

AI-ASSISTANT CODING

ASSIGNMENT-6.3

B.HARSHAVARDHAN

2303A52T02

Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals

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.

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.

AI Prompt Used

“Generate a Python class named Student with attributes name, roll_number, and branch. Include a constructor to initialize these attributes and a method display_details() to print the student information. Also show example object creation and method call.”

The screenshot shows a VS Code editor window with a file named 'ASS-6.3.py'. The code defines a class 'Student' with an '.__init__' method and a 'display_details' method. The '.__init__' method initializes 'name', 'roll_number', and 'branch' attributes. The 'display_details' method prints these attributes. Below the class definition, there is a task execution block that prints a separator, the task name, and then creates an instance of the 'Student' class with the name 'Harsha Vardhan', roll number '101', and branch 'Computer Science'. The terminal at the bottom shows the output of the execution, which matches the code's output.

```
1
2 class Student:
3     """A class to represent a student and manage their information."""
4
5     def __init__(self, name, roll_number, branch):
6         """Initialize student attributes."""
7         self.name = name
8         self.roll_number = roll_number
9         self.branch = branch
10
11     def display_details(self):
12         """Display student information."""
13         print(f"Name: {self.name}")
14         print(f"Roll Number: {self.roll_number}")
15         print(f"Branch: {self.branch}")
16
17
18 # Task 1 Execution
19 print("-" * 50)
20 print("TASK 1: STUDENT CLASS")
21 print("-" * 50)
22 student1 = Student("Harsha Vardhan", 101, "Computer Science")
23 student1.display_details()
24
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\HARSHVARDHAN\OneDrive\Desktop\AI-CODING> & C:\Users\HARSHVARDHAN\AppData\Local\Programs\Python\Python315\python.exe c:/Users/HARSHVARDHAN/OneDrive/Desktop/AI-CODING/ASS-6.3.py

TASK 1: STUDENT CLASS

Name: Harsha Vardhan

Roll Number: 101

Branch: Computer Science

PS C:\Users\HARSHVARDHAN\OneDrive\Desktop\AI-CODING>

Justification

This prompt was designed to:

- Clearly specify the **class name and required attributes**
- Request a **constructor (.__init__)**, ensuring object-oriented principles are followed
- Ask for a **method implementation** and **sample execution**
- Allow the AI to generate a **complete, testable class** rather than partial code

The prompt ensures that the AI output aligns with the lab requirement of demonstrating **basic OOP concepts** in Python.

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).

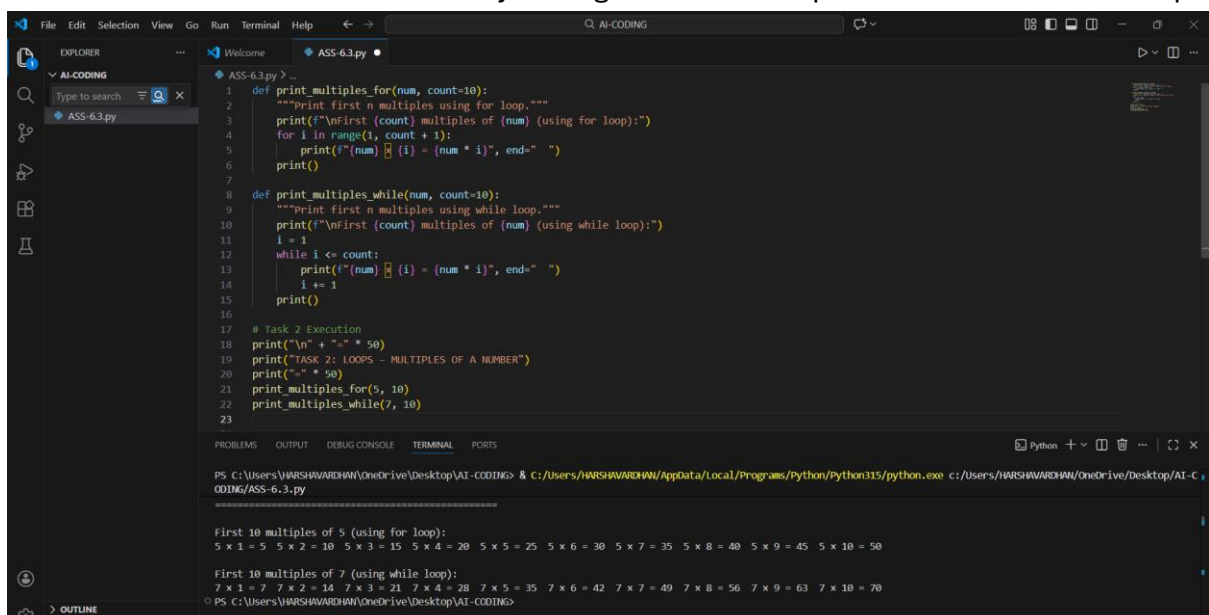
Expected Output #2

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

AI Prompt Used

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

“Generate the same functionality using a while loop instead of a for loop.”



The screenshot shows a VS Code editor with a file named `ASS-6.3.py`. The code defines two functions: `print_multiples_for` and `print_multiples_while`. The `print_multiples_for` function uses a `for` loop to print the first 10 multiples of a given number. The `print_multiples_while` function uses a `while` loop to achieve the same result. Below the functions, there is a section for task execution that calls both functions with the number 5 and 7. The terminal output shows the results of these function calls, displaying the first 10 multiples for both numbers using both loop structures.

```
1 def print_multiples_for(num, count=10):
2     """Print first n multiples using for loop."""
3     print(f"\nFirst {count} multiples of {num} (using for loop):")
4     for i in range(1, count + 1):
5         print(f"{num} x {i} = {num * i}", end=" ")
6     print()
7
8 def print_multiples_while(num, count=10):
9     """Print first n multiples using while loop."""
10    print(f"\nFirst {count} multiples of {num} (using while loop):")
11    i = 1
12    while i <= count:
13        print(f"{num} x {i} = {num * i}", end=" ")
14        i += 1
15    print()
16
17 # Task 2 Execution
18 print("\n" + "=" * 50)
19 print("TASK 2: LOOPS - MULTIPLES OF A NUMBER")
20 print("=" * 50)
21 print_multiples_for(5, 10)
22 print_multiples_while(7, 10)
23
```

Terminal Output:

```
PS C:\Users\HARSHAVARDHAN\OneDrive\Desktop\AI-CODING> & C:\Users\HARSHAVARDHAN\AppData\Local\Programs\Python\Python315\python.exe c:\Users\HARSHAVARDHAN\OneDrive\Desktop\AI-CODING\ASS-6.3.py
=====
First 10 multiples of 5 (using for loop):
5 x 1 = 5 5 x 2 = 10 5 x 3 = 15 5 x 4 = 20 5 x 5 = 25 5 x 6 = 30 5 x 7 = 35 5 x 8 = 40 5 x 9 = 45 5 x 10 = 50

First 10 multiples of 7 (using while loop):
7 x 1 = 7 7 x 2 = 14 7 x 3 = 21 7 x 4 = 28 7 x 5 = 35 7 x 6 = 42 7 x 7 = 49 7 x 8 = 56 7 x 9 = 63 7 x 10 = 70
PS C:\Users\HARSHAVARDHAN\OneDrive\Desktop\AI-CODING>
```

Justification

This prompt:

- Explicitly defines the **task scope** (first 10 multiples)
- Forces use of a **for loop**, allowing analysis of fixed-iteration looping
- Helps evaluate AI's understanding of **range-based iteration**
- Produces predictable and verifiable output for correctness checking
- Encourages the AI to use an **alternative looping construct**
- Enables **comparison between for and while loops**
- Helps analyze how AI handles **manual loop control and termination conditions**
- Supports the lab objective of evaluating different looping approaches

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).

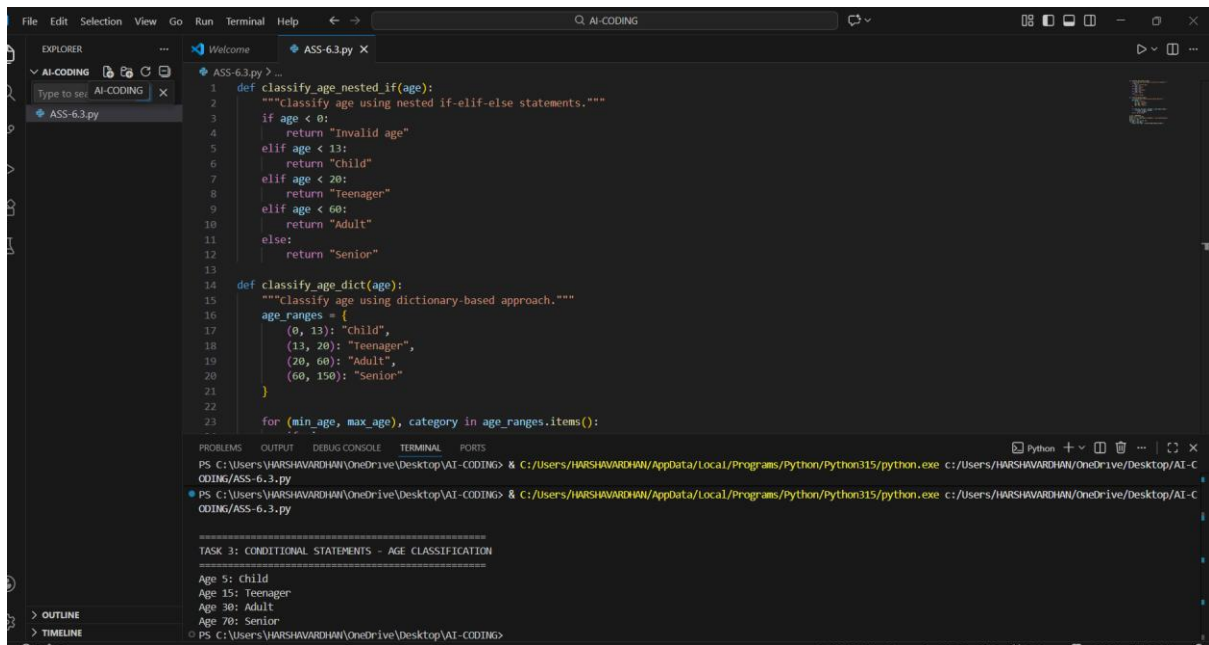
- Analyze the generated conditions and logic.
- Ask the AI to generate the same classification using alternative conditional structures (e.g., simplified conditions or dictionary-based logic).

Expected Output #3

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

AI Prompt Used

“Generate a Python function that uses nested if-elif-else statements to classify age into child, teenager, adult, and senior categories.” “Generate an alternative implementation for age classification using simplified conditions or a different logical structure.”



```
1 def classify_age_nested_if(age):
2     """Classify age using nested if-elif-else statements."""
3     if age < 0:
4         return "Invalid age"
5     elif age < 13:
6         return "Child"
7     elif age < 20:
8         return "Teenager"
9     elif age < 60:
10        return "Adult"
11    else:
12        return "Senior"
13
14 def classify_age_dict(age):
15     """Classify age using dictionary-based approach."""
16     age_ranges = {
17         (0, 13): "Child",
18         (13, 20): "Teenager",
19         (20, 60): "Adult",
20         (60, 150): "Senior"
21     }
22
23     for (min_age, max_age), category in age_ranges.items():
24         if min_age <= age < max_age:
25             return category
26     return "Invalid age"
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\HARSHAVARDHAN\OneDrive\Desktop\AI-CODING> & C:/Users/HARSHAVARDHAN/AppData/Local/Programs/Python/Python315/python.exe c:/Users/HARSHAVARDHAN/OneDrive/Desktop/AI-CODING/ASS-6.3.py

PS C:\Users\HARSHAVARDHAN\OneDrive\Desktop\AI-CODING> & C:/Users/HARSHAVARDHAN/AppData/Local/Programs/Python/Python315/python.exe c:/Users/HARSHAVARDHAN/OneDrive/Desktop/AI-CODING/ASS-6.3.py

=====

TASK 3: CONDITIONAL STATEMENTS - AGE CLASSIFICATION

=====

Age 5: Child
Age 15: Teenager
Age 30: Adult
Age 70: Senior

PS C:\Users\HARSHAVARDHAN\OneDrive\Desktop\AI-CODING>

Justification

This follow-up prompt:

- Encourages the AI to use an **alternative looping construct**
- Enables **comparison between for and while loops**
- Helps analyze how AI handles **manual loop control and termination conditions**
- Supports the lab objective of evaluating different looping approaches

This prompt was chosen to:

- Require **conditional branching logic**
- Ensure use of **nested if-elif-else statements**
- Test AI’s ability to create **non-overlapping and logically ordered conditions**
- Provide a real-world classification problem suitable for control-flow analysis

- Encourages AI to explore **different logical designs**
- Helps assess **code readability and maintainability**
- Allows comparison between **traditional conditionals and optimized logic**
- Aligns with the lab goal of critical evaluation of AI-generated solutions

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.

AI Prompt Used

“Generate a Python function `sum_to_n()` that calculates the sum of the first n natural numbers using a for loop. “Suggest an alternative implementation of the same function using a while loop or a mathematical formula.”

```
1
2 def sum_to_n_for(n):
3     """Calculate sum of first n natural numbers using for loop."""
4     total = 0
5     for i in range(1, n + 1):
6         total += i
7     return total
8
9 def sum_to_n_while(n):
10    """Calculate sum of first n natural numbers using while loop."""
11    total = 0
12    i = 1
13    while i <= n:
14        total += i
15        i += 1
16    return total
17
18 def sum_to_n_formula(n):
19    """Calculate sum using mathematical formula: n(n+1)/2."""
20    return n * (n + 1) // 2
21
22 # Task 4 Execution
23 print("\n" + "-" * 50)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\HARSHAVARDHAN\OneDrive\Desktop\AI-CODING> & C:/Users/HARSHAVARDHAN/AppData/Local/Programs/Python/Python315/python.exe c:/Users/HARSHAVARDHAN/OneDrive/Desktop/AI-CODING/ASS-6.3.py

TASK 4: FOR AND WHILE LOOPS - SUM OF FIRST N NUMBERS

Sum of first 10 numbers (for loop): 55
Sum of first 10 numbers (while loop): 55
Sum of first 10 numbers (formula): 55

Justification

This prompt:

- Clearly defines **input, output, and looping structure**
- Forces AI to implement **iterative accumulation**
- Makes it easy to test correctness with sample values
- Demonstrates how AI handles **basic algorithmic problems**

This prompt:

- Encourages AI to suggest **more efficient or varied solutions**
- Helps analyze **time complexity differences**
- Supports comparison between **iterative and formula-based approaches**
- Reinforces understanding of optimization concepts

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.

Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.

AI Prompt Used

“Generate a Python BankAccount class with methods deposit(), withdraw(), and check_balance(). Include validation logic and demonstrate usage with sample transactions.”

```
1 class BankAccount:
2     """A class to represent a bank account with deposit and withdrawal operations."""
3
4     def __init__(self, account_holder, initial_balance=0):
5         """Initialize bank account with account holder name and balance."""
6         self.account_holder = account_holder
7         self.balance = initial_balance
8
9     def deposit(self, amount):
10        """Deposit money into the account."""
11        if amount > 0:
12            self.balance += amount
13            print(f"✓ Deposited: Rs. {amount}")
14        else:
15            print("X Invalid deposit amount")
16
17    def withdraw(self, amount):
18        """Withdraw money from the account."""
19        if amount > 0 and amount <= self.balance:
20            self.balance -= amount
21            print(f"✓ Withdrawn: Rs. {amount}")
22        else:
23            print("X Invalid withdrawal amount or insufficient balance")
24
```

PS C:\Users\HARSHAVARDHAN\OneDrive\Desktop\AI-CODING> & C:/Users/HARSHAVARDHAN/AppData/Local/Programs/Python/Python315/python.exe c:/Users/HARSHAVARDHAN/OneDrive/Desktop/AI-CODING/ASS-6.3.py

Account Balance: Rs. 5000
✓ Deposited: Rs. 2000
Account Balance: Rs. 7000
✓ Withdrawn: Rs. 1500
Account Balance: Rs. 5500

=====

ALL TASKS COMPLETED SUCCESSFULLY

=====

© PS C:\Users\HARSHAVARDHAN\OneDrive\Desktop\AI-CODING>

Justification

This prompt was structured to:

- Require a **real-world class design**
- Test AI's ability to implement **state management (balance)**
- Include **input validation** for deposits and withdrawals
- Demonstrate **method interaction and object behavior**
- Produce code suitable for explanation and commenting

It aligns strongly with evaluating **AI-generated OOP logic and correctness**.