ensure only required data is processed. GitHub Copilot makes coding faster and simpler.

Question 3:

TASK 3: Banking Transaction Simulation

PROMPT:

Create a method to deposit money into the account.Add the deposit amount to the balance.Print a success message with updated balance.Create a method to withdraw money from the account.Check if balance is sufficient using if-else.If sufficient, subtract amount and print success message.Else, print insufficient balance message

CODE:

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

    # Method to deposit money
    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited: ₹{amount}")
        print(f"Updated Balance: ₹{self.balance}")

    # Method to withdraw money
    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrawn: ₹{amount}")
            print(f"Remaining Balance: ₹{self.balance}")
        else:
            print("Withdrawal failed: Insufficient balance")
```

OUTPUT:

```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Programs\Python\Python
313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab 6.py"
Deposited: ₹2000
Updated Balance: ₹7000
Withdrawn: ₹3000
Remaining Balance: ₹4000
Withdrawal failed: Insufficient balance
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding>
```

JUSTIFICATION:

This banking module helps manage customer accounts safely and easily. It stores account holder details and balance using a class structure. Deposit and withdrawal methods make money handling simple and organized. The use of if–else conditions prevents withdrawals when there is not enough balance. GitHub Copilot helps generate clear and user-friendly code quickly.

Question 4:

TASK 4: Student Scholarship Eligibility Check

PROMPT:

Use a while loop to go through the student list.Check if the student's score is greater than 75.If eligible, print the student's name.Increase the index to avoid infinite loop

CODE:

```
# List of students (initialization written manually)
students = [
    {"name": "Shivani", "score": 88},
    {"name": "Rahul", "score": 72},
    {"name": "Ananya", "score": 91},
    {"name": "Kiran", "score": 65},
    {"name": "Meena", "score": 79}
]

# Use a while loop to iterate through the student list
# Check if the student's score is greater than 75
# If yes, print the student's name as eligible
# Increment the index after each iteration

i = 0
print("Eligible Students for Scholarship:")

while i < len(students):
    if students[i]["score"] > 75:
        print(students[i]["name"])
    i += 1
```

OUTPUT:

```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Programs\Python\Python
313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab 6.py"
Eligible Students for Scholarship:
Shivani
Ananya
Meena
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding>
```

JUSTIFICATION:

This task helps identify students eligible for a merit scholarship easily. Student details are stored in dictionaries, making the data clear and organized. A while loop is used to go through each student one by one. Conditional checks ensure only students with scores above 75 are selected. Using GitHub Copilot makes writing the loop faster and reduces logical errors.

Question 5:

TASK 5: Online Shopping Cart Module

PROMPT:

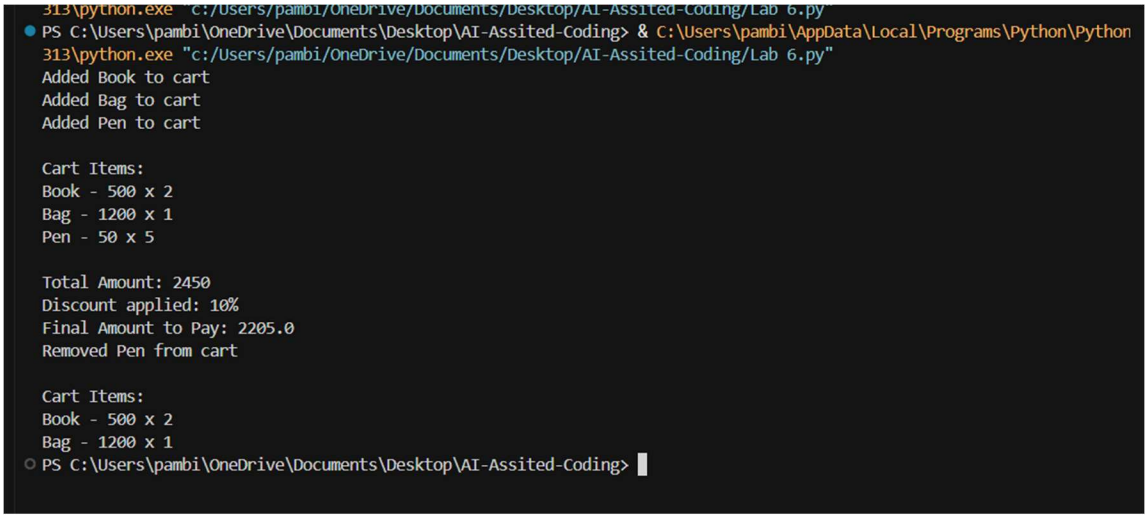
Initialize an empty list to store cart items.Create a method to add an item (name, price, quantity) to the cart.Create a method to remove an item from the cart by name.Create a method to calculate the total bill using a loop.Apply a discount if the total amount exceeds a certain value.Print the final amount in a readable format

CODE:

```
class ShoppingCart:
    def __init__(self):
        # Empty list to store items
        self.items = []
    def add_item(self, name, price, quantity):
        # Add an item to the cart
        self.items.append({
            "name": name,
            "price": price,
            "quantity": quantity
        })
        print(f"Added {name} to cart")
    def remove_item(self, name):
        # Remove item from cart by name
        for item in self.items:
            if item["name"] == name:
                self.items.remove(item)
                print(f"Removed {name} from cart")
                return
        print("Item not found")
    def calculate_total(self):
        # Calculate total bill using loop
        total = 0
        for item in self.items:
            total += item["price"] * item["quantity"]
        return total
    def apply_discount(self, total):
        # Apply conditional discount
        if total > 2000:
```

```
        if total > 2000:
            print("Discount applied: 10%")
            total = total - (total * 0.10)
        else:
            print("No discount applied")
        return total
    def display_cart(self):
        # Display cart items
        print("\nCart Items:")
        for item in self.items:
            print(item["name"], "-", item["price"], "x", item["quantity"])
# ---------- Execution / Sample Input ----------
cart = ShoppingCart()
cart.add_item("Book", 500, 2)
cart.add_item("Bag", 1200, 1)
cart.add_item("Pen", 50, 5)
cart.display_cart()
total = cart.calculate_total()
print("\nTotal Amount:", total)
final_amount = cart.apply_discount(total)
print("Final Amount to Pay:", final_amount)
cart.remove_item("Pen")
cart.display_cart()
```

OUTPUT:



JUSTIFICATION:

This shopping cart module helps manage items in an online store easily. Items are stored in a list, which makes adding and removing products simple. Loops are used to calculate the total bill accurately. Conditional statements apply discounts only when required. GitHub Copilot helps generate clean and correct code faster, reducing mistakes.