

Assignment-6.4

NAME : Ch.Adarsh

HTNO : 2303A51044

BATCH : 16

Task-01: 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.

Expected Outcome

- A completed **Student** class with Copilot-generated methods
- Proper use of:
 - o self attributes
 - o Conditional statements (if-else)
- Sample output showing student details and performance status

The screenshot shows the GitHub Copilot interface. On the left, the code for `Task1.py` is displayed:

```

1  # Task1.py > ...
2  """
3  University Performance System - Student Class
4  """
5
6  class Student:
7      """Student class to manage student information and performance tracking."""
8
9      # Class variable for class average
10     class_average = 75
11
12     def __init__(self, name, roll_number, marks):
13         """
14             Initialize student with name, roll_number, and marks.
15
16             Args:
17                 name (str): Student's name
18                 roll_number (int): Student's roll number
19                 marks (float): Student's marks
20
21             self.name = name
22             self.roll_number = roll_number
23             self.marks = marks
24
25     def display_student_details(self):
26         """Display student information - name, roll_number, and marks."""
27         # TODO: Print student details in a formatted way
28         print("Student Name: " + self.name)
29         print("Roll Number: " + str(self.roll_number))
30         print("Marks: " + str(self.marks))
31
32     def check_performance_status(self):
33         """Check if student's marks are above class average using if-else statement."""
34         # TODO: Compare self.marks with class_average and return performance status
35         if self.marks >= self.class_average:
36             status = "Above Average"
37         else:
38             status = "Below Average"
39         return status
40
41     def is_above_average(self):
42         """Return True if marks are above class average, False otherwise."""
43         if self.marks > self.class_average:
44             return True
45         else:
46             return False

```

On the right, the interface shows a guide for creating a Python Student class, a "Read Python file" section, and a "Run push command?" dialog box with "Allow" and "Skip" buttons.

OUTPUT:

The terminal window shows the execution of `Task1.py`:

```

Is Above Average: False

--- Student 3 Details ---
Student Name: Carol Davis
Roll Number: 103
Marks: 75
Performance Status: Above Average
Is Above Average: False

=====
Class Average: 75
=====
PS C:\Users\tloke\OneDrive\AI>

```

Task-02: 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.

Expected Outcome

- A complete for loop generated by Copilot

- Use of:

- Modulus operator to identify even numbers ◦

Conditional statements

- Correct and formatted output for valid inputs

The screenshot shows a Python code editor interface with the following details:

- Code Area:** The code is for a file named Task2.py. It starts with a multi-line comment and defines a function to analyze sensor readings. It uses a for loop to iterate over a list of sensor readings, checking if each is even using the modulus operator (%). If even, it calculates the square and prints the reading, number, and square. If odd, it prints the reading and number as skipped. It also includes a list comprehension alternative at the end.
- Completion Panel:** A large panel on the right provides guidance for Copilot to complete the loop logic. It includes:
 - A title: "PYTHON STUDENT CLASS FOR UNIVERSITY PERFORMANCE..."
 - A main text area: "Write the initial part of a for loop to iterate over a list of integer sensor readings, then add clear comments to guide Copilot to identify even numbers using the modulus operator (%), calculate their square, and print the result in a readable format using if conditions, allowing Copilot to complete the remaining loop logic and generate the full working code with correct formatted output."
 - A "Read Python file" button.
 - A "Created" message: "Perfect! I've created a comprehensive sensor reading analysis script with:"
 - A "Key Features:" section with bullet points:
 - For loop structure iterating over a list of sensor readings (12, 7, 24, 15, 8, 3, 36, 11, 20, 5)
 - Clear comments guiding Copilot to:
 - Use the modulus operator (%) to identify even numbers
 - Calculate squares using multiplication
 - Apply if-else conditions for even/odd classification
 - Formatted output with aligned columns showing:
 - Reading value
 - Classification (Even/Odd)
 - Square value (for even numbers only)
 - Complete working code that processes all sensor readings
 - Bonus section showing a list comprehension alternative
 - A note: "The code demonstrates proper formatting with readable output and includes comments at each step to guide code generation logic. The output will show which readings are even numbers and their respective squares."
 - A "Describe what to build next" input field.
 - Agent and Auto dropdown menus.
 - Build and Run buttons.

OUTPUT:

```

SENSOR READINGS ANALYSIS - EVEN NUMBERS AND THEIR SQUARES
=====
Original Sensor Readings: [12, 7, 24, 15, 8, 3, 36, 11, 20, 5]

Reading: 12 | Even Number | Square: 144
Reading: 7 | Odd Number | Skipped
Reading: 24 | Even Number | Square: 576
Reading: 15 | Odd Number | Skipped
Reading: 8 | Even Number | Square: 64
Reading: 3 | Odd Number | Skipped
Reading: 36 | Even Number | Square: 1296
Reading: 11 | Odd Number | Skipped
Reading: 20 | Even Number | Square: 400
Reading: 5 | Odd Number | Skipped

=====
Analysis Complete!
=====

--- Alternative: Extract even numbers using list comprehension ---
Even Numbers: [12, 24, 8, 36, 20]
Their Squares: [144, 576, 64, 1296, 400]
PS C:\Users\tloke\OneDrive\AI>

```

Task-03: 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 `BankAccount` with attributes:

- `account_holder`
- `balance`

Use GitHub Copilot to complete methods for:

- Depositing money
 - Withdrawing money
 - Preventing withdrawals when the balance is insufficient
- Guide Copilot using method names and short comments.

Expected Outcome

- A fully functional `BankAccount` class
- Copilot-generated methods using:

o if-else conditions o Class

attributes via self

- Proper handling of invalid withdrawal attempts with user-friendly Messages

Task3.py ...

```
1 """
2 Banking System - BankAccount Class
3 A simple yet functional banking simulation system
4 """
5
6 class BankAccount:
7     """Class to manage bank account operations including deposits, withdrawals, and balance tracking."""
8
9     def __init__(self, account_holder, initial_balance):
10         """
11             Initialize bank account with account holder name and initial balance.
12
13             Args:
14                 account_holder (str): Name of the account holder
15                 initial_balance (float): Initial balance in the account
16
17             self.account_holder = account_holder
18             self.balance = initial_balance
19
20     def deposit_money(self, amount):
21         """
22             Deposit money into the account.
23             Add the deposit amount to self.balance and display confirmation message.
24
25             # Check if the deposit amount is valid (positive number)
26             if amount > 0:
27                 self.balance += amount
28                 print(f"\n Deposit Successful!")
29                 print(f" Amount Deposited: ${amount:.2f}")
30                 print(f" New Balance: ${self.balance:.2f}")
31             else:
32                 print(f"\n Invalid Deposit Amount: Amount must be greater than zero.")
33
34     def withdraw_money(self, amount):
35         """
36             Withdraw money from the account.
37             Use if-else to check if sufficient balance exists before allowing withdrawal.
38
39             # Check if withdrawal amount is valid
40             if amount <= 0:
41                 print(f"\n Invalid Withdrawal Amount: Amount must be greater than zero.")
42             # Check if account has sufficient balance for the withdrawal
43             elif amount > self.balance:
44                 print(f"\n Insufficient Balance Error!")
45                 print(f" Requested Amount: ${amount:.2f}")
```

← PYTHON STUDENT CLASS FOR UNIVERSITY PERFORMANCE... □

Create a Python class called BankAccount with attributes account_holder and balance initialized in `__init__`. Then use method names and short comments to guide Copilot to generate methods for depositing money, withdrawing money, and preventing withdrawals when the balance is insufficient using self and if-else conditions, with user-friendly messages for invalid withdrawal attempts and a fully functional banking simulation.

✓ Created and read Task3.py file

Perfect! I've created a comprehensive BankAccount class with:

Key Features:

- Class Attributes:** `account_holder` and `balance` initialized in `__init__`.
- Three Core Methods:**
 - `deposit_money()` - Adds funds to the account with validation
 - `withdraw_money()` - Withdraws funds with multiple if-else conditions:
 - Checks for valid withdrawal amount
 - Validates sufficient balance before withdrawal
 - Prevents withdrawal if insufficient funds
 - `check_balance()` and `display_account_info()` - Helper methods
- User-Friendly Messages:**
 - Success indicators (✓) for valid operations
 - Error indicators (✗) for invalid operations
 - Clear error messages showing requested amount vs. available balance
 - Shortage amount display for insufficient balance attempts
- Fully Functional Banking Simulation** demonstrating:
 - Creating accounts with initial balances
 - Multiple deposits and withdrawals

Describe what to build next

Agent Auto

OUTPUT:

```
--- Initial Account Information ---
Account Holder: John Doe
Balance: $1000.00

--- Operation 1: Deposit $500 ---
✓ Deposit Successful!
  Amount Deposited: $500.00
  New Balance: $1500.00

--- Operation 2: Withdraw $200 ---
✓ Withdrawal Successful!
  Amount Withdrawn: $200.00
  New Balance: $1300.00

--- Operation 3: Attempt to withdraw $2000 (Insufficient Balance) ---
X Insufficient Balance Error!
  Requested Amount: $2000.00
  Current Balance: $1300.00
  Shortage: $700.00

--- Operation 4: Deposit $800 ---
✓ Deposit Successful!
  Amount Deposited: $800.00
  New Balance: $2100.00

--- Operation 5: Withdraw $1500 ---
✓ Withdrawal Successful!
  Amount Withdrawn: $1500.00
  New Balance: $600.00

--- Operation 6: Attempt to withdraw -$100 (Invalid Amount) ---
X Invalid Withdrawal Amount: Amount must be greater than zero.

--- Operation 7: Attempt to deposit -$300 (Invalid Amount) ---
X Invalid Deposit Amount: Amount must be greater than zero.

--- Final Account Information ---
Account Holder: John Doe
Balance: $600.00

=====
BANKING SIMULATION COMPLETE
=====

--- MULTI-ACCOUNT DEMONSTRATION ---
=====

Account 1 - Alice Smith
```

Task-04:

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 Use comments to guide Copilot's code completion.

Expected Outcome

- A complete while loop generated by Copilot
- Correct index handling and condition checks
- Cleanly formatted output listing eligible students

The screenshot shows the GitHub Copilot interface with a code editor and a sidebar. The code editor contains a Python script named Task4.py. The script defines a list of student dictionaries and uses a while loop to print names of students with scores above 75. The sidebar provides context about the task, lists key features like 'List of Dictionaries' and 'While Loop Implementation', and shows build logs and additional analysis.

```
Task4.py > ...
1 """
2 Student Score Analysis System
3 Use while loops to identify and display students who scored above 75
4 """
5
6 # Define a list of dictionaries containing student information
7 # Each dictionary has 'name' and 'score' keys
8 students = [
9     {"name": "Alice Johnson", "score": 78},
10    {"name": "Bob Smith", "score": 65},
11    {"name": "Carol Davis", "score": 82},
12    {"name": "David Wilson", "score": 72},
13    {"name": "Emma Brown", "score": 88},
14    {"name": "Frank Miller", "score": 91},
15    {"name": "Grace Lee", "score": 70},
16    {"name": "Henry Taylor", "score": 79},
17    {"name": "Ivy Martinez", "score": 55},
18    {"name": "Jack Anderson", "score": 85}
19 ]
20
21 print("*" * 70)
22 print("STUDENT SCORE ANALYSIS - ELIGIBILITY CHECK (Score > 75)")
23 print("*" * 70)
24
25 print(f"\nTotal Students: {len(students)}\n")
26
27 # Initialize counter for proper index handling in while loop
28 index = 0
29
30 # Use a while loop to iterate through the list of students
31 # Continue looping while index is within the range of the list
32 while index < len(students):
33     # Get the current student dictionary from the list
34     student = students[index]
35
36     # Extract the student's name and score from the dictionary
37     name = student["name"]
38     score = student["score"]
39
40     # Check if the student's score is greater than 75 (eligibility criteria)
41     if score > 75:
42         # Print the eligible student's name and score in a clean, readable format
43         print(f"✓ {name:20s} - Score: {score:3d} [ELIGIBLE]")
44     else:
45         # Print non-eligible students for comparison
```

OUTPUT:

```

PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/Task4
.py
=====
STUDENT SCORE ANALYSIS - ELIGIBILITY CHECK (Score > 75)
=====

Total Students: 10

✓ Alice Johnson      - Score: 78 [ELIGIBLE]
✗ Bob Smith          - Score: 65 [NOT ELIGIBLE]
✓ Carol Davis        - Score: 82 [ELIGIBLE]
✗ David Wilson       - Score: 72 [NOT ELIGIBLE]
✓ Emma Brown         - Score: 88 [ELIGIBLE]
✓ Frank Miller        - Score: 91 [ELIGIBLE]
✗ Grace Lee           - Score: 70 [NOT ELIGIBLE]
✓ Henry Taylor        - Score: 79 [ELIGIBLE]
✗ Ivy Martinez        - Score: 55 [NOT ELIGIBLE]
✓ Jack Anderson       - Score: 85 [ELIGIBLE]

=====
Eligible Students (Score > 75): 6 out of 10
Eligibility Percentage: 60.0%
=====

--- LIST OF ELIGIBLE STUDENTS ---
1. Alice Johnson
2. Carol Davis
3. Emma Brown
4. Frank Miller
5. Henry Taylor
6. Jack Anderson

=====
ANALYSIS COMPLETE
=====

PS C:\Users\tloke\OneDrive\AI>

```

Task-05: 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 `ShoppingCart` 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)

Use meaningful comments and method names to guide Copilot.

Expected Outcome

- A fully implemented `ShoppingCart` class
- Copilot-generated loops and conditional logic
- Correct handling of item addition, removal, and discount calculation

- Sample input/output demonstrating cart functionality

The screenshot shows a Jupyter Notebook interface with multiple tabs open. The active tab is 'Task5.py' which contains the following Python code:

```
1  """
2  E-Commerce Shopping Cart System
3  Complete shopping cart with item management, billing, and discount functionality
4  """
5
6 class ShoppingCart:
7     """Shopping cart class to manage items, calculate totals, and apply discounts."""
8
9     # Discount thresholds
10    DISCOUNT_THRESHOLD_1 = 100 # 5% discount for orders > $100
11    DISCOUNT_THRESHOLD_2 = 250 # 10% discount for orders > $250
12    DISCOUNT_THRESHOLD_3 = 500 # 15% discount for orders > $500
13
14    def __init__(self, customer_name=""):
15        """
16            Initialize shopping cart with customer name and empty items list.
17
18            Args:
19                customer_name (str): Name of the customer
20            """
21        self.customer_name = customer_name
22        self.items = [] # Empty list to store items as dictionaries
23
24    def add_item(self, name, price, quantity):
25        """
26            Add an item to the shopping cart.
27            Store item as a dictionary with name, price, and quantity keys.
28            Check if item already exists and update quantity if it does.
29        """
30        # Check if item already exists in the cart
31        item_found = False
32        for item in self.items:
33            if item["name"].lower() == name.lower():
34                # Item exists, update the quantity
35                item["quantity"] += quantity
36                item_found = True
37                print(f"✓ Updated '{name}' quantity to {item['quantity']}")
38                break
39
40        # If item doesn't exist, add it as a new item
41        if not item_found:
42            new_item = {
43                "name": name,
44                "price": price,
45                "quantity": quantity
46            }
47            self.items.append(new_item)
48
49    def calculate_subtotal(self):
50        subtotal = 0
51        for item in self.items:
52            subtotal += item["price"] * item["quantity"]
53        return subtotal
54
55    def apply_discount(self):
56        total_bill = self.calculate_subtotal()
57        if total_bill > self.DISCOUNT_THRESHOLD_1:
58            total_bill -= total_bill * 0.05
59        elif total_bill > self.DISCOUNT_THRESHOLD_2:
60            total_bill -= total_bill * 0.10
61        elif total_bill > self.DISCOUNT_THRESHOLD_3:
62            total_bill -= total_bill * 0.15
63        return total_bill
64
65    def calculate_total_bill(self):
66        subtotal = self.calculate_subtotal()
67        discount = self.apply_discount()
68        tax = subtotal * 0.08
69        final_total = subtotal - discount + tax
70        return final_total
71
72    def display_cart(self):
73        print("Cart Items:")
74        for item in self.items:
75            print(f"- {item['name']} x{item['quantity']} (${item['price']})")
76
77    def display_bill(self):
78        total_bill = self.calculate_total_bill()
79        print(f"Total Bill: ${total_bill:.2f}")
80
```

The notebook also includes a sidebar with 'Key Features:' and a 'Description' box at the bottom.

OUTPUT:

```
E-COMMERCE SHOPPING CART SYSTEM - DEMO
-----
--- ADDING ITEMS TO CART ---
✓ Added 'Laptop' ($899.99 x 1) to cart
✓ Added 'USB-C Cable' ($15.99 x 2) to cart
✓ Added 'Wireless Mouse' ($29.99 x 1) to cart
✓ Added 'Monitor Stand' ($45.00 x 1) to cart

-----
SHOPPING CART - Alice Smith
-----
Item Name          Price   Qty   Total
----- 
Laptop             $ 899.99  1 $ 899.99
USB-C Cable        $ 15.99  2 $ 31.98
Wireless Mouse     $ 29.99  1 $ 29.99
Monitor Stand      $ 45.00  1 $ 45.00
----- 

-----
--- ADDING MORE ITEMS ---
✓ Added 'Keyboard' ($79.99 x 1) to cart
✓ Updated 'USB-C Cable' quantity to 3

-----
SHOPPING CART - Alice Smith
-----
Item Name          Price   Qty   Total
----- 
Laptop             $ 899.99  1 $ 899.99
USB-C Cable        $ 15.99  3 $ 47.97
Wireless Mouse     $ 29.99  1 $ 29.99
Monitor Stand      $ 45.00  1 $ 45.00
Keyboard           $ 79.99  1 $ 79.99
----- 

-----
--- REMOVING ITEM ---
✓ Removed 'Monitor Stand' from cart

-----
SHOPPING CART - Alice Smith
-----
Item Name          Price   Qty   Total
----- 
Laptop             $ 899.99  1 $ 899.99
USB-C Cable        $ 15.99  3 $ 47.97
```

--- GENERATING BILL ---

=====

BILLING SUMMARY

=====

Subtotal:	\$ 1057.94
Discount (PREMIUM (15%))	-\$ 158.69
Subtotal After Discount:	\$ 899.25
Tax (8%):	\$ 71.94
TOTAL AMOUNT:	\$ 971.19

=====

=====

SECOND DEMO - HIGH-VALUE PURCHASE WITH PREMIUM DISCOUNT

=====

--- ADDING HIGH-VALUE ITEMS ---

- ✓ Added 'Desktop Computer' (\$1299.99 x 1) to cart
- ✓ Added 'Gaming Monitor' (\$399.99 x 2) to cart
- ✓ Added 'RGB Keyboard' (\$129.99 x 1) to cart
- ✓ Added 'Professional Headset' (\$199.99 x 1) to cart

=====

SHOPPING CART - Bob Johnson

=====

Item Name	Price	Qty	Total
Desktop Computer	\$ 1299.99	1	\$ 1299.99
Gaming Monitor	\$ 399.99	2	\$ 799.98
RGB Keyboard	\$ 129.99	1	\$ 129.99
Professional Headset	\$ 199.99	1	\$ 199.99

=====

BILLING SUMMARY

=====

Subtotal:	\$ 2429.95
Discount (PREMIUM (15%))	-\$ 364.49
Subtotal After Discount:	\$ 2065.46
Tax (8%):	\$ 165.24
TOTAL AMOUNT:	\$ 2230.69

=====

DEMO COMPLETE

=====