Name :V.Shiva krishna

Hall Ticket No :2303a52151

Batch-41

Task 1: Student Performance Evaluation System

Scenario

You are building a simple academic management module for a university

system where student performance needs to be evaluated automatically.

Task Description

Create the skeleton of a Python class named Student with the attributes:

• name

• roll_number

• marks

Write only the class definition and attribute initialization.

Then, using GitHub Copilot, prompt the tool to complete:

• A method to display student details

• A method that checks whether the student's marks are above the

class average and returns an appropriate message

Use comments or partial method names to guide Copilot for code

Completion.

Prompt:

Generate a Python program with student name ,roll no,and marks add two methods

Two prints the student details and Performance through the overall class students to return

A message above or at or below performance through the loops.

CODE:

```python
"""
Student Performance Evaluation System

This script defines a Student class and a simple CLI flow that:
- Collects N students from user input.
- Calculates the class average.
- Displays each student's details and whether they are above the average.

The display and performance methods were intended as Copilot-completed
sections; comments below mark where completions would have been guided.
"""
```

> (variable) annotations: _Feature

```python
from __future__ import annotations


class Student:
    def __init__(self, name: str, roll_number: str, marks: float) -> None:
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    # Copilot hint: def display_details(self): ...
    def display_details(self) -> None:
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks:.2f}")

    # Copilot hint: def performance_status(self, class_average): ...
    def performance_status(self, class_average: float) -> str:
        if self.marks > class_average:
            return "Above class average"
        elif self.marks == class_average:
            return "Exactly at class average"
        return "Below class average"
```

```python
def collect_students() -> list[Student]:
    while True:
        try:
            total = int(input("How many students? "))
            if total <= 0:
                print("Enter a positive number.")
                continue
            break
        except ValueError:
            print("Please enter a valid integer.")

    students: list[Student] = []
    for idx in range(1, total + 1):
        print(f"\nStudent {idx} details:")
        name = input("  Name: ").strip() or f"Student {idx}"
        roll_number = input("  Roll Number: ").strip() or f"R{idx:03d}"
        while True:
            try:
                marks = float(input("  Marks (0-100): "))
                if 0 <= marks <= 100:
                    break
                print("  Marks must be between 0 and 100.")
            except ValueError:
                print("  Please enter a numeric value.")
        students.append(Student(name, roll_number, marks))

    return students


def compute_average(students: list[Student]) -> float:
    if not students:
        return 0.0
    return sum(s.marks for s in students) / len(students)


def compute_average(students: list[Student]) -> float:
    if not students:
        return 0.0
    return sum(s.marks for s in students) / len(students)


def main() -> None:
    students = collect_students()
    class_average = compute_average(students)

    print(f"\nClass average: {class_average:.2f}\n")
    for student in students:
        student.display_details()
        status = student.performance_status(class_average)
        print(f"Performance: {status}\n")

    query = input("Enter a student's name or roll number to check against the class average: ").strip().lc
    if query:
        match = next((s for s in students if s.name.lower() == query or s.roll_number.lower() == query), N
        if match:
            status = match.performance_status(class_average)
            print(f"{match.name} ({match.roll_number}) is {status}.")
        else:
            print("No student found with that name or roll number.")
    else:
        print("No lookup provided; skipping individual check.")


if __name__ == "__main__":
    main()
```

OUTPUT:

```
PS C:\Users\ksair\Downloads\Competitive Programing> & C:/Users/ksair/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Use
/Competitive Programing/assignment-6.py"
How many students? 3

Student 1 details:
  Name: sai kumar
  Roll Number: 52223
  Marks (0-100): 85

Student 2 details:
  Name: abhinav
  Roll Number: 2486
  Marks (0-100): 90

Student 3 details:
  Name: shashidhar
  Roll Number: 2291
  Marks (0-100): 96

Class average: 90.33

Name: sai kumar
Roll Number: 52223
Marks: 85.00
Performance: Below class average

Name: abhinav
Roll Number: 52223
Marks: 85.00
Performance: Below class average
```

```
Enter a student's name or roll number to check against the class average: 2291
Name: abhinav
Roll Number: 2486
Marks: 90.00
Performance: Below class average

Name: shashidhar
Roll Number: 2291
Marks: 96.00
Performance: Above class average

Enter a student's name or roll number to check against the class average: 2291

Name: shashidhar
Roll Number: 2291
Marks: 96.00
Performance: Above class average
```

JUSTIFICATION:

The code meets the task by defining student with display and average-check methods using if/elif/else, gathering N students with validation, computing the class average once, printing each student's details and status, and allowing a final lookup by name or roll number.

Task 2: Data Processing in a Monitoring System

Scenario

You are working on a basic data monitoring script where sensor readings

are collected as numbers. Only even readings need further processing.

Task Description

Write the initial part of a for loop to iterate over a list of integers

representing sensor readings.

Add a comment prompt instructing GitHub Copilot to:

• Identify even numbers

• Calculate their square

• Print the result in a readable format

Allow Copilot to complete the remaining loop logic.

Prompt:

Check if it's even using the modulus operator. If it is even, go ahead and calculate the square of that number. Then print it out nicely so I can see the original reading and its squared value. Oh, and if it's odd, just print a message saying we skipped it. Make sure everything looks clean and readable.

Code:

```python
def task2_sensor_monitoring():
    """
    Task 2: Data Processing in a Monitoring System
    Process sensor readings and identify even numbers
    """
    print("\n" + "="*60)
    print("TASK 2: DATA PROCESSING IN A MONITORING SYSTEM")
    print("="*60)

    # Get sensor readings from user
    readings_input = input("Enter sensor readings (comma-separated numbers): ")
    sensor_readings = [int(x.strip()) for x in readings_input.split(',')]

    print(f"\nProcessing {len(sensor_readings)} sensor readings...")
    print("-" * 60)

    # Iterate over sensor readings
    for reading in sensor_readings:
        # GitHub Copilot Prompt:
        # Check if the reading is even using modulus operator
        # If even, calculate its square
        # Print the result in a readable format
        if reading % 2 == 0:
            squared_value = reading ** 2
            print(f"Sensor Reading: {reading} (Even) → Square: {squared_value}")
        else:
            print(f"Sensor Reading: {reading} (Odd) → Skipped")

    print("-" * 60)
    print("✓ Sensor data processing complete!\n")
```

Output:

```
Enter your choice (2-5 or 0 to exit): 2

============================================================
TASK 2: DATA PROCESSING IN A MONITORING SYSTEM
============================================================
Enter sensor readings (comma-separated numbers): 2,5,7,8,9,10,15,30

Processing 8 sensor readings...
------------------------------------------------------------
Sensor Reading: 2 (Even) → Square: 4
Sensor Reading: 5 (Odd) → Skipped
Sensor Reading: 7 (Odd) → Skipped
Sensor Reading: 8 (Even) → Square: 64
Sensor Reading: 9 (Odd) → Skipped
Sensor Reading: 10 (Even) → Square: 100
Sensor Reading: 15 (Odd) → Skipped
Sensor Reading: 30 (Even) → Square: 900
------------------------------------------------------------
/ Sensor data processing complete!
```

Justification:

Loop through each sensor reading in the list. For every reading, check if it's an even number by using the modulus operator (reading % 2 == 0). If the reading is even, calculate its square using the exponentiation operator (reading ** 2) and print both the original reading and its squared value in a clean, readable format. If the reading is odd, print a message indicating that it was skipped and won't be processed further. Make sure all output includes clear labels so users can easily understand which readings were processed and which were ignored.

Task 3: Banking Transaction Simulation

Scenario

You are developing a basic banking module that handles deposits and withdrawals for customers.

Task Description

Create the structure of a Python class named Bank Account with attributes:

- Account holder

- balance

Use GitHub Copilot to complete methods for:

- Depositing money

- Withdrawing money

- Preventing withdrawals when the balance is insufficient

Prompt:

I'm building a Bank Account class here. It needs account holder and balance attributes. Can you help me create a deposit method that takes an amount, checks if it's positive, adds it to the balance using self, and prints a nice success message with the new balance.Also need a withdraw method that checks if there's enough money in the account before taking it out. If the balance is too low, show an error message saying insufficient funds. Use if-else for the balance check and work with self balance throughout.

Code:

```python
def task3_banking_simulation():
    """
    Task 3: Banking Transaction Simulation
    Handle deposits and withdrawals with balance validation
    """
    print("\n" + "="*60)
    print("TASK 3: BANKING TRANSACTION SIMULATION")
    print("="*60)

    class BankAccount:
        """
        BankAccount class to manage customer banking operations
        """
        def __init__(self, account_holder, balance=0):
            self.account_holder = account_holder
            self.balance = balance

        # GitHub Copilot Prompt:
        # Method to deposit money to the account
        # Add the amount to balance and display success message
        def deposit(self, amount):
            if amount > 0:
                self.balance += amount
                print(f"✓ Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")
            else:
                print("X Invalid deposit amount. Must be positive.")

        # GitHub Copilot Prompt:
        # Method to withdraw money from the account
        # Check if sufficient balance exists before withdrawal
        # If insufficient, display error message
```

```python
# GitHub Copilot Prompt:
# Method to withdraw money from the account
# Check if sufficient balance exists before withdrawal
# If insufficient, display error message
def withdraw(self, amount):
    if amount > 0:
        if self.balance >= amount:
            self.balance -= amount
            print(f"✓ Withdrew ${amount:.2f}. Remaining balance: ${self.balance:.2f}")
        else:
            print(f"X Insufficient balance! Available: ${self.balance:.2f}, Requested: ${amount:.2f}")
    else:
        print("X Invalid withdrawal amount. Must be positive.")

def display_balance(self):
    print(f"\n💰 Account Holder: {self.account_holder}")
    print(f"💰 Current Balance: ${self.balance:.2f}\n")

# User interaction
holder_name = input("\nEnter account holder name: ")
initial_balance = float(input("Enter initial balance: $"))

account = BankAccount(holder_name, initial_balance)
account.display_balance()

while True:
    print("\n--- Banking Menu ---")
    print("1. Deposit")
    print("2. Withdraw")
    print("3. Check Balance")
    print("4. Exit Banking")

    choice = input("\nSelect operation (1-4): ")
```

Output:

```
(.venv) PS C:\Users\ksair\Downloads\Competitive Programing> & "c:/Users/ksair/Downloads/Competitive Programing/.venv/Scr
ownloads/Competitive Programing/assignment-6.py"
=========================================================
TASK 3: BANKING TRANSACTION SIMULATION
=========================================================

Enter account holder name: sai kumar
Enter initial balance: $1000

💰 Account Holder: sai kumar
💰 Current Balance: $1000.00


--- Banking Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Exit Banking

Select operation (1-4): 1
Enter deposit amount: $5000
√ Deposited $5000.00. New balance: $6000.00

--- Banking Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Exit Banking

Select operation (1-4): 2
Enter withdrawal amount: $500
√ Withdrew $500.00. Remaining balance: $5500.00

--- Banking Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Exit Banking

Select operation (1-4): 3
```

```
Select operation (1-4): 2
Enter withdrawal amount: $500
√ Withdrew $500.00. Remaining balance: $5500.00

--- Banking Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Exit Banking

Select operation (1-4): 3

💰 Account Holder: sai kumar
💰 Current Balance: $5500.00


--- Banking Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Exit Banking
```

Justification:

Create a method named deposit that accepts an amount parameter. First validate that the amount is positive. If valid, add the amount to self.balance using the += operator and print a success message showing the deposited amount and the new balance formatted to 2 decimal places. If the amount is not positive, display an error message asking for a positive value.Create a method named withdraw that accepts an amount parameter. First check if the amount is positive. Then use an if-else statement to verify whether self.balance is greater than or equal to the requested amount. If sufficient balance exists, subtract the amount from self.balance and print a success message with the withdrawn amount .

Task 4: Student Scholarship Eligibility Check

Scenario: A university wants to identify students eligible for a merit-based scholarship based on their scores.

Task Description

Define a list of dictionaries where each dictionary represents a student with:

- name

- score

Write the initialization and list structure yourself.
Then, prompt GitHub Copilot to generate a **while loop** that:

- Iterates through the list

- Prints the names of students who scored **more than 75**

Prompt:

I need a while loop to go through this students list. Start with index at 0, then in each loop get the current student using students[index]. Check if their score is more than 75. If yes, print their name and say they're eligible for the scholarship. If not, print they're not eligible. Remember to increment the index each time and keep looping while index is less than the length of students. Also keep track of how many students are eligible so I can show a total count at the end.

Code:

```python
def task4_scholarship_eligibility():
    """
    Task 4: Student Scholarship Eligibility Check
    Identify students eligible for merit-based scholarship
    """
    print("\n" + "="*60)
    print("TASK 4: STUDENT SCHOLARSHIP ELIGIBILITY CHECK")
    print("="*60)

    # Get number of students
    num_students = int(input("\nEnter number of students: "))

    # Define list of student dictionaries
    students = []

    print("\nEnter student details:")
    for i in range(num_students):
        print(f"\n--- Student {i+1} ---")
        name = input("Name: ")
        score = float(input("Score: "))
        students.append({"name": name, "score": score})

    print("\n" + "-" * 60)
    print("SCHOLARSHIP ELIGIBILITY RESULTS (Score > 75)")
    print("-" * 60)

    # GitHub Copilot Prompt:
    # Use a while loop to iterate through the students list
    # Check if each student's score is greater than 75
    # Print the names of eligible students in a formatted way

# GitHub Copilot Prompt:
# Use a while loop to iterate through the students list
# Check if each student's score is greater than 75
# Print the names of eligible students in a formatted way
index = 0
eligible_count = 0

while index < len(students):
    student = students[index]
    if student["score"] > 75:
        print(f"√ {student['name']} - Score: {student['score']} (ELIGIBLE)")
        eligible_count += 1
    else:
        print(f"X {student['name']} - Score: {student['score']} (Not Eligible)")
    index += 1

print("-" * 60)
print(f"Total Eligible Students: {eligible_count} out of {num_students}\n")
```

Output:

```
(.venv) PS C:\Users\ksair\Downloads\Competitive Programing> & "c:/Users/ksair/Downloads/Competitive Programing/.venv/S
ownloads/Competitive Programing/assignment-6.py"
========================================================
TASK 4: STUDENT SCHOLARSHIP ELIGIBILITY CHECK
========================================================

Enter number of students: 2

Enter student details:

--- Student 1 ---
Name: sai kumar
Score: 87

--- Student 2 ---
Name: kruthik
Score: 90


------------------------------------------------------------
SCHOLARSHIP ELIGIBILITY RESULTS (Score > 75)
------------------------------------------------------------
√ sai kumar - Score: 87.0 (ELIGIBLE)
√ kruthik - Score: 90.0 (ELIGIBLE)
------------------------------------------------------------
Total Eligible Students: 2 out of 2
```

Justification:

This approach uses a while loop with an index to manually control iteration over the students list, ensuring clear access to each student record using students[index]. The if condition cleanly separates eligible and non-eligible students based on the score threshold of 75, making the decision logic easy to understand. Maintaining the eligible_count variable allows accurate tracking of qualified students during iteration, and displaying the final count after the loop provides a clear summary of results. Incrementing the index in each iteration prevents infinite loops and ensures all applicants are processed systematically.

Task 5: Online Shopping Cart Module

Scenario

You are designing a simplified shopping cart system for an e-commerce website that supports item management and discount calculation.

Task Description

Begin writing a Python class named Shopping cart with:

- An empty list to store items (each item may include name, price, quantity)

Use GitHub Copilot to generate methods that:

- Add items to the cart

- Remove items from the cart

- Calculate the total bill using a loop

- Apply conditional discounts (e.g., discount if total exceeds a certain amount)

Prompt:

Create a Shopping cart class with an empty items list. Add three methods:
add item(name, price, quantity) to store an item dict and confirm addition;
remove item(name) to find and remove an item case-insensitively with success/error
messages, calculate total() to print a receipt showing each item, compute subtotal, apply a
10% discount if subtotal > 100, and display subtotal, discount (if any), and final total
formatted to 2 decimals.

Code:

```python
def task5_shopping_cart():
    """
    Task 5: Online Shopping Cart Module
    Manage shopping cart with item addition, removal, and discount calculation
    """
    print("\n" + "="*60)
    print("TASK 5: ONLINE SHOPPING CART MODULE")
    print("="*60)

    class ShoppingCart:
        """
        ShoppingCart class to manage e-commerce cart operations
        """
        def __init__(self):
            # Initialize empty list to store items
            self.items = []
            item = {
                "name": name,
                "price": price,
                "quantity": quantity
            }
            self.items.append(item)
            print(f"✓ Added: {quantity}x {name} @ ${price:.2f} each")

        def remove_item(self, name):
            for item in self.items:
                if item["name"].lower() == name.lower():
                    self.items.remove(item)
                    print(f"✓ Removed: {item['name']}")
                    return
            print(f"X Item '{name}' not found in cart")
```

```python
    def calculate_total(self, discount_threshold=100, discount_percent=10):
        if len(self.items) == 0:
            print("\nX Cart is empty!")
            return 0

        print("\n" + "="*50)
        print("SHOPPING CART SUMMARY")
        print("="*50)

        subtotal = 0

        # Calculate subtotal using loop
        for item in self.items:
            item_total = item["price"] * item["quantity"]
            subtotal += item_total
            print(f"{item['name']:<20} {item['quantity']:>3} x ${item['price']:>6.2f} = ${item_total:>8.2f}")

        print("-" * 50)
        print(f"{'Subtotal:':<20} ${subtotal:>8.2f}")

        # Apply conditional discount
        discount_amount = 0
        if subtotal > discount_threshold:
            discount_amount = subtotal * (discount_percent / 100)
            print(f"{'Discount (' + str(discount_percent) + '%):':<20} -${discount_amount:>7.2f}")
            print(f"   (Applied: Total > ${discount_threshold})")

        final_total = subtotal - discount_amount
        print("="*50)
        print(f"{'TOTAL:':<20} ${final_total:>8.2f}")
        print("="*50 + "\n")

        return final_total

    def display_cart(self):
        if len(self.items) == 0:
            print("\n🛒 Cart is empty")
        else:
            print(f"\n🛒 Cart contains {len(self.items)} item(s):")
            for i, item in enumerate(self.items, 1):
                print(f"   {i}. {item['name']} - ${item['price']:.2f} x {item['quantity']}")
        print()

# User interaction
cart = ShoppingCart()

while True:
    print("\n--- Shopping Cart Menu ---")
    print("1. Add Item")
    print("2. Remove Item")
    print("3. View Cart")
    print("4. Calculate Total & Checkout")
    print("5. Exit Shopping")

    choice = input("\nSelect operation (1-5): ")

    if choice == '1':
        name = input("Enter item name: ")
        price = float(input("Enter item price: $"))
        quantity = int(input("Enter quantity: "))
        cart.add_item(name, price, quantity)
    elif choice == '2':
        name = input("Enter item name to remove: ")
        cart.remove_item(name)
    elif choice == '3':
        cart.display_cart()
    elif choice == '4':
        cart.calculate_total()
```

Output:

```
============================================================
TASK 5: ONLINE SHOPPING CART MODULE
============================================================

--- Shopping Cart Menu ---
1. Add Item
2. Remove Item
3. View Cart
4. Calculate Total & Checkout
5. Exit Shopping

Select operation (1-5): 1
Enter item name: rice
Enter item price: $50
Enter quantity: 1
√ Added: 1x rice @ $50.00 each

--- Shopping Cart Menu ---
1. Add Item
2. Remove Item
3. View Cart
4. Calculate Total & Checkout
5. Exit Shopping

Select operation (1-5): 1
Enter item name: oil
Enter item price: $160
Enter quantity: 1
√ Added: 1x oil @ $160.00 each
```

```
-- Shopping Cart Menu ---
. Add Item
. Remove Item
. View Cart
. Calculate Total & Checkout
. Exit Shopping

elect operation (1-5): 1
nter item name: eggs
nter item price: $6
nter quantity: 1
Added: 1x eggs @ $6.00 each

-- Shopping Cart Menu ---
. Add Item
. Remove Item
. View Cart
. Calculate Total & Checkout
. Exit Shopping

elect operation (1-5): 2
nter item name to remove: eggs
Removed: eggs

-- Shopping Cart Menu ---
. Add Item
. Remove Item
. View Cart
. Calculate Total & Checkout
. Exit Shopping
```

```
--- Shopping Cart Menu ---
1. Add Item
2. Remove Item
3. View Cart
4. Calculate Total & Checkout
5. Exit Shopping

Select operation (1-5): 3

▤ Cart contains 2 item(s):
  1. rice - $50.00 x 1
  2. oil - $160.00 x 1


--- Shopping Cart Menu ---
1. Add Item
2. Remove Item
3. View Cart
4. Calculate Total & Checkout
5. Exit Shopping

Select operation (1-5): 4


=================================================
SHOPPING CART SUMMARY
=================================================
rice                    1 x $ 50.00 = $    50.00
oil                     1 x $160.00 = $   160.00
-------------------------------------------------
Subtotal:           $  210.00
Discount (10%):     -$   21.00
  (Applied: Total > $100)
=================================================
TOTAL:              $  189.00
```

Justification:

The Shopping cart class uses a list to manage items easily. Adding and removing items lets users update their cart. A loop ensures accurate total calculation based on price and quantity. Conditional discounts mimic real e-commerce offers and encourage higher spending.