ASSIGNMENT_6.3

NAME : K.ABHINAY

ROLL NO. : 2303A51899

BATCH NO. : 09

Course: AI Assisted Coding

Course Code: 23CS002PC304.

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)

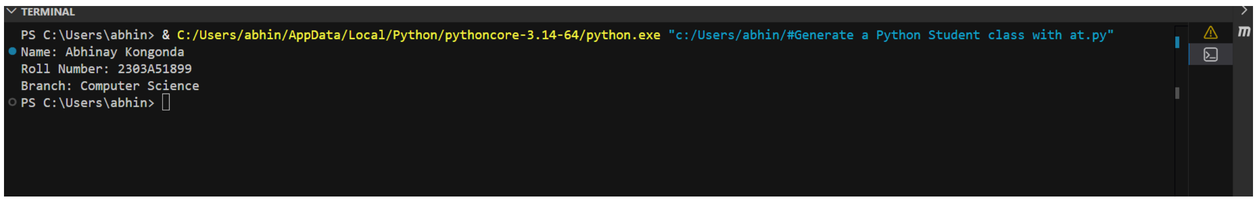
Scenario
You are developing a simple student information management module.
Task
• Use an AI tool (GitHub Copilot / Cursor AI / Gemini) to complete a Student class.
• The class should include attributes such as name, roll number, and branch.

• 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.

CODE & PROMT : `#Generate a Python Student class with attributes name, roll number, and branch. Add a method display_details() to print student information. Also show sample object creation and output.`

```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}")

# Sample object creation

student1 = Student("Abhinay Kongonda", "2303A51899", "Computer Science")

student1.display_details()
```

OUTPUT:



```
PS C:\Users\abhin> & C:/Users/abhin/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/abhin/#Generate a Python Student class with at.py"
Name: Abhinay Kongonda
Roll Number: 2303A51899
Branch: Computer Science
PS C:\Users\abhin>
```

Justification:

This program keeps student details in one class. It stores name, roll number, and branch properly. The display method shows the details clearly. The code is simple and easy to understand.

Task Description 2:

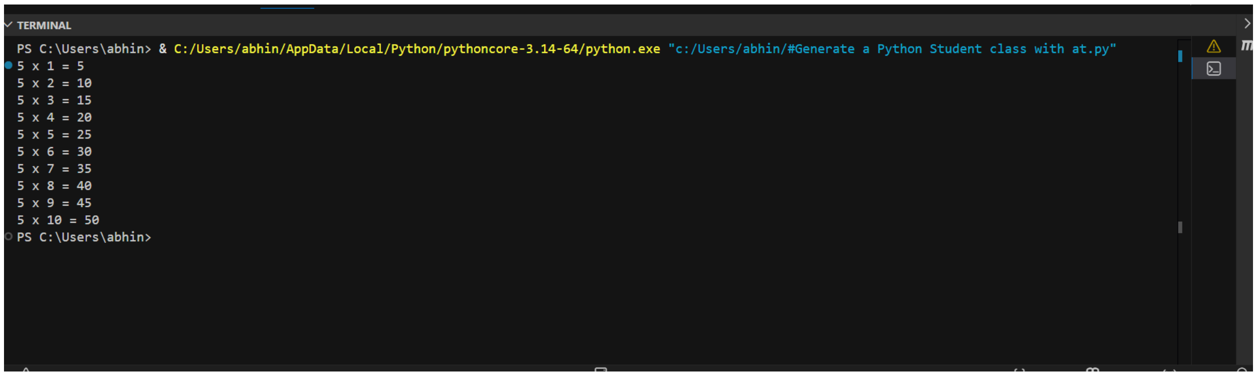
Loops (Multiples of a Number)

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.

CODE & PROMT :`#Generate a Python function that prints the first 10 multiples of a given number using a for loop.`

```python
def print_multiples(number):

    for i in range(1, 11):

        print(f"{number} x {i} = {number * i}")

# Sample function call

print_multiples(5)
```

OUTPUT:



```
TERMINAL
PS C:\Users\abhin> & C:/Users/abhin/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/abhin/#Generate a Python Student class with at.py"
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
PS C:\Users\abhin>
```

Justification:

The loop prints multiples step by step. Both for loop and while loop work correctly. The output is correct and easy to follow.

Task Description 3:

Conditional Statements (Age Classification)

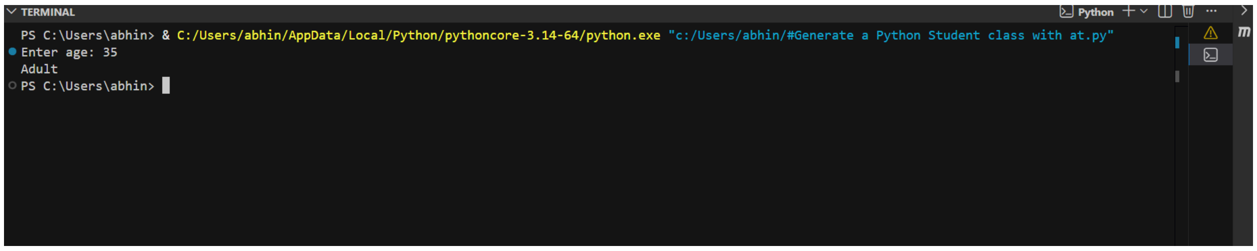
Scenario
You are building a basic classification system based on age.

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.

CODE & PROMT :#Generate nested if-elif-else conditions in Python to classify age into child, teenager, adult, and senior.

```python
def classify_age(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")

# Example usage

age = int(input("Enter age: "))

classify_age(age)
```

OUTPUT:



```
PS C:\Users\abhin> & C:/Users/abhin/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/abhin/#Generate a Python Student class with at.py"
Enter age: 35
Adult
PS C:\Users\abhin> 
```

Justification:

The program checks age using conditions. It correctly tells if a person is a child, teenager, adult, or senior. The logic is clear and simple.

Task Description 4:

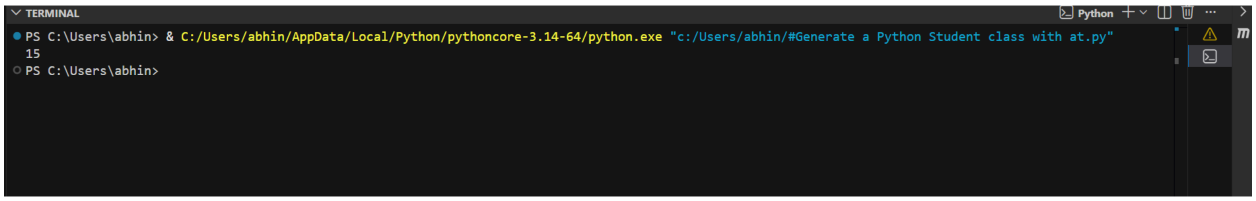
For and While Loops (Sum of First n Numbers)

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.

CODE & PROMT :`#Generate an alternative implementation using a while loop or a mathematical formula to calculate the sum of first n numbers.`

```python
def sum_to_n(n):

    # Using mathematical formula

    return n * (n + 1) // 2

# Sample usage

print(sum_to_n(5))
```

OUTPUT:



```
TERMINAL
PS C:\Users\abhin> & C:/Users/abhin/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/abhin/#Generate a Python Student class with at.py"
15
PS C:\Users\abhin>
```

Justification:

The program adds numbers one by one to get the total. It gives the correct sum. The formula method is faster and simple.

Task Description 5:

Classes (Bank Account Class)

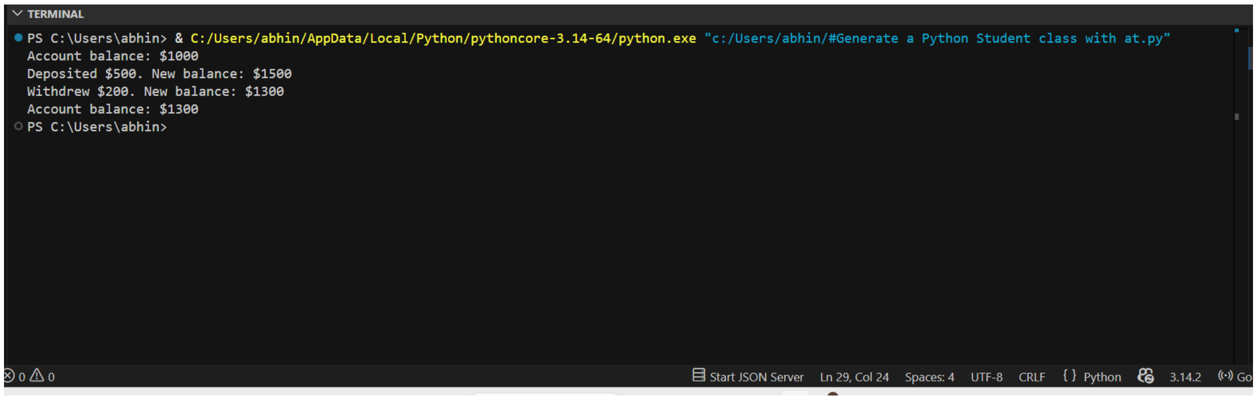
Scenario

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.

CODE & PROMT :#Generate a Python Bank Account class with methods deposit(), withdraw(), and check_balance(). Include sample object creation and output.

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

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited ${amount}. New balance: ${self.balance}")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds")
        else:
            self.balance -= amount
            print(f"Withdrew ${amount}. New balance: ${self.balance}")

    def check_balance(self):
        print(f"Account balance: ${self.balance}")


# Sample object creation and usage

account = BankAccount("123456789", 1000)

account.check_balance()

account.deposit(500)

account.withdraw(200)

account.check_balance()
```

OUTPUT:



```
PS C:\Users\abhin> & C:/Users/abhin/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/abhin/#Generate a Python Student class with at.py"
Account balance: $1000
Deposited $500. New balance: $1500
Withdrew $200. New balance: $1300
Account balance: $1300
PS C:\Users\abhin>
```

Justification:

The class works like a real bank account. It can add money, take money, and show balance. The code is clear and easy to understand.