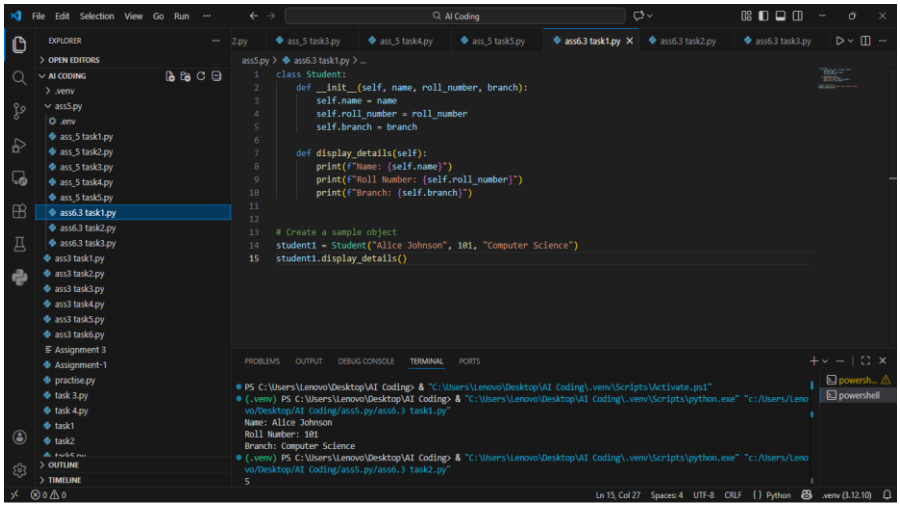
SCHOOLOFCOMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENTOFCOMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name		Dr. Rishabh Mittal	
CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week3 – Wednesday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Name	B.SUSHMA	**Batch**	Batch-48
AssignmentNumber:6.3(Presentassignmentnumber)/**24**(Totalnumberofassignments)			

Q.No.	Question	*ExpectedTime 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`. Use a constructor (`__init__`) and a `display_details()` method. Create a sample object, display the output on the console, and give a **brief**	Week3 - Wednesday

analysis of the code's correctness and clarity.

Code :



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 :

A class is a blueprint used to create objects in object-oriented programming. The Student class represents student-related data such as name, roll number, and branch. The constructor is used to initialize these data members at the time of object creation. Member functions defined inside the class operate on the data of the object. This approach improves data organization, reusability, and readability of the program.

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

Scenario

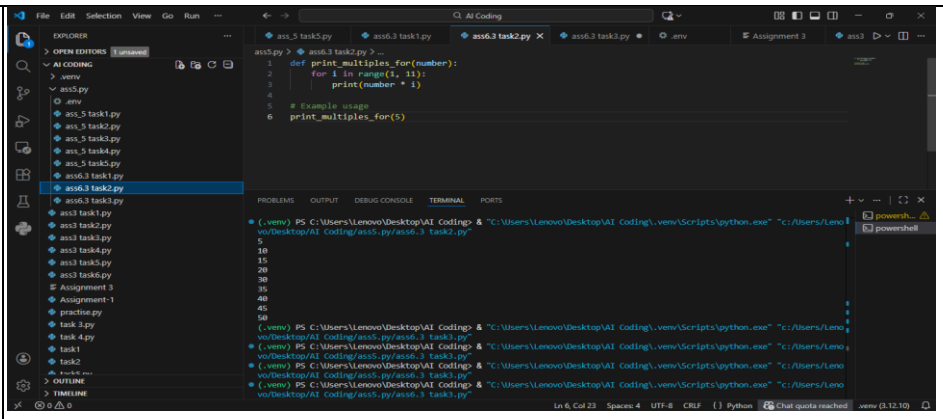
You are writing a utility function to display multiples of a given number.

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 : Generate a Python function that prints the first 10 multiples of a given number using a `for` loop.
Analyze the loop logic, then generate the same functionality using a `while` loop and compare both approaches.

Code :



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: Loops are control structures that allow a block of code to be executed repeatedly. To display multiples of a number, a loop performs repeated multiplication for a fixed number of times. A `for` loop is suitable when the number of iterations is known in advance. A `while` loop executes based on a condition and is useful when the termination depends on logic. Both looping structures achieve the same result but differ in control and usage.

Task Description #3: Conditional Statements (Age Classification)

Scenario

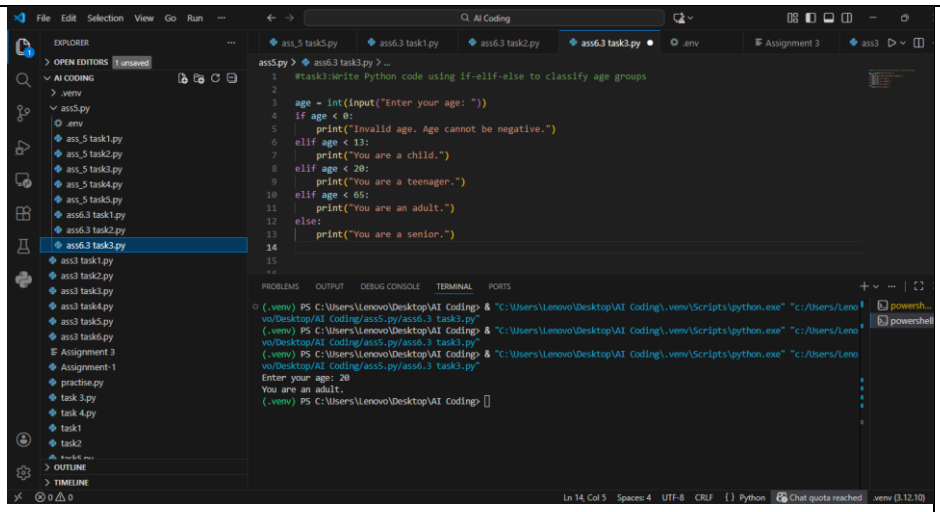
You are building a basic classification system based on age.

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 that classifies age into child, teenager, adult, and senior using nested `if-elif-else`.

Code :



Expected Output #3

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

Explanation : Conditional statements are used to make decisions based on given conditions. The `if-elif-else` structure allows checking multiple conditions in sequence. Age classification divides individuals into categories such as child, teenager, adult, and senior based on age ranges. Only one condition is executed at a time, ensuring correct classification. This structure helps in implementing decision-making logic clearly and efficiently.

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

Scenario

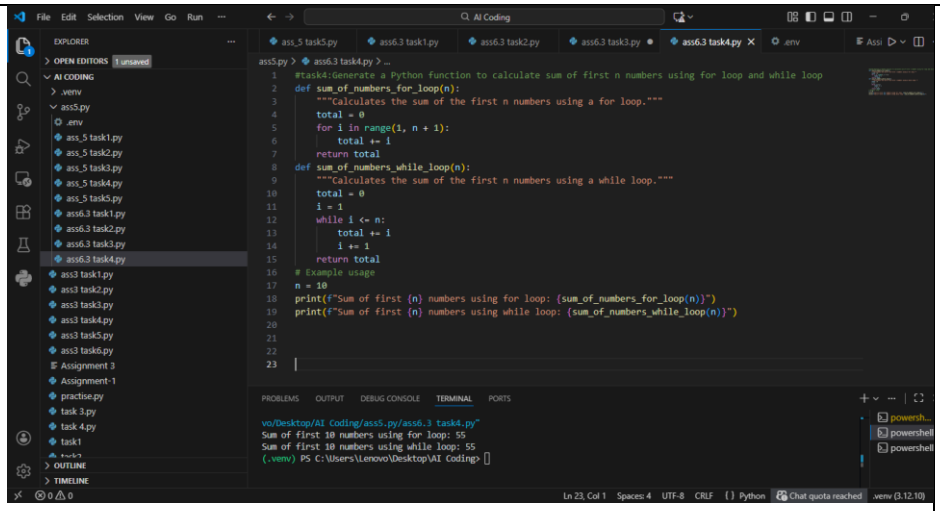
You need to calculate the sum of the first n natural numbers.

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 : Generate a Python function to calculate sum of first n numbers using for loop and while loop

Code :



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 sum of the first `n` natural numbers can be computed using iterative loops. A `for` loop is commonly used when the range of values is known. A `while` loop continues execution until a specified condition becomes false. An alternative approach uses a mathematical formula to compute the sum directly. Different methods provide flexibility based on efficiency and program requirements.

Task Description #5: Classes (BankAccount Class)

Scenario

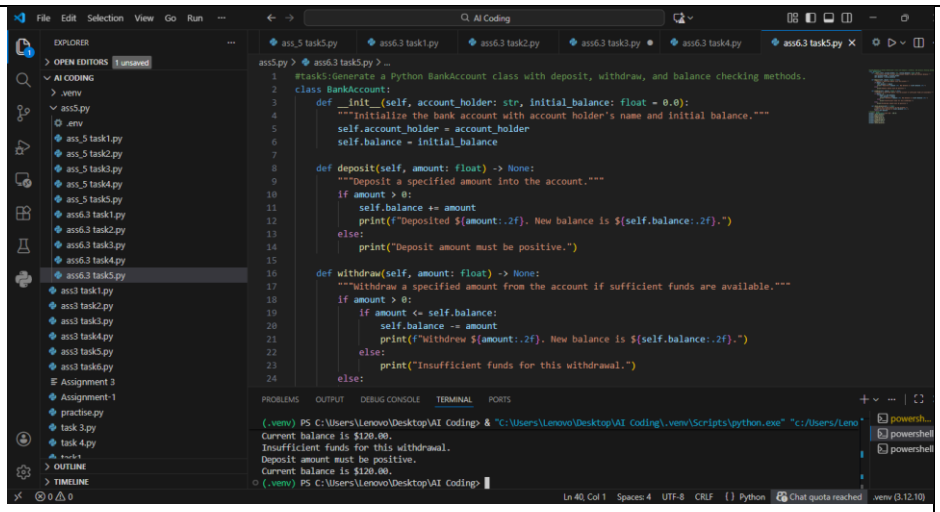
You are designing a basic banking application.

Task

- Use AI tools to generate a BankAccount 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 balance checking methods.

Code :



Expected Output #5

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

Explanation : The BankAccount class models real-world banking operations using object-oriented concepts. It stores account-related data such as balance and provides methods to perform operations like deposit and withdrawal. Encapsulation ensures that data is accessed and modified only through defined methods. This design improves security, modularity, and maintainability of the program. Such class-based designs are commonly used in real-world software applications.