

# **Assignment 6.4**

## **AI Assisted Coding**

**Name: Sunny Kolluri**

**HTNO: 2303A52444**

**Batch 43**

**Task 1:**

**Prompt:**

Student Performance Evaluation System

class Student:

```
#def __init__(self, name, roll_number, marks):  
    self.name = name  
    self.roll_number = roll_number  
    self.marks = marks
```

# Create a method to display student details it should print student name, roll number and marks clearly. Create a method named check\_performance it should take class\_average as parameter.

#use if-else to check if student marks are above class average return a message about performance

#Create a sample student object and test the methods

```

236     # self.name = name
237     # self.roll_number = roll_number
238     # self.marks = marks
239     # create a method to display student details it should print student name, roll number and marks clearly
240     # create a method named check_performance it should take class_average as parameter
241     # use if-else to check if student marks are above class average return a message about performance
242     # create a sample student object and test the methods
243
244     class Student:
245         def __init__(self, name, roll_number, marks):
246             self.name = name
247             self.roll_number = roll_number
248             self.marks = marks
249
250         def display_details(self):
251             print(f"Name: {self.name}, Roll Number: {self.roll_number}, Marks: {self.marks}")
252
253         def check_performance(self, class_average):
254             if self.marks > class_average:
255                 return f"{self.name} performed above average."
256             else:
257                 return f"{self.name} performed at or below average."
258
259     # Create a sample student object and test the methods
260     student1 = Student("Alice", "12345", 85)
261     student1.display_details()
262     print(student1.check_performance(75))

```

## Observation – Student Performance Evaluation System

In this task, GitHub Copilot was used to complete the implementation of a Python class based on comment-based prompts. After providing the class skeleton and descriptive comments, Copilot successfully generated the required methods.

The AI assistant created:

- A `display_details()` method that prints student information using class attributes (`self.name`, `self.roll_number`, `self.marks`).
- A `check_performance()` method that uses an **if-else conditional statement** to compare student marks with the class average and return an appropriate message.

### Task 2:

**Prompt:**

# list of sensor readings

readings = [10, 15, 22, 9, 8, 30]

Write a for loop to iterate through the readings identify even numbers using modulus operator Calculate square of even numbers. Print the number and its square in a readable format

The screenshot shows a GitHub Copilot interface integrated into VS Code. The code editor displays a Python script with the following content:

```

262 print(student1.check_performance(75))"""
263 # list of sensor readings
264 readings = [10, 15, 22, 9, 8, 30]
265 # write a for loop to iterate through the readings
266 # identify even numbers using modulus operator
267 # calculate square of even numbers
268 # print the number and its square in a readable format
269 for reading in readings:
270     if reading % 2 == 0:
271         square = reading ** 2
272         print(f"Number: {reading}, Square: {square}")

```

The terminal window shows the output of the script:

```

s/navya/OneDrive/Desktop/AI Assist/Assignment 2
Number: 10, Square: 100
Number: 22, Square: 484
Number: 8, Square: 64
Number: 30, Square: 900
PS C:\Users\navya\OneDrive\Desktop\AI Assist>

```

The GitHub Copilot sidebar on the right suggests "Build with Agent" and provides options to "Generate Agent Instructions" and "onboard AI onto your codebase".

## Observation

Using the provided comment prompts, GitHub Copilot generated the remaining loop logic successfully. The AI created a proper **for loop**, used the **modulus operator (%)** to identify even numbers, and calculated the square of each valid reading. The output was printed in a clear and readable format, and the program executed without errors. This task shows how AI can assist in writing loops and conditional statements efficiently while still requiring user verification.

### Task 3:

Prompt:

# Banking Transaction Simulation

class Bank Account:

```

def __init__(self, account_holder, balance):
    self.account_holder = account_holder
    self.balance = balance

```

Create a method to deposit money into the account update the balance and print confirmation message. Create a method to withdraw money. Use if-else to check if balance is sufficient. Prevent withdrawal if balance is insufficient. Print user friendly messages. Create a sample account and test deposit and withdrawal

```

282     class BankAccount:
283         def __init__(self, account_holder, balance):
284             self.account_holder = account_holder
285             self.balance = balance
286
287         def deposit(self, amount):
288             self.balance += amount
289             print(f"Deposited ${amount}. New balance: ${self.balance}")
290
291         def withdraw(self, amount):
292             if self.balance >= amount:
293                 self.balance -= amount
294                 print(f"Withdrew ${amount}. New balance: ${self.balance}")
295             else:
296                 print("Insufficient funds.")
297
298     # Create a sample account and test deposit and withdrawal
299     account = BankAccount("John", 100)
300     account.deposit(50)
301     account.withdraw(30)
302     account.withdraw(150)

```

TERMINAL

```

Number: 10, Square: 100
Number: 22, Square: 484
Number: 8, Square: 64
Number: 30, Square: 900
PS C:\Users\navya\OneDrive\Desktop\AI Assit> & "C:/Users/navya/OneDrive/Desktop/AI Assit/.venv/Scripts/python.exe" "c:/Users/navya/OneDrive/Desktop/AI Assit/Assignment 2"
Deposited $50. New balance: $150
Withdrew $30. New balance: $120
Insufficient funds.
PS C:\Users\navya\OneDrive\Desktop\AI Assit>

```

PROBLEMS OUTPUT TERMINAL GITLENS PORTS

RECENT SESSIONS

- Classifying chatbot query types Completed Local • 1 wk
- Output clarification request Completed Local • 3 wks
- Code to check if a number is prime Completed Local • 4 wks

Build with Agent

AI responses may be inaccurate. Generate Agent instructions to onboard AI onto your codebase.

## Observation

GitHub Copilot successfully generated the deposit and withdraw methods based on the comment prompts. The AI used class attributes through `self`, applied **if-else conditional logic** to prevent withdrawals when the balance was insufficient, and produced clear user-friendly messages. The program executed correctly and demonstrated how AI can help build functional class methods while still requiring human review.

### Task 4:

#### Prompt:

# list of students with name and score

```

students = [{"name": "Asha", "score": 80},
            {"name": "Ravi", "score": 65},
            {"name": "Kiran", "score": 90},
            {"name": "Meena", "score": 70}]

```

#Write a while loop to iterate through the student list

#Check students who scored more than 75

#Print names of eligible students for scholarship

The screenshot shows a code editor with a Python script. The script defines a list of students and uses a while loop to iterate through them, printing names of students who scored above 75. The GitHub Copilot interface is visible on the right, showing a message about AI responses being inaccurate and a button to "Generate Agent Instructions".

```
303     # list of students with name and score
304     students = [{"name": "Asha", "score": 80},
305                  {"name": "Ravi", "score": 65},
306                  {"name": "Kiran", "score": 90},
307                  {"name": "Meena", "score": 70}]
308
309     #Write a while loop to iterate through the student list
310     #Check students who scored more than 75
311     #Print names of eligible students for scholarship
312     i = 0
313     while i < len(students):
314         student = students[i]
315         if student["score"] > 75:
316             print(student["name"])
317             i += 1
```

## Observation– Scholarship Eligibility (While Loop)

GitHub Copilot generated the while loop based on the given prompts. The loop correctly iterated through the list using an index, applied a conditional check to identify students scoring above 75, and printed the eligible names. The program executed successfully and demonstrated how AI can assist in generating loop and condition-based logic

Task 5:

Prompt:

# Online Shopping Cart Module

class ShoppingCart:

def \_\_init\_\_(self):

#Create an empty list to store items each item should contain name, price and quantity

self.items= []

#Create a method to add items to the cart create a method to remove items from the cart

#Create a method to calculate total bill using loop apply discount using if condition when total exceeds a limit

#Create sample cart and test all methods

The screenshot shows the Visual Studio Code interface with the AI Assistant extension active. The Explorer sidebar on the left lists files like '.venv', 'Assignment 3.py', 'assignment 3', 'assignment 1', 'Assignment 2' (which is currently selected), 'Assignment 5', 'assignment1.py', and 'week 4'. The main code editor area displays Python code for an 'Assignment 2' module, specifically a 'ShoppingCart' class with methods for adding items, removing items, and calculating total bill with a 20% discount if the total exceeds \$1000. The terminal at the bottom shows the code running successfully with output: 'Total bill: \$840.0'. The status bar indicates the file is Python, has 3.13.7 (venv) version, and is in Go Live mode.

```
318 # Online Shopping Cart Module
319
320
321 # Create a method to store items each item should contain name, price and quantity
322 self.items = []
323
324 #Create a method to add items to the cart create a method to remove items from the cart
325
326 #Create a method to calculate total bill using loop apply discount using if condition when total exceeds a limit
327
328 class ShoppingCart:
329     def __init__(self):
330         self.items = []
331
332     def add_item(self, name, price, quantity):
333         self.items.append({"name": name, "price": price, "quantity": quantity})
334
335     def remove_item(self, name):
336         self.items = [item for item in self.items if item["name"] != name]
337
338     def calculate_total(self):
339         total = sum(item["price"] * item["quantity"] for item in self.items)
340         if total > 1000:
341             total *= 0.8 # Apply 20% discount
342
343     # Create sample cart and test all methods
344     cart = ShoppingCart()
345     cart.add_item("Laptop", 1000, 1)
346     cart.add_item("Mouse", 25, 2)
347     print(f"Total bill: ${cart.calculate_total()}")
```

## Observation – Online Shopping Cart Module

Using the comment prompts, GitHub Copilot completed the Shopping Cart class with methods to add items, remove items, and calculate the total bill. The AI correctly used loops and conditional statements to compute the total and apply a discount when the amount exceeded a limit. The program ran successfully and showed how AI can help build functional class-based applications efficiently.