AI Assisted Coding

Assignment 6.3

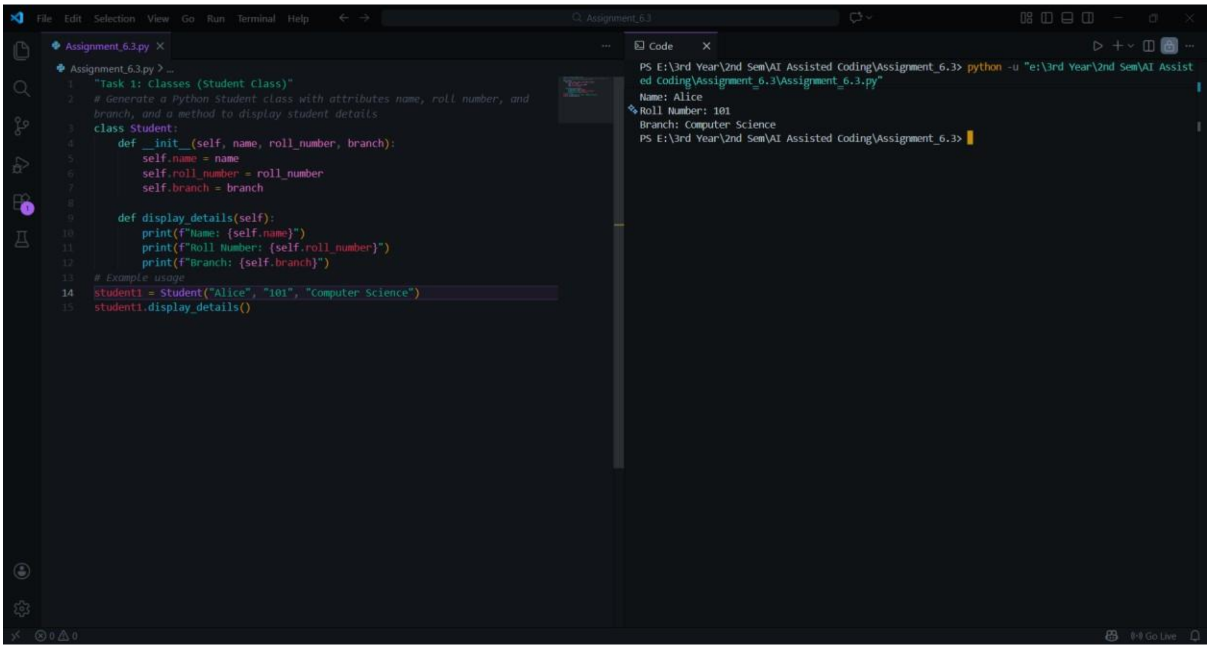
Name: ch. koushik
Hall ticket no: 2303a51938
Batch no: 19

Task 1: Classes (Student Class)

Prompt:

Generate a Python Student class with attributes name, roll number, and branch, and a method to display student details.

Code & Output:



```python
"Task 1: Classes (Student Class)"
# Generate a Python Student class with attributes name, roll number, and branch, and a method to display student details
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()
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3\Assignment_6.3.py"
Name: Alice
Roll Number: 101
Branch: Computer Science
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3>
```

Explanation:

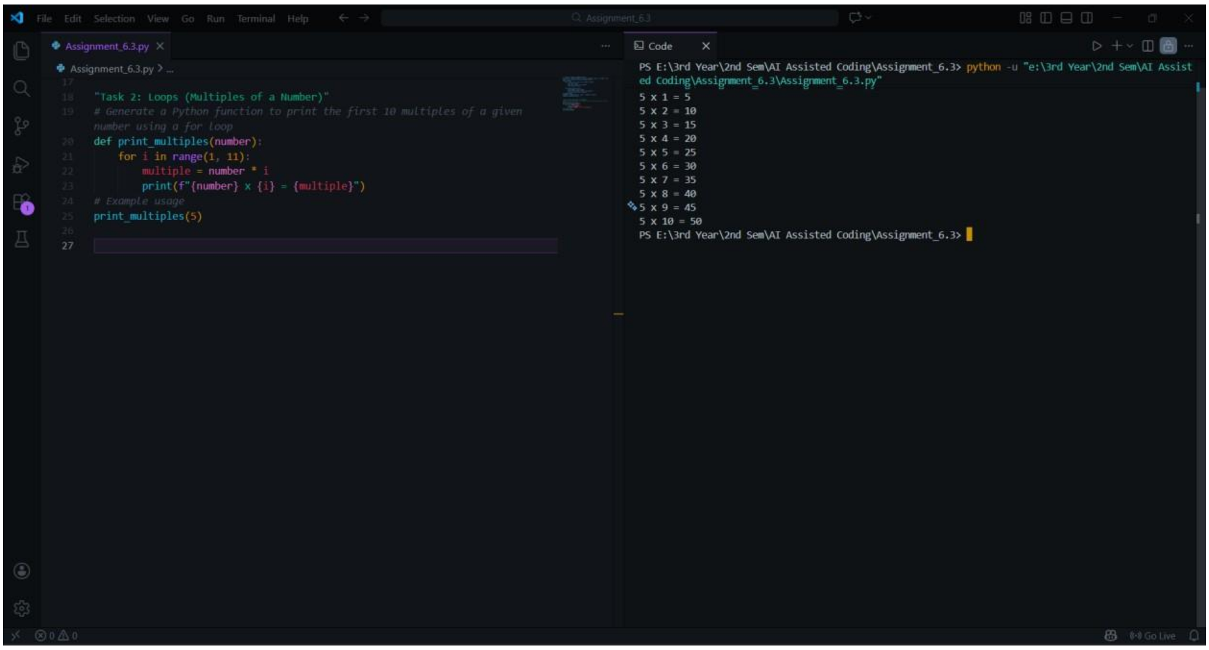
The AI-generated code correctly defines a Student class using object-oriented principles. The constructor initializes student attributes, and the display_details() method prints them clearly. The class structure is simple, readable, and functions correctly when an object is created and executed.

Task 2: Loops (Multiples of a Number)

Prompt:

Generate a Python function to print the first 10 multiples of a given number using a for loop.

Code & Output:



```python
"Task 2: Loops (Multiples of a Number)"
# Generate a Python function to print the first 10 multiples of a given
number using a for loop
def print_multiples(number):
    for i in range(1, 11):
        multiple = number * i
        print(f"{number} x {i} = {multiple}")
# Example usage
print_multiples(5)
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3> python -u "e:\3rd Year\2nd Sem\AI Assist
ed Coding\Assignment_6.3\Assignment_6.3.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 E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3>
```

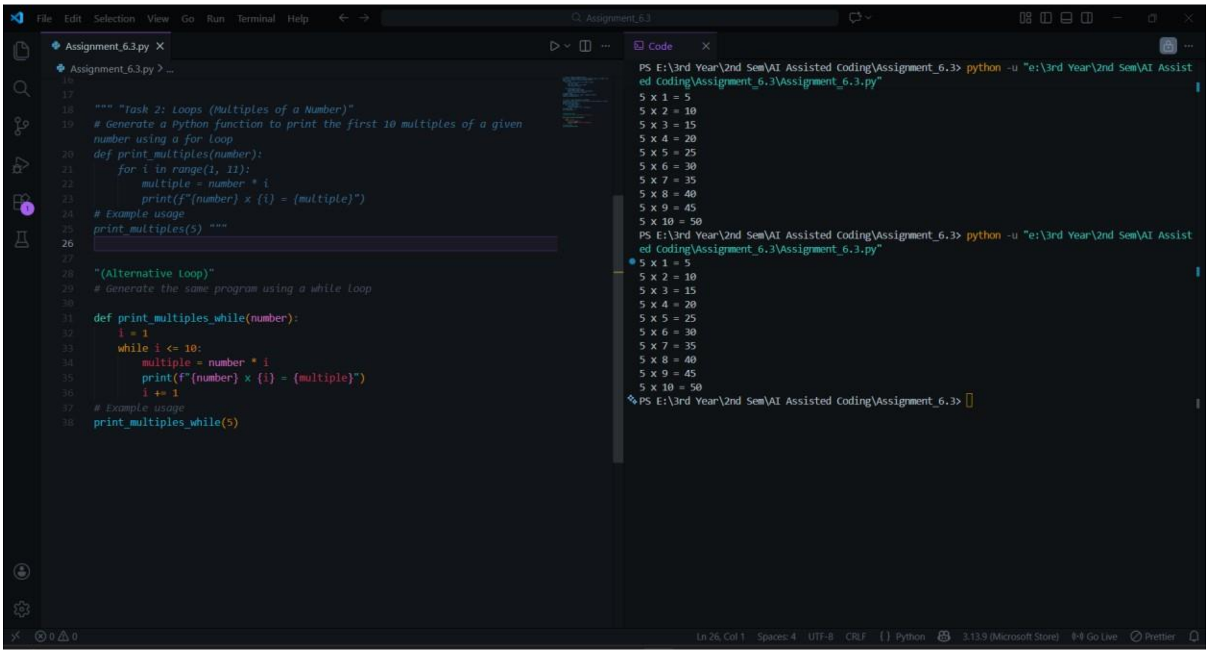
Explanation:

The AI-generated function uses a for loop to iterate from 1 to 10 and prints the multiples of the given number. In each iteration, the loop variable is multiplied by the input number. The loop boundaries are correctly defined, and the logic produces accurate results. This implementation is efficient and readable, making it ideal for tasks with a fixed number of iterations.

Prompt (Alternative Loop):

Generate the same program using a while loop

Code & Output:



```python
""" "Task 2: Loops (Multiples of a Number)"
# Generate a Python function to print the first 10 multiples of a given number using a for loop
def print_multiples(number):
    for i in range(1, 11):
        multiple = number * i
        print(f"{number} x {i} = {multiple}")
# Example usage
print_multiples(5) """


"(Alternative Loop)"
# Generate the same program using a while loop

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

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3\Assignment_6.3.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 E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3\Assignment_6.3.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 E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3>
```

Explanation:

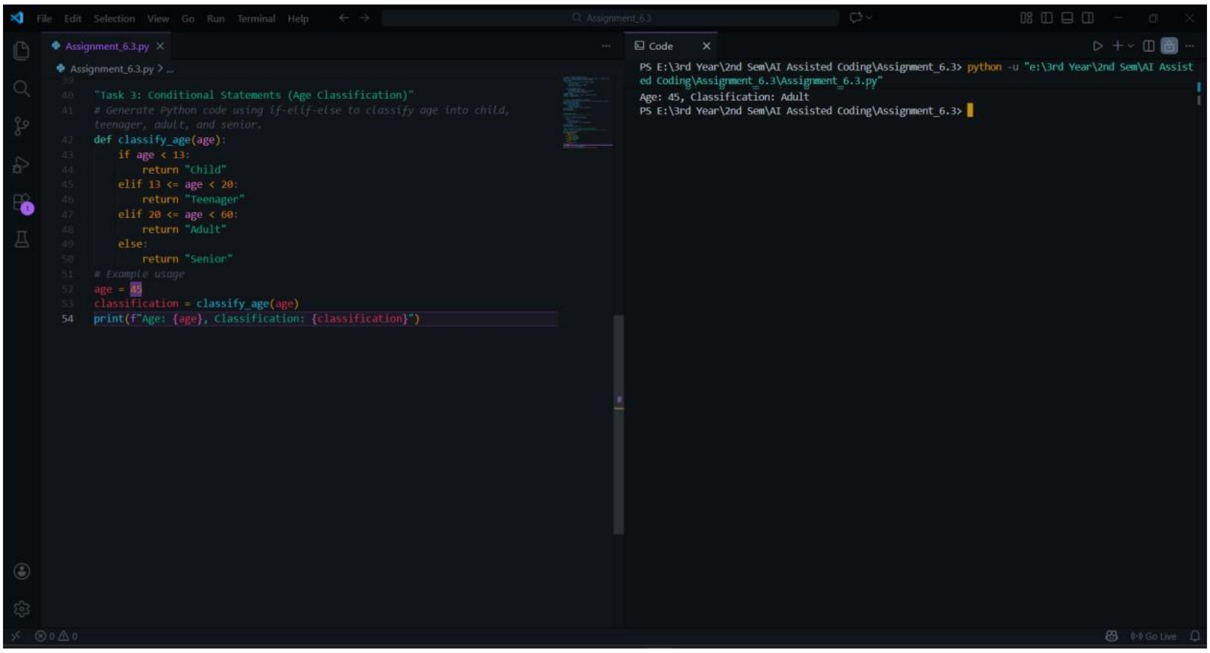
The while-loop version produces the same output as the for-loop version. While loops require manual control of the counter variable, making the for loop slightly cleaner and more readable for fixed iterations.

Task 3: Conditional Statements (Age Classification)

Prompt:

Generate Python code using if-elif-else to classify age into child, teenager, adult, and senior.

Code & Output:



```python
"Task 3: Conditional Statements (Age Classification)"
# Generate Python code using if-elif-else to classify age into child, teenager, adult, and senior.
def classify_age(age):
    if age < 13:
        return "Child"
    elif 13 <= age < 20:
        return "Teenager"
    elif 20 <= age < 60:
        return "Adult"
    else:
        return "Senior"
# Example usage
age = 45
classification = classify_age(age)
print(f"Age: {age}, Classification: {classification}")
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3\Assignment_6.3.py"
Age: 45, Classification: Adult
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3>
```

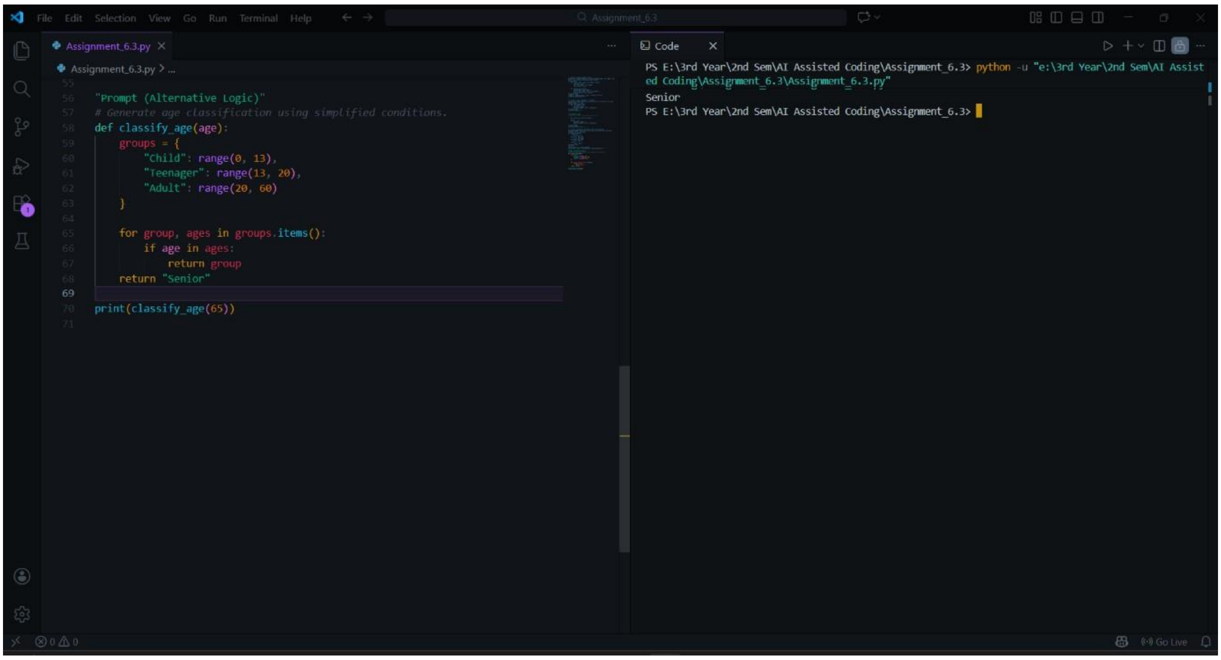
Explanation:

The AI-generated function uses nested if-elif-else conditions to classify age groups. Each condition checks a specific age range in increasing order. The structure ensures that only one category is returned for a given age. The logic is clear, correct, and easy to verify, making the code understandable for beginners and suitable for real-world classification tasks.

Prompt (Alternative Logic):

Generate age classification using simplified conditions.

Code & Output:



```python

"Prompt (Alternative Logic)"
# Generate age classification using simplified conditions.
def classify_age(age):
    groups = {
        "Child": range(0, 13),
        "Teenager": range(13, 20),
        "Adult": range(20, 60)
    }

    for group, ages in groups.items():
        if age in ages:
            return group
    return "Senior"

print(classify_age(65))

```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3> python -u "e:\3rd Year\2nd Sem\AI Assist
ed Coding\Assignment_6.3\Assignment_6.3.py"
Senior
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3>
```

Explanation:

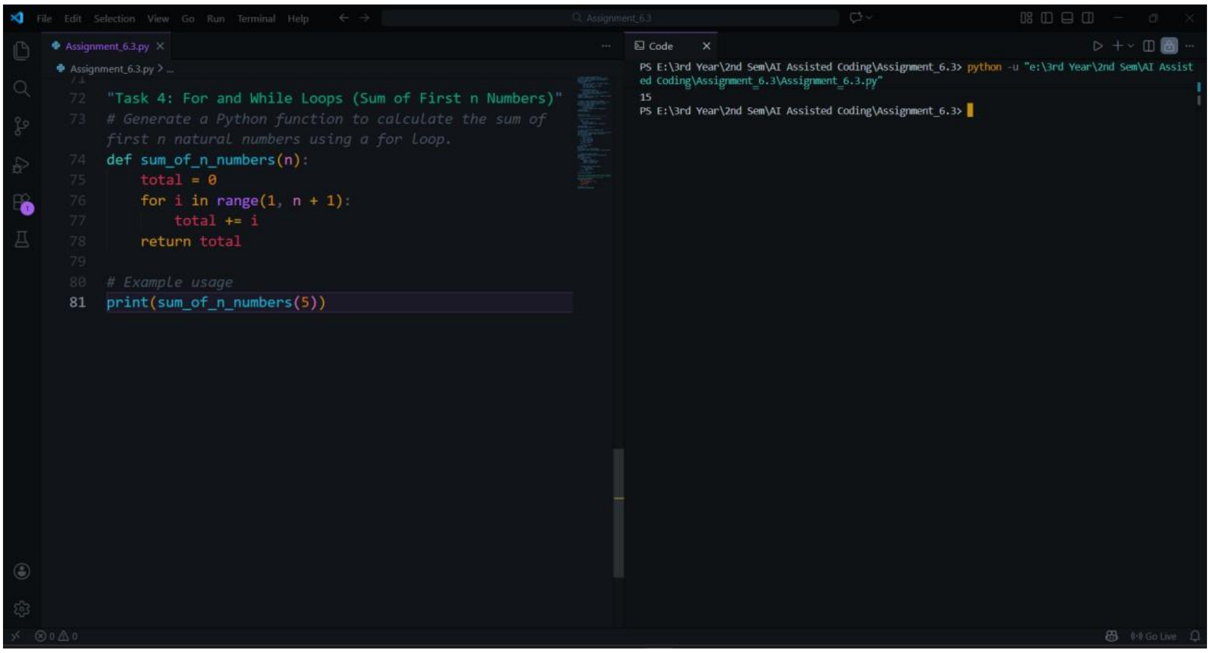
This alternative approach uses a dictionary and a loop to determine the age group. While this method is flexible and scalable, it is more complex than the if-elif-else approach. For simple classification problems, the original conditional structure is more readable and easier to maintain.

Task 4: For and While Loops (Sum of First n Numbers)

Prompt:

Generate a Python function to calculate the sum of first n natural numbers using a for loop.

Code & Output:



```python
"Task 4: For and While Loops (Sum of First n Numbers)"
# Generate a Python function to calculate the sum of
first n natural numbers using a for loop.
def sum_of_n_numbers(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

# Example usage
print(sum_of_n_numbers(5))
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3> python -u "e:\3rd Year\2nd Sem\AI Assist
ed Coding\Assignment_6.3\Assignment_6.3.py"
15
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3>
```

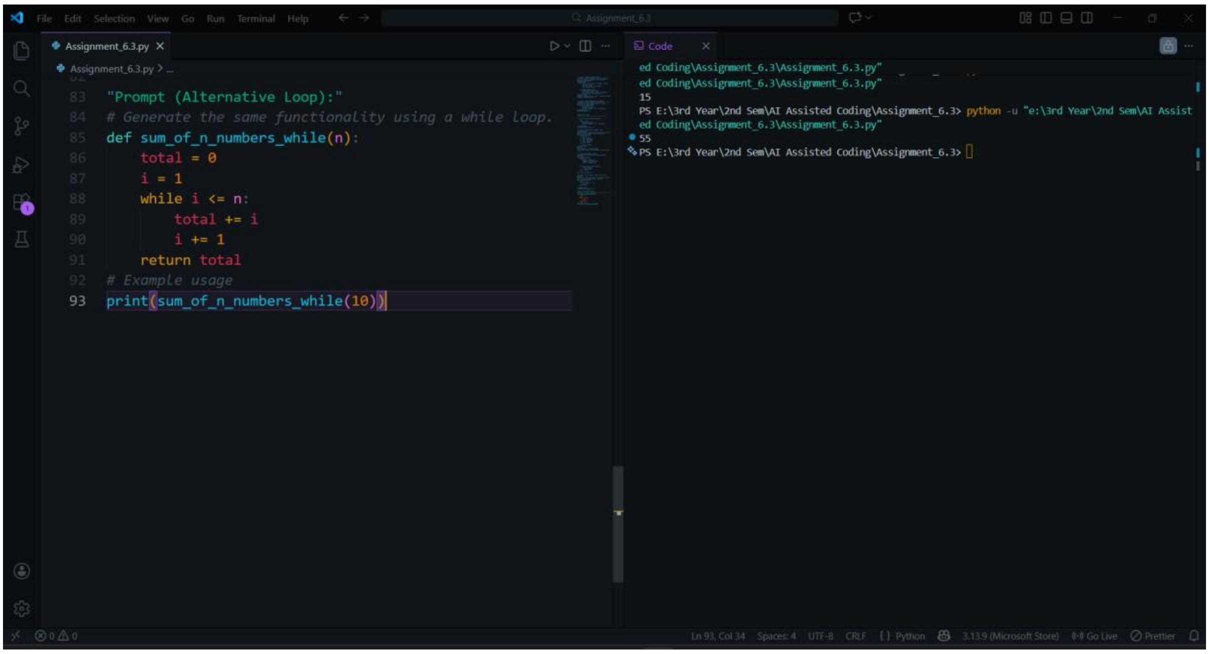
Explanation:

The AI-generated function calculates the sum by iterating through numbers from 1 to n and adding them to a total variable. The loop logic is correct and produces accurate results. This approach is easy to understand and works efficiently for small to moderate values of n.

Prompt (Alternative Loop):

Generate the same functionality using a while loop.

Code & Output:



```python
"Prompt (Alternative Loop):"
# Generate the same functionality using a while loop.
def sum_of_n_numbers_while(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total
# Example usage
print(sum_of_n_numbers_while(10))
```

```
ed Coding\Assignment_6.3\Assignment_6.3.py"
ed Coding\Assignment_6.3\Assignment_6.3.py"
15
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3> python -u "e:\3rd Year\2nd Sem\AI Assist
ed Coding\Assignment_6.3\Assignment_6.3.py"
55
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3>
```

Explanation:

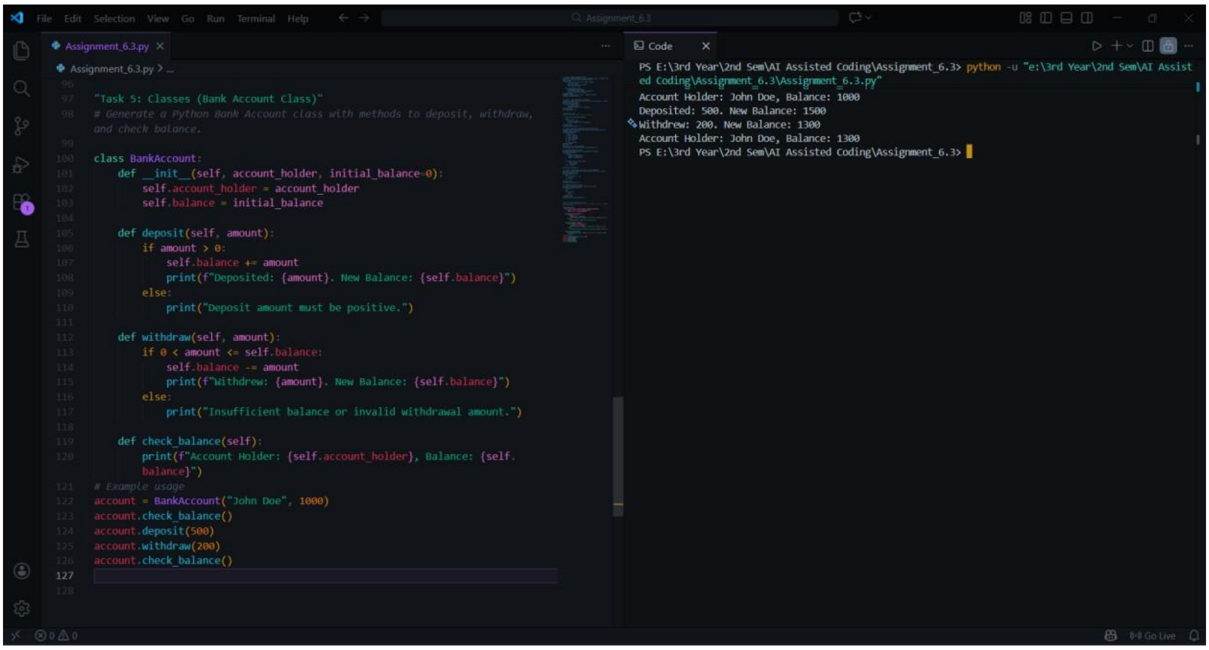
The while-loop version produces the same output as the for-loop version. While loops require manual control of the counter variable, making the for loop slightly cleaner and more readable for fixed iterations.

Task 5: Classes (Bank Account Class)

Prompt:

Generate a Python Bank Account class with methods to deposit, withdraw, and check balance.

Code & Output:



```python
"Task 5: Classes (Bank Account Class)"
# Generate a Python Bank Account class with methods to deposit, withdraw, and check balance.

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}. New Balance: {self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew: {amount}. New Balance: {self.balance}")
        else:
            print("Insufficient balance or invalid withdrawal amount.")

    def check_balance(self):
        print(f"Account Holder: {self.account_holder}, Balance: {self.balance}")
# Example usage
account = BankAccount("John Doe", 1000)
account.check_balance()
account.deposit(500)
account.withdraw(200)
account.check_balance()
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3> python -u "e:\3rd Year\2nd Sem\AI Assist
ed Coding\Assignment_6.3\Assignment_6.3.py"
Account Holder: John Doe, Balance: 1000
Deposited: 500. New Balance: 1500
Withdrew: 200. New Balance: 1300
Account Holder: John Doe, Balance: 1300
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_6.3>
```

Explanation:

The AI-generated Bank Account class demonstrates effective use of object-oriented programming. The constructor initializes the balance, and the methods allow depositing, withdrawing, and checking the balance. Conditional logic prevents withdrawal when the balance is insufficient. The code is clear, logically sound, and easy to extend, making it suitable for a basic banking application.

Final Conclusion:

This lab assignment demonstrates how AI-based code completion tools assist in generating Python programs using classes, loops, and conditional statements. Although AI accelerates coding, human review remains essential to verify correctness, readability, and efficiency.