AI ASSISTANT CODING

Assignment-6.3

NAME : Irumalla Hasini

HTNO: 2303A51286

BATCH : 05

TASK-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 :

a simple student information management module.

#The class should include attributes such as name, roll number, and branch.

#Add a method display_details() to print student information.

EXPLAINATION :

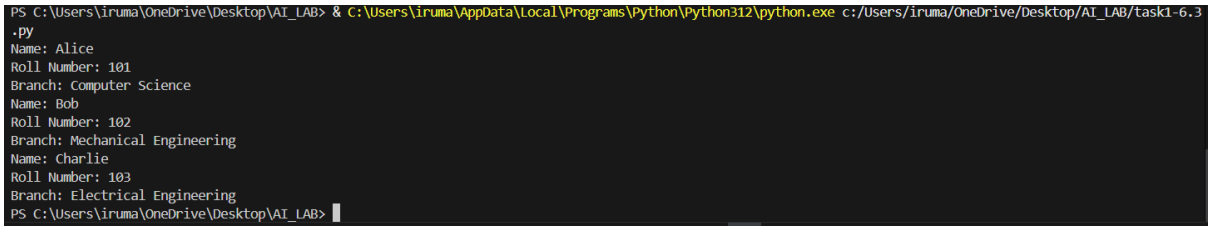
- An AI tool was used to generate a Python Student class.
- The class contains attributes: name, roll number, and branch.
- The __init__ constructor initializes student details.
- The display_details() method prints the student information.
- Object creation and method execution produced correct output.
- The AI-generated code was clear, readable, and correctly structured.

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}")
# Example usage
student1 = Student("Alice", "101", "Computer Science")
student1.display_details()
student2 = Student("Bob", "102", "Mechanical Engineering")
student2.display_details()
student3 = Student("Charlie", "103", "Electrical Engineering")
student3.display_details()
```

OUTPUT:



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).

PROMPT :

#utility function to display multiples of a given number.

#a function that prints the first 10 multiples of a given number using a loop.

EXPLAINATION :

- AI generated a function to print the first 10 multiples of a given number.
- A for loop was used to iterate from 1 to 10.
- Each iteration multiplies the number with the loop variable.
- A while loop version was also generated for comparison.
- Both approaches produced the same correct output.
- The for loop was more concise and readable.

CODE: USING FOR LOOP

```
def display_multiples(number):
    print(f"First 10 multiples of {number}:")
    for i in range(1, 11):
        multiple = number * i
        print(multiple)
# Example usage
display_multiples(5)
```

OUTPUT:

PROMPT :

#utility function to display multiples of a given number.

#a function that prints the first 10 multiples of a given number using while loop.

CODE : USING WHILE LOOP

```
def display_multiples(number):
    count = 1
    while count <= 10:
        multiple = number * count
        print(f"{number} x {count} = {multiple}")
        count += 1
# Example usage
display_multiples(5)
```

OUTPUT :

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).

PROMPT :

#a basic classification system based on age.

#nested if-elif-else conditional statements to classify age groups (e.g., child, teenager, adult, senior).

EXPLAINATION:

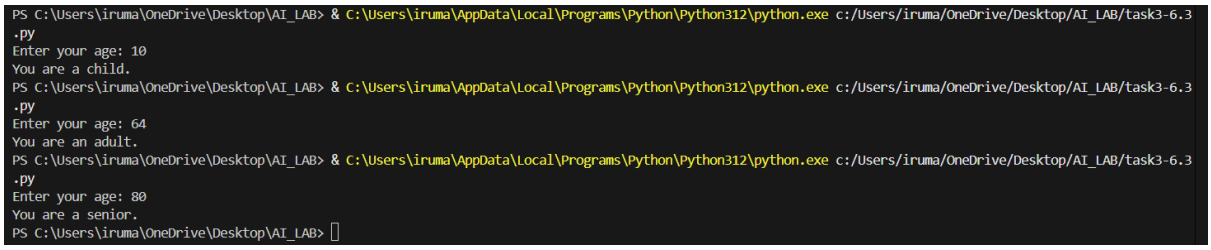
- AI generated if-elif-else conditions to classify age groups.
- Age categories included child, teenager, adult, and senior.
- Conditions were logically ordered to avoid incorrect classification.
- An alternative simplified conditional approach was suggested.
- The output correctly classified the age.
- The logic was clear and easy to understand.

CODE:

```
age = int(input("Enter your age: "))
if age < 0:
    print("Invalid age.")
elif age <= 12:
    print("You are a child.")
elif age <= 19:
    print("You are a teenager.")
```

```
elif age <= 64:
    print("You are an adult.")
else:
    print("You are a senior.")
```

OUTPUT :



TASK-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:

the sum of the first n natural numbers.

#a sum_to_n() function using a for loop.

EXPLAINATION :

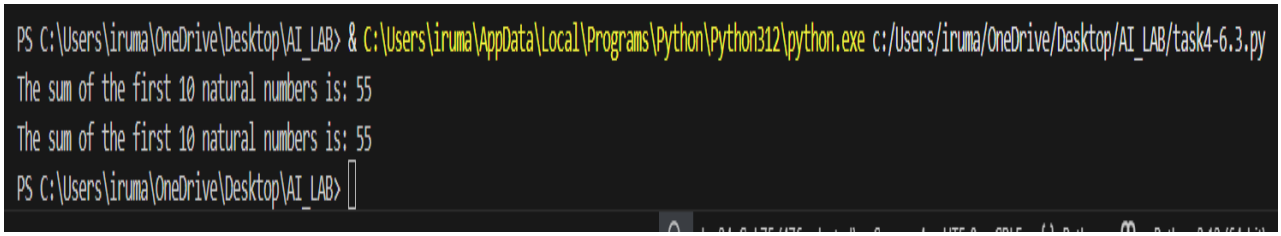
- AI generated a function to calculate the sum of first n natural numbers.
- A for loop was used to add numbers from 1 to n.

- A while loop version was also implemented.
- AI suggested a mathematical formula as an optimized approach.
- All methods produced correct results.
- Loop-based methods were easier to understand.

CODE:

```
ef sum_to_n(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
# Example usage:
n = 10
print(f"The sum of the first {n} natural numbers is: {sum_to_n(n)}")
#a sum_to_n() function using a while loop.
def sum_to_n_while(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total
# Example usage:
n = 10
print(f"The sum of the first {n} natural numbers is: {sum_to_n_while(n)}")
```

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.

PROMPT :

#Classes (Bank Account Class)

#basic banking application.

#a Bank Account class with methods such as deposit(), withdraw(), and check_balance()

EXPLAINATION :

- AI generated a BankAccount class for basic banking operations.
- The constructor initializes account holder name and balance.
- deposit() method adds amount to the balance.
- withdraw() method checks balance before deduction.
- check_balance() displays the current balance.
- The code was well-structured and properly commented.

CODE:

```
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        self.account_holder = account_holder
        self.balance = initial_balance
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited: ${amount:.2f}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew: ${amount:.2f}")
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        print(f"Current balance: ${self.balance:.2f}")
# Example usage:
if __name__ == "__main__":
    account = BankAccount("John Doe", 1000)
```

```
account.check_balance()
account.deposit(500)
account.check_balance()
account.withdraw(200)
account.check_balance()
account.withdraw(2000)  # Attempt to withdraw more than the balance
account.deposit(-50)     # Attempt to deposit a negative amount
account.check_balance()
```

OUTPUT :