Name:P.Sai Venkatesh
H.NO: 2303A53039 - B46

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.

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.

PROMPT:

#Generate a Python class Student with attributes name, roll number, and branch. Include constructor and display_details() method. Create sample object and print details.

CODE:

```
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("Alice", "12345", "Computer Science")
# Print detailsstudent1.display_details()
student1.display_details()
```

OUTPUT:

```
Name: Alice
Roll Number: 12345
Branch: Computer Science
```

A6.3.py > ...

```python
#Generate a Python class Student with attributes name, roll number, and branch. Include con
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("Alice", "12345", "Computer Science")
# Print detailsstudent1.display_details()
student1.display_details()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
/usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/LAB Assignments/A6.3.py"
saivenkatesh@Sais-MacBook-Air LAB Assignments % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coo
Name: Alice
Roll Number: 12345
Branch: Computer Science
saivenkatesh@Sais-MacBook-Air LAB Assignments %
```

EXPLAINATION:

The Student class uses a constructor to initialize name, roll number, and branch, demonstrating basic object-oriented programming.The `display_details()` method prints student information clearly, showing how methods access object attributes.

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.

PROMPT:

#Generate Python function to print first 10 multiples of a number using loop.

CODE:

```
def print_multiples(number):
    for i in range(1, 11):
        print(f"{number} x {i} = {number * i}")
# Example usage
print_multiples(5)
```

OUTPUT:

```
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
```

A6.3.py > ...

```python

#Generate Python function to print first 10 multiples of a number using loop.
def print_multiples(number):
    for i in range(1, 11):
        print(f"{number} x {i} = {number * i}")
# Example usage
print_multiples(5)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
/usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/LAB Assignments/A6.3.py"
saivenkatesh@Sais-MacBook-Air LAB Assignments % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/LAB Assig
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
```

EXPLAINATION:

The loop iterates from 1 to 10 and multiplies the given number each time to generate its first ten multiples. Both `for` and `while` loops produce the same result, but the `for` loop is simpler while the `while` loop gives more control.

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.

PROMPT:

#Generate nested if-elif-else to classify age groups.

CODE:

```
def classify_age_group(age):
    if age < 0:
        print("Invalid age")
    elif age < 13:
        print("Child")
    elif age < 20:
        print("Teenager")
    elif age < 65:
        print("Adult")
    else:
        print("Senior Citizen")
# Example usage
classify_age_group(10)  # Output: Child
classify_age_group(16)  # Output: Teenager
classify_age_group(30)  # Output: Adult
classify_age_group(70)  # Output: Senior Citizen
```

OUTPUT:

Child
Teenager
Adult
Senior Citizen

```python
#Generate nested if-elif-else to classify age groups.
def classify_age_group(age):
    if age < 0:
        print("Invalid age")
    elif age < 13:
        print("Child")
    elif age < 20:
        print("Teenager")
    elif age < 65:
        print("Adult")
    else:
        print("Senior Citizen")
# Example usage
classify_age_group(10)  # Output: Child
classify_age_group(16)  # Output: Teenager
classify_age_group(30)  # Output: Adult
classify_age_group(70)  # Output: Senior Citizen
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

/usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/LAB Assignments/A6.3.py"
saivenkatesh@Sais-MacBook-Air LAB Assignments % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/LAB Assignments/A6.3.py"
Child
Teenager
Adult
Senior Citizen
```

EXPLAINATION:

The conditional statements check age ranges sequentially to classify a person into child, teenager, adult, or senior.
The alternative dictionary-based logic separates classification rules from conditions, improving readability and maintainability.

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.

PROMPT:

#Generate a Python function sum_to_n(n) that calculates the sum of the first n natural numbers using a for loop.
#Analyze the generated code for correctness and clarity. Then provide an alternative implementation using a while loop and another using the mathematical formula n(n+1)/2, with example usage and output.

CODE:

```
def sum_to_n_for_loop(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
# Example usage
print(sum_to_n_for_loop(10))  # Output: 55
def sum_to_n_while_loop(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total
# Example usage
print(sum_to_n_while_loop(10))  # Output: 55
```

```
def sum_to_n_formula(n):
    return n * (n + 1) // 2
# Example usage
print(sum_to_n_formula(10))  # Output: 55
```

OUTPUT:

```
55
55
55
```

A6.3.py > ...

```
60  def sum_to_n_for_loop(n):
62      for i in range(1, n + 1):
63          total += i
64      return total
65  # Example usage
66  print(sum_to_n_for_loop(10))  # Output: 55
67  def sum_to_n_while_loop(n):
68      total = 0
69      i = 1
70      while i <= n:
71          total += i
72          i += 1
73      return total
74  # Example usage
75  print(sum_to_n_while_loop(10))  # Output: 55
76  def sum_to_n_formula(n):
77      return n * (n + 1) // 2
78  # Example usage
79  print(sum_to_n_formula(10))  # Output: 55
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CO

```
/usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/LAB Assignmen
saivenkatesh@Sais-MacBook-Air LAB Assignments % /usr/bin/python3 "/Us
55
55
55
```

EXPLAINATION:

The loop-based approach adds numbers from 1 to n iteratively, demonstrating basic accumulation using control structures.
The mathematical formula computes the sum instantly in constant time, making it more efficient than loop methods.

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.

PROMPT:

#Generate BankAccount class with deposit, withdraw, and check_balance methods.

CODE:

```
class BankAccount:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
```

```
        if amount > 0:
            self.balance += amount
            print(f"Deposited: {amount}. New Balance: {self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds.")
        elif amount <= 0:
            print("Withdrawal amount must be positive.")
        else:
            self.balance -= amount
            print(f"Withdrew: {amount}. New Balance: {self.balance}")

    def check_balance(self):
        print(f"Current Balance: {self.balance}")
# Example usage
account = BankAccount("John Doe", 1000)
account.check_balance()  # Output: Current Balance: 1000
account.deposit(500)     # Output: Deposited: 500. New Balance: 1500
account.withdraw(200)    # Output: Withdrew: 200. New Balance: 1300
account.withdraw(1500)   # Output: Insufficient funds
account.deposit(-100)    # Output: Deposit amount must be positive.
account.withdraw(-50)    # Output: Withdrawal amount must be
                            positive.
```

OUTPUT:

```
Current Balance: 1000
Deposited: 500. New Balance: 1500
Withdrew: 200. New Balance: 1300
Insufficient funds.
Deposit amount must be positive.
Withdrawal amount must be positive.
```

```python
A6.3.py > ...
class BankAccount:
    def deposit(self, amount):
        print(f"Deposited: {amount}. New Balance: {self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds.")
        elif amount <= 0:
            print("Withdrawal amount must be positive.")
        else:
            self.balance -= amount
            print(f"Withdrew: {amount}. New Balance: {self.balance}")

    def check_balance(self):
        print(f"Current Balance: {self.balance}")
# Example usage
account = BankAccount("John Doe", 1000)
account.check_balance()  # Output: Current Balance: 1000
account.deposit(500)     # Output: Deposited: 500. New Balance: 1500
account.withdraw(200)    # Output: Withdrew: 200. New Balance: 1300
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

/usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/LAB Assignments/A6.3.py"
saivenkatesh@Sais-MacBook-Air LAB Assignments % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/LAB Assignments/A6.3.py"
Current Balance: 1000
Deposited: 500. New Balance: 1500
Withdrew: 200. New Balance: 1300
Insufficient funds.
Deposit amount must be positive.
Withdrawal amount must be positive.
saivenkatesh@Sais-MacBook-Air LAB Assignments %
```

EXPLAINATION:

The BankAccount class models real banking operations using methods for deposit, withdrawal, and balance checking with validation.
It prevents invalid transactions and demonstrates encapsulation by controlling balance updates through class methods.

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.