2303A52008

BATCH 43

WEEK 6.4

Task 1:

Code:

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

    def display_student_details(self):
        """Display student information"""
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")

    def check_performance(self, class_average):
        """Check if stude
        if self.marks >= class_average:
            return f"{self.name} performed above average with {self.marks} marks."
        else:
            return f"{self.name} performed below average. Score: {self.marks}, Average:
```

(parameter) class_average: Any

```python
if __name__ == "__main__":
    student1 = Student("Alice", "S001", 85)
    student2 = Student("Bob", "S002", 72)

    class_avg = 78

    student1.display_student_details()
    print(student1.check_performance(class_avg))
    print()

    student2.display_student_details()
    print(student2.check_performance(class_avg))
```

Output:

```
Name: Alice
Roll Number: S001
Marks: 85
Alice performed above average with 85 marks.
Could not find platform independent libraries <prefix>
Name: Alice
Roll Number: S001
Marks: 85
Alice performed above average with 85 marks.
Name: Alice
Roll Number: S001
Marks: 85
Alice performed above average with 85 marks.
Roll Number: S001
Marks: 85
Alice performed above average with 85 marks.
Alice performed above average with 85 marks.

Name: Bob
Name: Bob
Roll Number: S002
Marks: 72
Bob performed below average. Score: 72, Average: 78
```

Summary:

This program defines a basic `Student` class used to evaluate academic performance automatically. The class stores student details such as name, roll number, and marks using instance attributes. GitHub Copilot is guided through comments to generate methods for displaying details and evaluating performance. Conditional statements are used to compare student marks with the class average. The system outputs both student information and a clear performance status message.

TASK 2: Data Processing in a Monitoring System

CODE:

```
34  # Task 2: Data Processing in a Monitoring System
35  sensor_readings = [12, 15, 24, 7, 36, 19, 42, 8]
36  for reading in sensor_readings:
37      if reading % 2 == 0:
38          square = reading ** 2
39          print(f"Sensor Reading: {reading} | Square: {square}")
40
```

Output:

```
Sensor Reading: 12 | Square: 144
Sensor Reading: 24 | Square: 576
Sensor Reading: 36 | Square: 1296
Sensor Reading: 42 | Square: 1764
Sensor Reading: 8 | Square: 64
```

Summary:

This task demonstrates a simple data monitoring loop where sensor readings are processed selectively. A `for` loop iterates through a list of integer readings, and GitHub Copilot is guided using comments to identify even numbers with the modulus operator. Conditional statements ensure only valid even readings are processed further. For each even number, its square is calculated and displayed in a clear, readable format. Overall, the example shows how Copilot can complete logical code blocks efficiently using well-written prompts.

TASK 3:

Code:

```
#-----------------------------3-----------------------------#
# Banking Transaction Simulation
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance
    def deposit(self, amount):
        self.balance += amount
        print(f"Deposit successful. New balance: {self.balance}")
    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrawal successful. Remaining balance: {self.balance}")
        else:
            print("Withdrawal failed: Insufficient balance")
# Sample usage
account = BankAccount("Vikas", 5000)

account.deposit(2000)
account.withdraw(3000)
account.withdraw(5000)
```

Output:

```
Deposit successful. New balance: 7000
Withdrawal successful. Remaining balance: 4000
Withdrawal failed: Insufficient balance
```

SUMMARY:

This program simulates basic banking transactions using a `BankAccount` class. The class stores customer details and account balance using instance attributes. GitHub Copilot is guided through method names and comments to generate deposit and withdrawal logic. Conditional statements ensure withdrawals are allowed only when sufficient balance is available. The system provides clear, user-friendly messages for both successful and failed transactions.

TASK 4: Student Scholarship Eligibility Check

```
#-----------------------------4-----------------------------#
# Student Scholarship Eligibility Check

students = [
    {"name": "Vikas", "score": 82},
    {"name": "Ananya", "score": 74},
    {"name": "Rahul", "score": 90},
    {"name": "Sneha", "score": 68},
    {"name": "Arjun", "score": 78}
]

index = 0
while index < len(students):
    if students[index]["score"] > 75:
        print(students[index]["name"], "is eligible for the scholarship")
    index += 1
```

Output:

```
Withdrawal failed: Insufficient balance
Vikas is eligible for the scholarship
Rahul is eligible for the scholarship
Arjun is eligible for the scholarship
```

Summary:

This program identifies students eligible for a merit-based scholarship using a list of dictionaries. Each dictionary stores a student's name and score. GitHub Copilot is guided through comments to generate a `while` loop for iteration and condition checking. Proper index handling ensures all students are evaluated correctly. The output clearly lists only those students who meet the eligibility criteria.

Task 5:

```python
#-----------------------------5-----------------------------#
# Online Shopping Cart Module

class ShoppingCart:
    def __init__(self):
        self.items = []
    def add_item(self, name, price, quantity):
        self.items.append({"name": name, "price": price, "quantity": quantity})
        print(f"{name} added to cart")
    def remove_item(self, name):
        for item in self.items:
            if item["name"] == name:
                self.items.remove(item)
                print(f"{name} removed from cart")
                return
        print("Item not found in cart")
    def calculate_total(self):
        total = 0
        for item in self.items:
            total += item["price"] * item["quantity"]

        if total > 5000:
            discount = total * 0.10
            total -= discount
            print(f"Discount applied: ₹{discount}")
```

```python

cart = ShoppingCart()

cart.add_item("Laptop", 45000, 1)
cart.add_item("Mouse", 500, 2)
cart.remove_item("Mouse")

final_amount = cart.calculate_total()
print("Final amount to pay: ₹", final_amount)
```

Output:

```
Laptop added to cart
Mouse added to cart
Mouse removed from cart
Discount applied: ₹4500.0
Final amount to pay: ₹ 40500.0
```

Summary:

This program implements a simplified online shopping cart using a `ShoppingCart` class. The cart maintains a list of items with name, price, and quantity details. GitHub Copilot is guided through comments to generate methods for adding and removing items. A loop is used to calculate the total bill, and conditional logic applies a discount when the total exceeds a set amount. The sample output demonstrates correct cart operations and discount handling.