Lab-6.3

T.Rakshitha

2303a51172

Batch:18

Task Description #1: Classes (Student Class)

Scenario

You are developing a simple student information management module.

Task

- Use an AI tool (GitHub Copilot / Cursor AI / Gemini) to complete a Student class.
- The class should include attributes such as name, roll number, and branch.
- Add a method display_details() to print student information.
- Execute the code and verify the output.
- Analyze the code generated by the AI tool for correctness and clarity.

Expected Output #1

- A Python class with a constructor (__init__) and a display_details() method.
- Sample object creation and output displayed on the console.
- Brief analysis of AI-generated code.

Prompts:

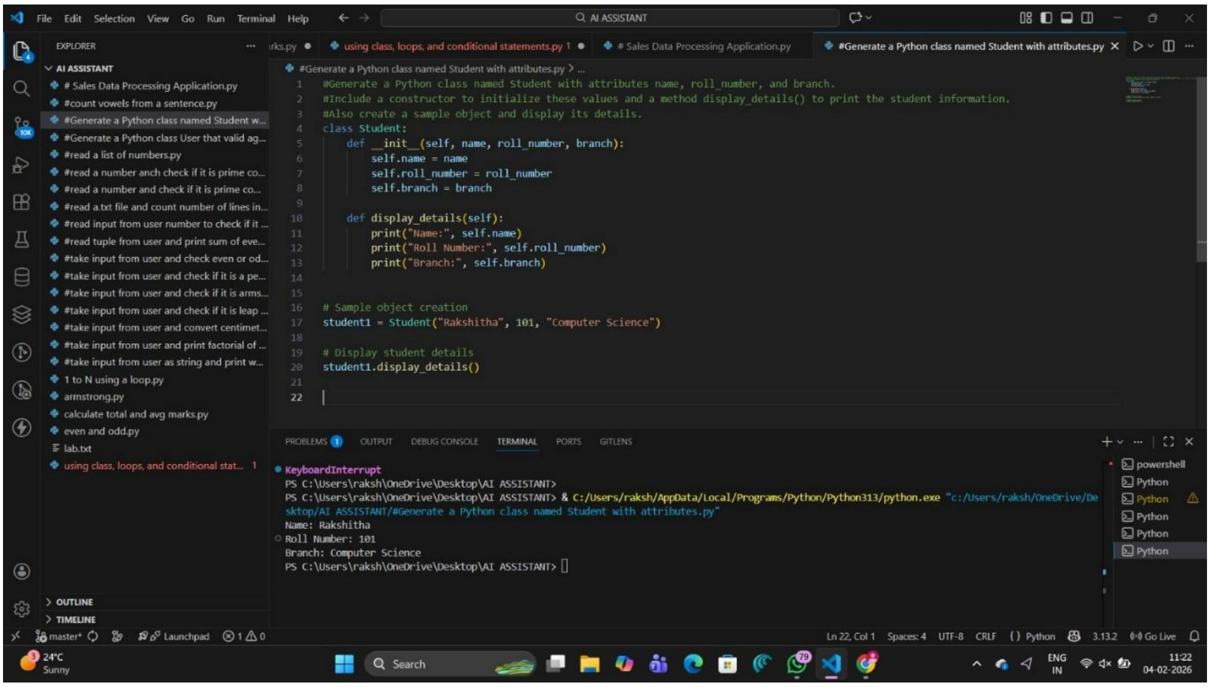
#Generate a Python class named Student with attributes name, roll_number, and branch.

#Include a constructor to initialize these values and a method display_details() to print the student information.

#Also create a sample object and display its details.

# Sample object creation

# Display student details



```python
#Generate a Python class named Student with attributes name, roll_number, and branch.
#Include a constructor to initialize these values and a method display_details() to print the student information.
#Also create a sample object and display its details.
class Student:
    def __init__(self, name, roll_number, branch):
        self.name = name
        self.roll_number = roll_number
        self.branch = branch

    def display_details(self):
        print("Name:", self.name)
        print("Roll Number:", self.roll_number)
        print("Branch:", self.branch)


# Sample object creation
student1 = Student("Rakshitha", 101, "Computer Science")

# Display student details
student1.display_details()
```

```
KeyboardInterrupt
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT>
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT> & C:/Users/raksh/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/raksh/OneDrive/Desktop/AI ASSISTANT/#Generate a Python class named Student with attributes.py"
Name: Rakshitha
Roll Number: 101
Branch: Computer Science
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT>
```

Analysis:

display_details() method correctly prints student information.

Code is simple, readable, and well-structured.

Task Description #2: Loops (Multiples of a Number)

Scenario

You are writing a utility function to display multiples of a given number.

Task

• Prompt the AI tool to generate a function that prints the first 10 multiples of a given number

using a loop.

• Analyze the generated loop logic.

• Ask the AI to generate the same functionality using another controlled looping structure (e.g.,

while instead of for).

Expected Output #2

• Correct loop-based Python implementation.

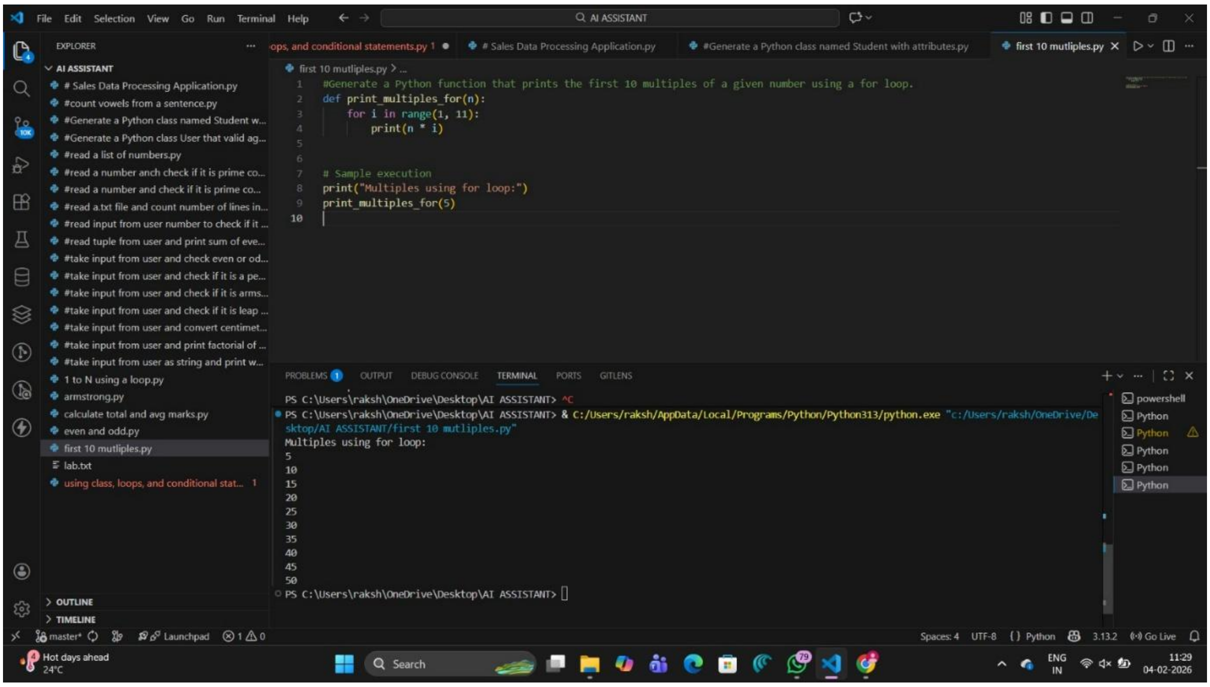
• Output showing the first 10 multiples of a number.

• Comparison and analysis of different looping approaches.

Prompts:

#Generate a Python function that prints the first 10 multiples of a given number using a for loop.

# Sample execution



```python
#Generate a Python function that prints the first 10 multiples of a given number using a for loop.
def print_multiples_for(n):
    for i in range(1, 11):
        print(n * i)


# Sample execution
print("Multiples using for loop:")
print_multiples_for(5)
```

```
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT> ^C
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT> & C:/Users/raksh/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/raksh/OneDrive/Desktop/AI ASSISTANT/first 10 mutliples.py"
Multiples using for loop:
5
10
15
20
25
30
35
40
45
50
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT> 
```

Analysis:

Uses for and while loops to generate multiples.

for loop is concise and suitable for fixed iterations.

while loop offers more control but requires manual counter handling.

Task Description #3: Conditional Statements (Age Classification)

Scenario

You are building a basic classification system based on age.

Task

- **Ask the AI tool to generate nested if-elif-else conditional statements to classify age groups**

(e.g., child, teenager, adult, senior).

- **Analyze the generated conditions and logic.**
- **Ask the AI to generate the same classification using alternative conditional structures (e.g.,**

simplified conditions or dictionary-based logic).

Expected Output #3

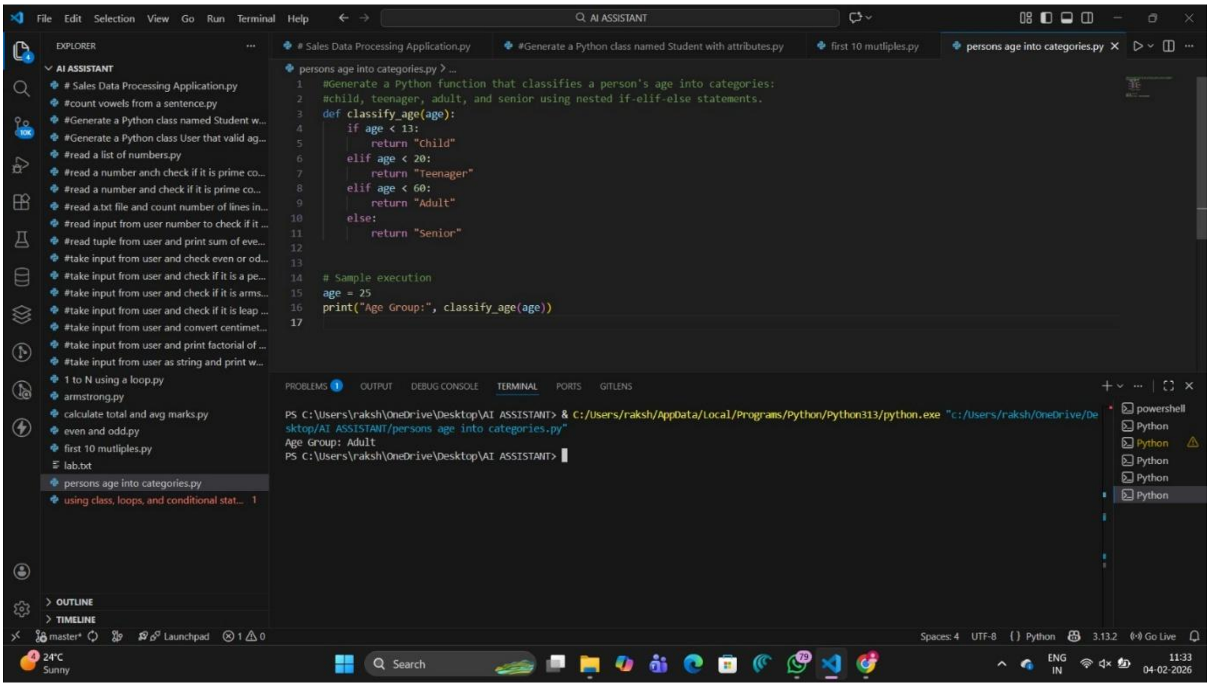
- **A Python function that classifies age into appropriate groups.**
- **Clear and correct conditional logic.**
- **Explanation of how the conditions work.**

Prompts:

#Generate a Python function that classifies a person's age into categories:

#child, teenager, adult, and senior using nested if-elif-else statements.

# Sample execution



```python
#Generate a Python function that classifies a person's age into categories:
#child, teenager, adult, and senior using nested if-elif-else statements.
def classify_age(age):
    if age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 60:
        return "Adult"
    else:
        return "Senior"


# Sample execution
age = 25
print("Age Group:", classify_age(age))
```

```
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT> & C:/Users/raksh/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/raksh/OneDrive/Desktop/AI ASSISTANT/persons age into categories.py"
Age Group: Adult
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT>
```

Analysis:

Uses if-elif-else to implement decision-making logic.

Conditions are ordered correctly to avoid overlap.

Task Description #4: For and While Loops (Sum of First n Numbers)

Scenario

You need to calculate the sum of the first n natural numbers.

Task

- **Use AI assistance to generate a sum_to_n() function using a for loop.**

- Analyze the generated code.

- Ask the AI to suggest an alternative implementation using a while loop or a mathematical

formula.

Expected Output #4

- Python function to compute the sum of first n numbers.

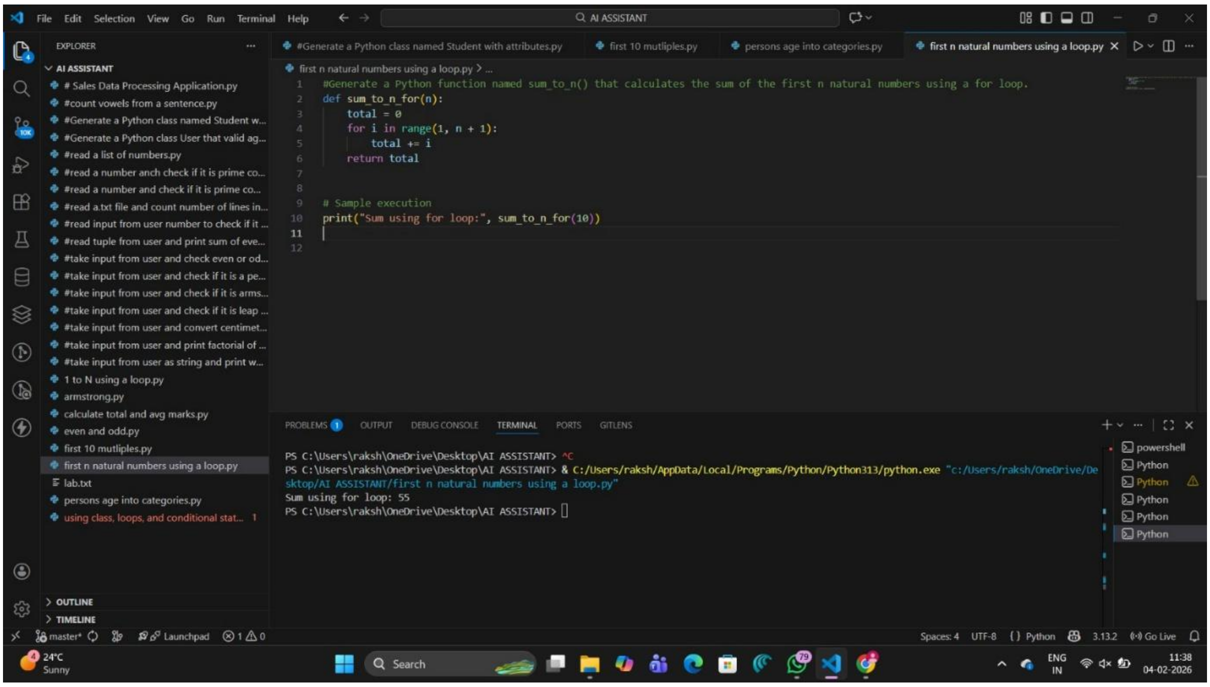
- Correct output for sample inputs.

- Explanation and comparison of different approaches.

Prompts:

#Generate a Python function named sum_to_n() that calculates the sum of the first n natural numbers using a for loop.

Sample execution



```python
#Generate a Python function named sum_to_n() that calculates the sum of the first n natural numbers using a for loop.
def sum_to_n_for(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total


# Sample execution
print("Sum using for loop:", sum_to_n_for(10))
```

```
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT> ^C
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT> & C:/Users/raksh/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/raksh/OneDrive/Desktop/AI ASSISTANT/first n natural numbers using a loop.py"
Sum using for loop: 55
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT>
```

Analysis:

Demonstrates iterative and mathematical problem-solving approaches.

Task Description #5: Classes (Bank Account Class)

Scenario

You are designing a basic banking application.

Task

• Use AI tools to generate a Bank Account class with methods such as deposit(), withdraw(),

and check_balance().

• Analyze the AI-generated class structure and logic.

• Add meaningful comments and explain the working of the code.

Expected Output #5

• Complete Python Bank Account class.

• Demonstration of deposit and withdrawal operations with updated balance.

• Well-commented code with a clear explanation.

Prompts:

#Generate a Python class named BankAccount with attributes for account_holderand balance. Include methods deposit(), withdraw(), and check_balance().

#Demonstrate the class with sample deposit and withdrawal operations and add meaningful comments.

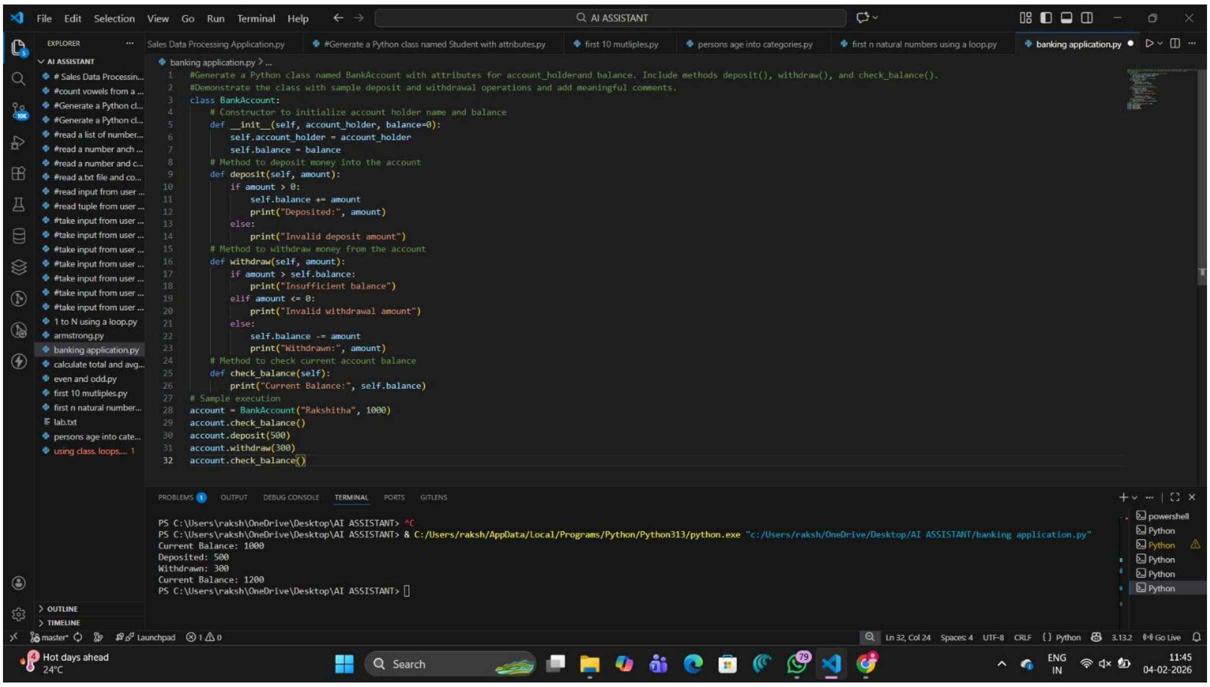
# Constructor to initialize account holder name and balance

# Method to deposit money into the account

# Method to withdraw money from the account

# Method to check current account balance

# Sample execution



```python
#Generate a Python class named BankAccount with attributes for account_holderand balance. Include methods deposit(), withdraw(), and check_balance().
#Demonstrate the class with sample deposit and withdrawal operations and add meaningful comments.
class BankAccount:
    # Constructor to initialize account holder name and balance
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance
    # Method to deposit money into the account
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print("Deposited:", amount)
        else:
            print("Invalid deposit amount")
    # Method to withdraw money from the account
    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient balance")
        elif amount <= 0:
            print("Invalid withdrawal amount")
        else:
            self.balance -= amount
            print("Withdrawn:", amount)
    # Method to check current account balance
    def check_balance(self):
        print("Current Balance:", self.balance)
# Sample execution
account = BankAccount("Rakshitha", 1000)
account.check_balance()
account.deposit(500)
account.withdraw(300)
account.check_balance()
```

```
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT> ^C
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT> & C:/Users/raksh/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/raksh/OneDrive/Desktop/AI ASSISTANT/banking application.py"
Current Balance: 1000
Deposited: 500
Withdrawn: 300
Current Balance: 1200
PS C:\Users\raksh\OneDrive\Desktop\AI ASSISTANT> 
```

Analysis:

Code structure is clear and easy to extend.