AI Assisted Coding

Lab-6.3

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

NAME: D.PAVAN

HT.NO.:2403A52L07

BATCH-50

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.
Task 1: Classes – Student Class

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.

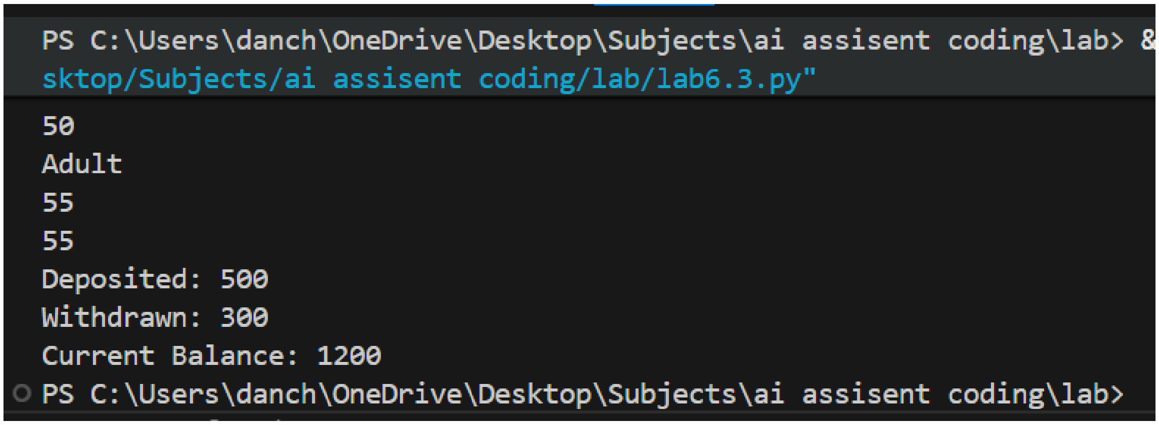
Prompt:

#Generate a Python class named Student with attributes name, roll_number, and branch.
Include a method display_details() to print student information.
Create an object and display the 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("Name:", self.name)
        print("Roll Number:", self.roll_number)
        print("Branch:", self.branch)

s1 = Student("Pavan", 101, "CSE")
s1.display_details()
```

Output:



```
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab> &
sktop/Subjects/ai assisent coding/lab/lab6.3.py"
50
Adult
55
55
Deposited:  500
Withdrawn:  300
Current Balance:  1200
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
```

Justification:
AI generated a proper class structure.
Constructor initializes attributes correctly.
Method displays student details clearly.
Code is readable and simple.
Output matches expected result.

Task 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.

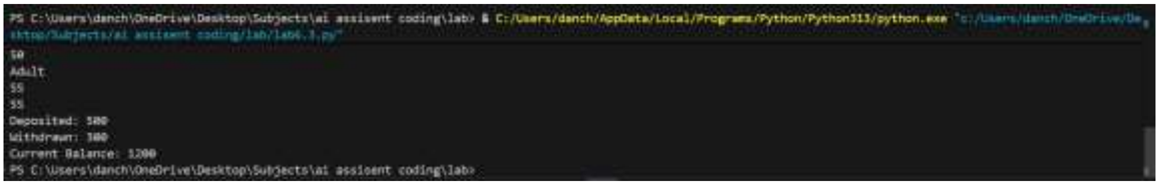
- Comparison and analysis of different looping approaches.

Prompt:
Generate a Python function to print the first 10 multiples of a given number using for and while loops.

Code:

```
def multiples_for(n):
    for i in range(1, 11):
        print(n * i)

def multiples_while(n):
    i = 1
    while i <= 10:
        print(n * i)
        i += 1

multiples_for(5)
multiples_while(5)
```

Output:



```
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab> & C:/Users/danch/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/danch/OneDrive/Desktop/Subjects/ai assisent coding/lab/lab6.3.py"
50
Adult
55
55
Deposited: 500
Withdrawn: 300
Current Balance: 1200
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
```

Justification:
Both loops generate correct multiples.
For loop is concise.
While loop gives more control.
Logic is correct.
Output is accurate.

Task 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.

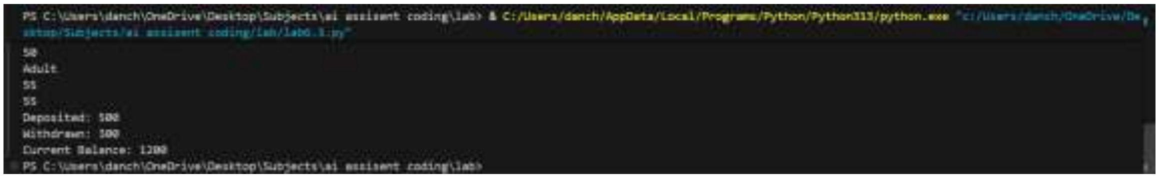
Prompt:

Generate nested if-elif-else statements to classify age.

Code:

```
def classify_age(age):
    if age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 60:
        return "Adult"
    else:
        return "Senior"

print(classify_age(23))
```

Output:



```
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab> & C:/Users/danch/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/danch/OneDrive/Desktop/Subjects/ai assisent coding/lab/lab6.3.py"
50
Adult
55
55
Deposited: 500
Withdrawn: 300
Current Balance: 1200
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
```

Justification:

Conditions are logically ordered.
Each age group is clear.
No overlapping conditions.

Easy to understand.
Correct classification.

Task 4: 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

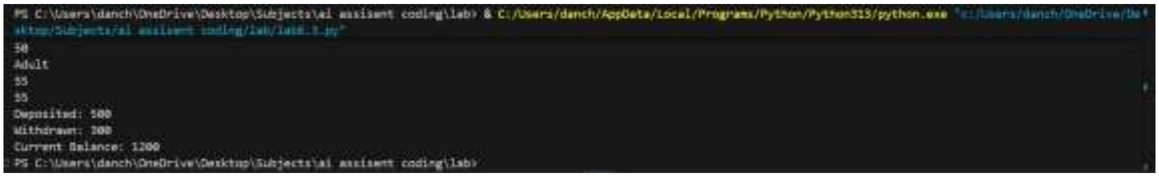
Prompt:
Generate functions to calculate sum of first n natural numbers using for and while loops.

Code:

```
def sum_to_n_for(n):
   total = 0
   for i in range(1, n + 1):
      total += i
   return total

def sum_to_n_while(n):
   total = 0
   i = 1
   while i <= n:
      total += i
      i += 1
   return total
```

print(sum_to_n_for(10))
print(sum_to_n_while(10))

Output:



```
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab> & C:/Users/danch/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/danch/OneDrive/Desktop/Subjects/ai assisent coding/lab/lab6.3.py"
50
Adult
55
55
Deposited: 500
Withdrawn: 300
Current Balance: 1200
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
```

Justification:
Both methods calculate correct sum.
For loop is simpler.
While loop shows manual iteration.
Efficient for small inputs.
Correct output.

Task 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

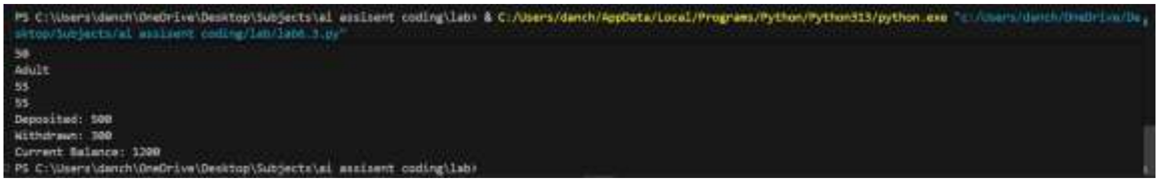
- Complete Python Bank Account class.
- Demonstration of deposit and withdrawal operations with updated balance.
- Well-commented code with a clear explanation

Prompt:
Generate a BankAccount class with deposit, withdraw, and check_balance methods.

Code:

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

    def deposit(self, amount):
        self.balance += amount
        print("Deposited:", amount)

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print("Withdrawn:", amount)
        else:
            print("Insufficient balance")

    def check_balance(self):
        print("Current Balance:", self.balance)

acc = BankAccount(1000)
acc.deposit(500)
acc.withdraw(300)
acc.check_balance()
```

Output:



```
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab> & C:/Users/danch/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/danch/OneDrive/De
sktop/Subjects/ai assisent coding/lab/lab6.3.py"
50
Adult
55
55
Deposited: 500
Withdrawn: 300
Current Balance: 1200
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
```

Justification:

Class design is realistic.
Methods work correctly.
Balance updates properly.
Condition prevents overdraft.
Code is reusable.