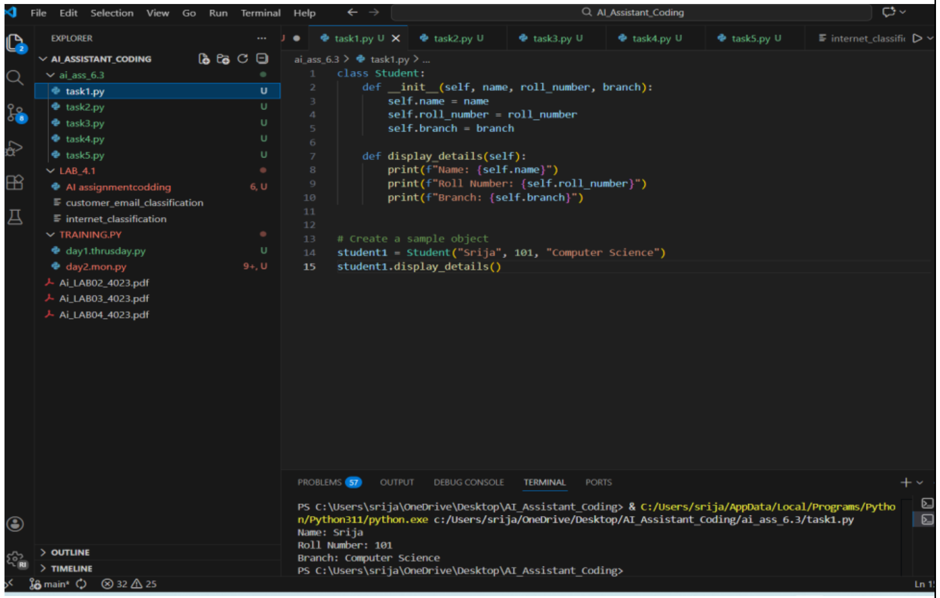
Program Name: B. Tech	**Assignment Type: Lab**		**Academic Year:**2025-2026
Course Coordinator Name	Dr. Rishabh Mittal		
Course Code	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week3 – Wednesday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Name	K.Srija	**Batches No**	2303A54023

AssignmentNumber:6.3(Present assignment number)/24(Total number of assignments)

Q.No.	**Question**	***Expected Time to complete***
1	**Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals** **Lab Objectives** • To explore AI-powered auto-completion features for core Python constructs such as classes, loops, and conditional statements. • To analyze how AI tools suggest logic for object-oriented programming and control structures. • To evaluate the correctness, readability, and completeness of AI-generated Python code. **Lab Outcomes (LOs)** After completing this lab, students will be able to: • Use AI tools to generate and complete Python class definitions and methods. • Understand and assess AI-suggested loop constructs for iterative tasks. • Generate and evaluate conditional statements using AI-driven prompts. • Critically analyze AI-assisted code for correctness, clarity, and efficiency. **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. **Prompt:** Generate a Python Student class with attributes name, roll number, and branch. Include a constructor and a method to display student details. Create a sample object and print the output.	Week3 - Wednesday



```python
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(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Branch: {self.branch}")


# Create a sample object
student1 = Student("Srija", 101, "Computer Science")
student1.display_details()
```

```
PS C:\Users\srija\OneDrive\Desktop\AI_Assistant_Coding> & C:/Users/srija/AppData/Local/Programs/Pytho
n/Python311/python.exe c:/Users/srija/OneDrive/Desktop/AI_Assistant_Coding/ai_ass_6.3/task1.py
Name: Srija
Roll Number: 101
Branch: Computer Science
PS C:\Users\srija\OneDrive\Desktop\AI_Assistant_Coding>
```

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.

Explanation:

> The prompt asks the AI to create a class, which is a blueprint for student objects.
> It clearly specifies attributes so the AI knows what data to store.
> Asking for a constructor (__init__) ensures values are initialized when an object is created.
> The display_details() method requirement tells the AI to add functionality to print information.
> Requesting sample object creation ensures practical demonstration and testing

Purpose: To practice Object-Oriented Programming (OOP) basics like classes, objects, and methods.

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

Scenario

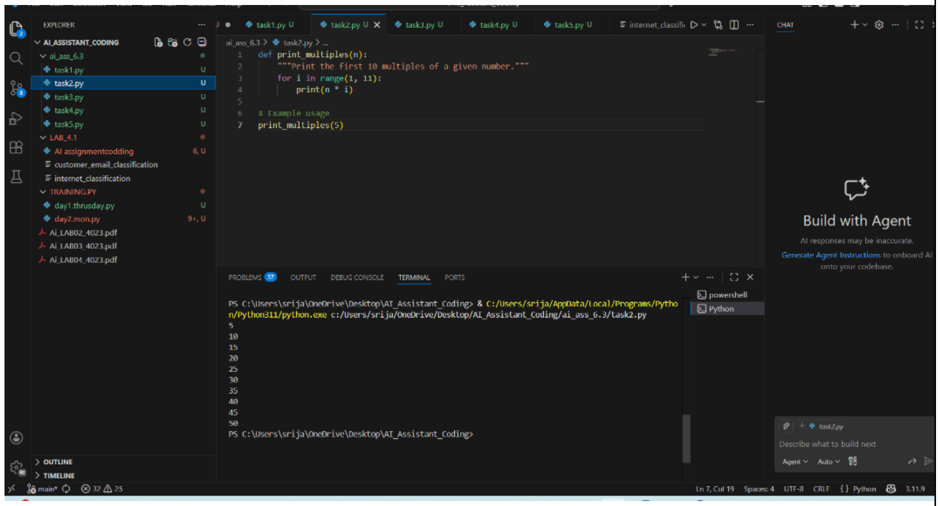
You are writing a utility function to display multiples of a given number.

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).

Prompt:

Write a Python function to print the first 10 multiples of a given number using a for loop.



```python
def print_multiples(n):
    """Print the first 10 multiples of a given number."""
    for i in range(1, 11):
        print(n * i)

# Example usage
print_multiples(5)
```

```
PS C:\Users\srija\OneDrive\Desktop\AI_Assistant_Coding> & C:/Users/srija/AppData/Local/Programs/Pytho
n/Python311/python.exe c:/Users/srija/OneDrive/Desktop/AI_Assistant_Coding/ai_ass_6.3/task2.py
5
10
15
20
25
30
35
40
45
50
PS C:\Users\srija\OneDrive\Desktop\AI_Assistant_Coding>
```

Expected Output #2
• Correct loop-based Python implementation.
• Output showing the first 10 multiples of a number.
• Comparison and analysis of different looping approaches.

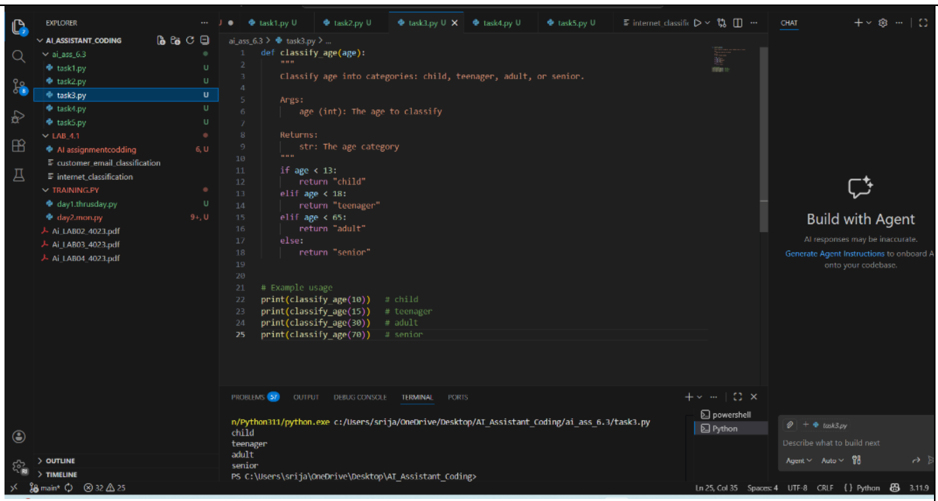
Explanation:
> The prompt defines a clear goal (print multiples).
> Specifying for loop forces AI to use a count-controlled loop.
> Asking for while loop version encourages understanding of alternative loop logic.
> Helps compare different looping mechanisms for the same problem.
Purpose: To understand loop structures and how iteration works differently in for vs while loops.

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).

Prompt:
Generate a Python function using if-elif-else statements to classify age as child, teenager, adult, or senior.



```python
def classify_age(age):
    """
    Classify age into categories: child, teenager, adult, or senior.

    Args:
        age (int): The age to classify

    Returns:
        str: The age category
    """
    if age < 13:
        return "child"
    elif age < 18:
        return "teenager"
    elif age < 65:
        return "adult"
    else:
        return "senior"


# Example usage
print(classify_age(10))   # child
print(classify_age(15))   # teenager
print(classify_age(30))   # adult
print(classify_age(70))   # senior
```

```
n/Python311/python.exe c:/Users/srija/OneDrive/Desktop/AI_Assistant_Coding/ai_ass_6.3/task3.py
child
teenager
adult
senior
PS C:\Users\srija\OneDrive\Desktop\AI_Assistant_Coding>
```

Expected Output #3
• A Python function that classifies age into appropriate groups.
• Clear and correct conditional logic.
• Explanation of how the conditions work.

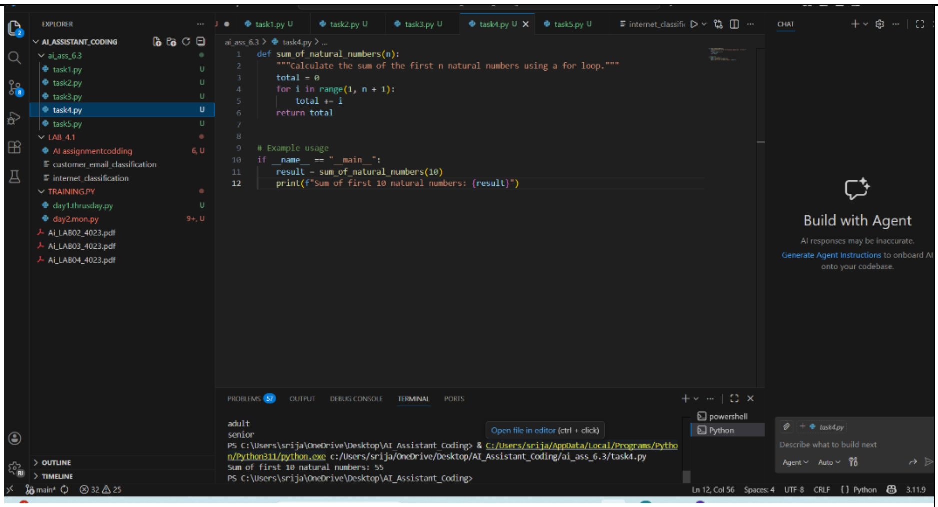
Explanation:
>The prompt asks AI to build decision-making logic using conditions.
>Age ranges ensure structured comparisons.
>Requesting nested if-elif-else teaches ordered condition checking.
>Asking for an alternative structure promotes optimization and multiple coding styles.
Purpose: To learn conditional logic and how to design classification systems.

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.

Prompt:
Write a Python function to calculate the sum of the first n natural numbers using a for loop.



```python
def sum_of_natural_numbers(n):
    """Calculate the sum of the first n natural numbers using a for loop."""
    total = 0
    for i in range(1, n + 1):
        total += i
    return total


# Example usage
if __name__ == "__main__":
    result = sum_of_natural_numbers(10)
    print(f"Sum of first 10 natural numbers: {result}")
```

```
adult
senior
PS C:\Users\srija\OneDrive\Desktop\AI_Assistant_Coding> & C:/Users/srija/AppData/Local/Programs/Pytho
n/Python311/python.exe c:/Users/srija/OneDrive/Desktop/AI_Assistant_Coding/ai_ass_6.3/task4.py
Sum of first 10 natural numbers: 55
PS C:\Users\srija\OneDrive\Desktop\AI_Assistant_Coding>
```

Expected Output #4
• Python function to compute the sum of first n numbers.
• Correct output for sample inputs.
• Explanation and comparison of different approaches.

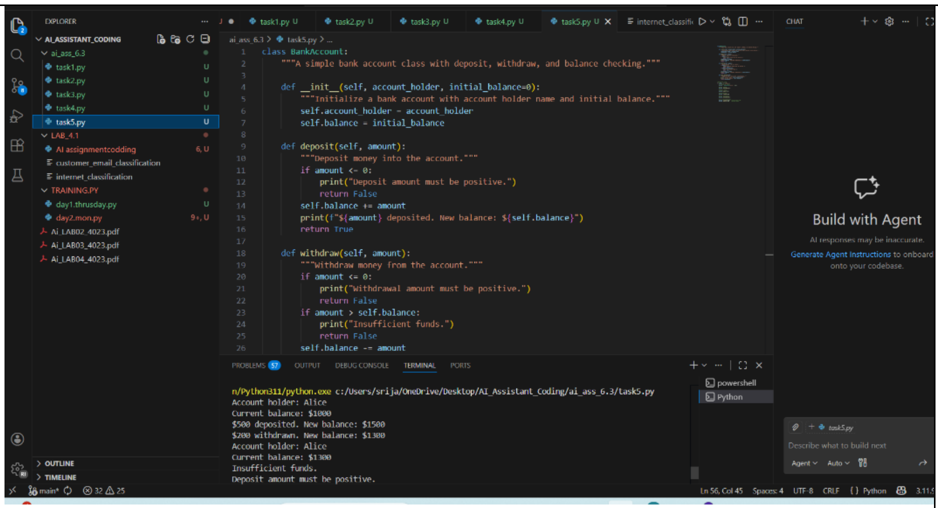
Explanation:
>The prompt defines a mathematical task (sum of numbers).
>For loop requirement ensures use of iteration.
>Alternative methods encourage exploring different problem-solving approaches.
>Mathematical formula highlights efficiency compared to loops.
Purpose: To understand loops, iteration, and algorithm optimization.

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.

Prompt:
Generate a Python BankAccount class with deposit, withdraw, and check_balance methods. Add comments and demonstrate usage.



```python
class BankAccount:
    """A simple bank account class with deposit, withdraw, and balance checking."""

    def __init__(self, account_holder, initial_balance=0):
        """Initialize a bank account with account holder name and initial balance."""
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit(self, amount):
        """Deposit money into the account."""
        if amount <= 0:
            print("Deposit amount must be positive.")
            return False
        self.balance += amount
        print(f"${amount} deposited. New balance: ${self.balance}")
        return True

    def withdraw(self, amount):
        """Withdraw money from the account."""
        if amount <= 0:
            print("Withdrawal amount must be positive.")
            return False
        if amount > self.balance:
            print("Insufficient funds.")
            return False
        self.balance -= amount
```

```
n/Python311/python.exe c:/Users/srija/OneDrive/Desktop/AI_Assistant_Coding/ai_ass_6.3/task5.py
Account holder: Alice
Current balance: $1000
$500 deposited. New balance: $1500
$200 withdrawn. New balance: $1300
Account holder: Alice
Current balance: $1300
Insufficient funds.
Deposit amount must be positive.
```

Expected Output #5

• Complete Python Bank Account class.
• Demonstration of deposit and withdrawal operations with updated balance.
• Well-commented code with a clear explanation.

Explanation:

>The prompt defines a real-world scenario (banking system).
>Specifying methods helps AI structure behavior logically.
>deposit() adds money, withdraw() subtracts with condition checking.
>check_balance() displays current state.
Purpose: To apply OOP concepts like encapsulation, methods, and real-world m