

T. VIKAS

2303A52281

BATCH 43

WEEK 6.4

Task 1:

Code:

```
week6.4.py >...
1  class Student:
2      def __init__(self, name, roll_number, marks):
3          self.name = name
4          self.roll_number = roll_number
5          self.marks = marks
6
7      def display_student_details(self):
8          """Display student information"""
9          print(f"Name: {self.name}")
10         print(f"Roll Number: {self.roll_number}")
11         print(f"Marks: {self.marks}")
12
13     def check_performance():
14         """Check if student (parameter) class_average: Any"""
15         if self.marks >= class_average:
16             return f"{self.name} performed above average with {self.marks} marks."
17         else:
18             return f"{self.name} performed below average. Score: {self.marks}, Average: {class_average}"
19
20
21 if __name__ == "__main__":
22     student1 = Student("Alice", "S001", 85)
23     student2 = Student("Bob", "S002", 72)
24
25     class_avg = 78
26
27     student1.display_student_details()
28     print(student1.check_performance(class_avg))
29     print()
30
31     student2.display_student_details()
32     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:**

```
41 #-----#
42 # Banking Transaction Simulation
43 class BankAccount:
44     def __init__(self, account_holder, balance):
45         self.account_holder = account_holder
46         self.balance = balance
47     def deposit(self, amount):
48         self.balance += amount
49         print(f"Deposit successful. New balance: {self.balance}")
50     def withdraw(self, amount):
51         if amount <= self.balance:
52             self.balance -= amount
53             print(f"Withdrawal successful. Remaining balance: {self.balance}")
54         else:
55             print("Withdrawal failed: Insufficient balance")
56 # Sample usage
57 account = BankAccount("vikas", 5000)
58
59 account.deposit(2000)
60 account.withdraw(3000)
61 account.withdraw(5000)
62
```

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

```
63 #-----#
64 # Student Scholarship Eligibility Check
65
66 students = [
67     {"name": "Vikas", "score": 82},
68     {"name": "Ananya", "score": 74},
69     {"name": "Rahul", "score": 90},
70     {"name": "Sneha", "score": 68},
71     {"name": "Arjun", "score": 78}
72 ]
73
74 index = 0
75 while index < len(students):
76     if students[index]["score"] > 75:
77         print(students[index]["name"], "is eligible for the scholarship")
78     index += 1
```

Output:

```
Mehar and Rahul: 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:

```
79  #-----#
80  # Online Shopping Cart Module
81
82  class ShoppingCart:
83      def __init__(self):
84          self.items = []
85      def add_item(self, name, price, quantity):
86          self.items.append({"name": name, "price": price, "quantity": quantity})
87          print(f"{name} added to cart")
88      def remove_item(self, name):
89          for item in self.items:
90              if item["name"] == name:
91                  self.items.remove(item)
92                  print(f"{name} removed from cart")
93                  return
94          print("Item not found in cart")
95      def calculate_total(self):
96          total = 0
97          for item in self.items:
98              total += item["price"] * item["quantity"]
99
100         if total > 5000:
101             discount = total * 0.10
102             total -= discount
103             print(f"Discount applied: {discount}")
104
105
106     cart = ShoppingCart()
107
108     cart.add_item("Laptop", 45000, 1)
109     cart.add_item("Mouse", 500, 2)
110     cart.remove_item("Mouse")
111
112
113     final_amount = cart.calculate_total()
114     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.