

# AI Assisted Coding

## Assignment 6.3

Name: R.Bharadwaj

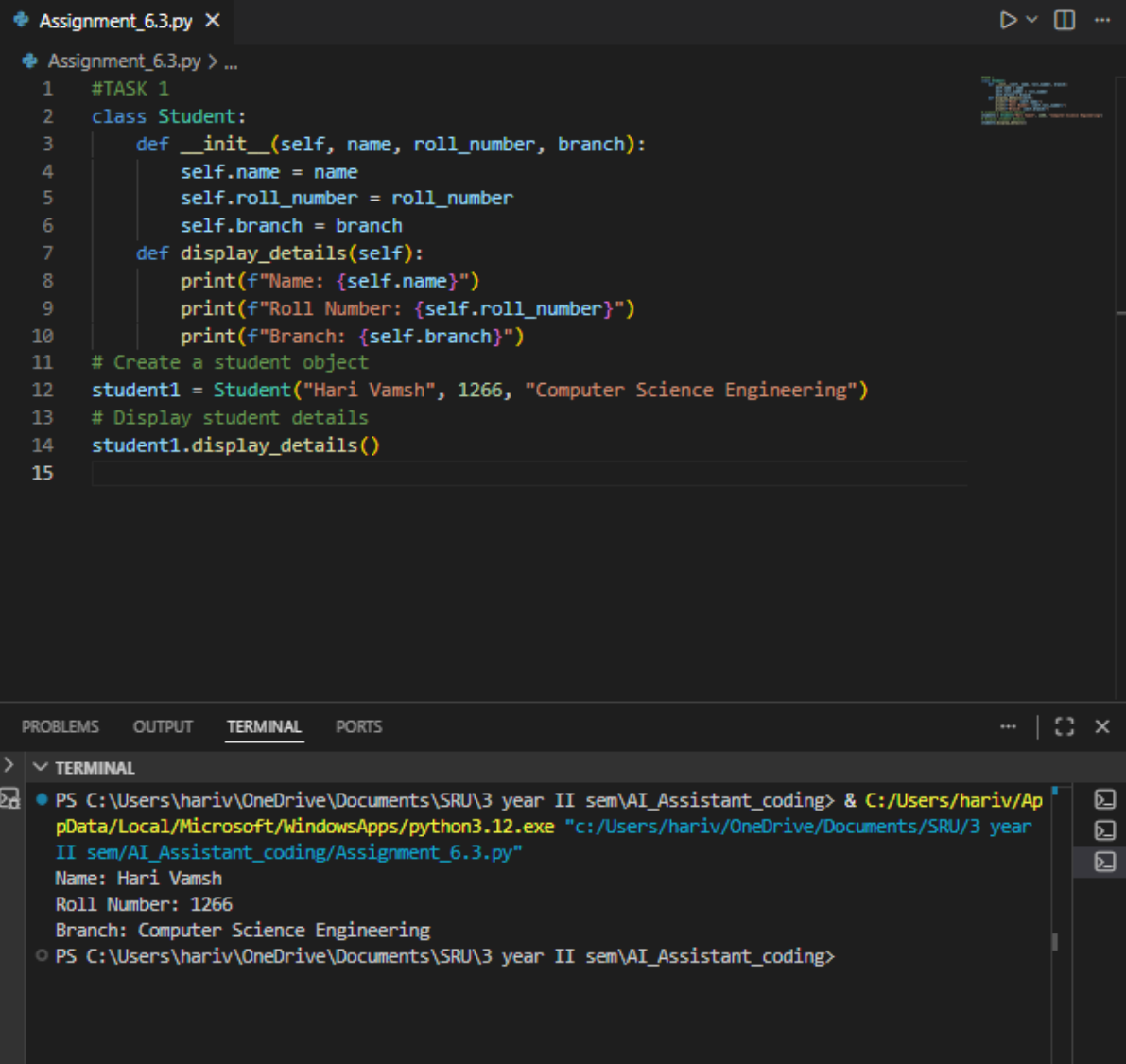
Hall ticket no: 2303A51610

Batch no: 19

### Task 1:

#### Prompt:

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



```
Assignment_6.3.py X
Assignment_6.3.py > ...
1 #TASK 1
2 class Student:
3     def __init__(self, name, roll_number, branch):
4         self.name = name
5         self.roll_number = roll_number
6         self.branch = branch
7     def display_details(self):
8         print(f"Name: {self.name}")
9         print(f"Roll Number: {self.roll_number}")
10        print(f"Branch: {self.branch}")
11 # Create a student object
12 student1 = Student("Hari Vamsh", 1266, "Computer Science Engineering")
13 # Display student details
14 student1.display_details()
15

PROBLEMS OUTPUT TERMINAL PORTS
> TERMINAL
• PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/Assignment_6.3.py"
Name: Hari Vamsh
Roll Number: 1266
Branch: Computer Science Engineering
○ PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

#### Explanation:

The AI-generated code properly implements a Student class by following object-oriented programming concepts. The constructor is used to initialize the student's details, and the

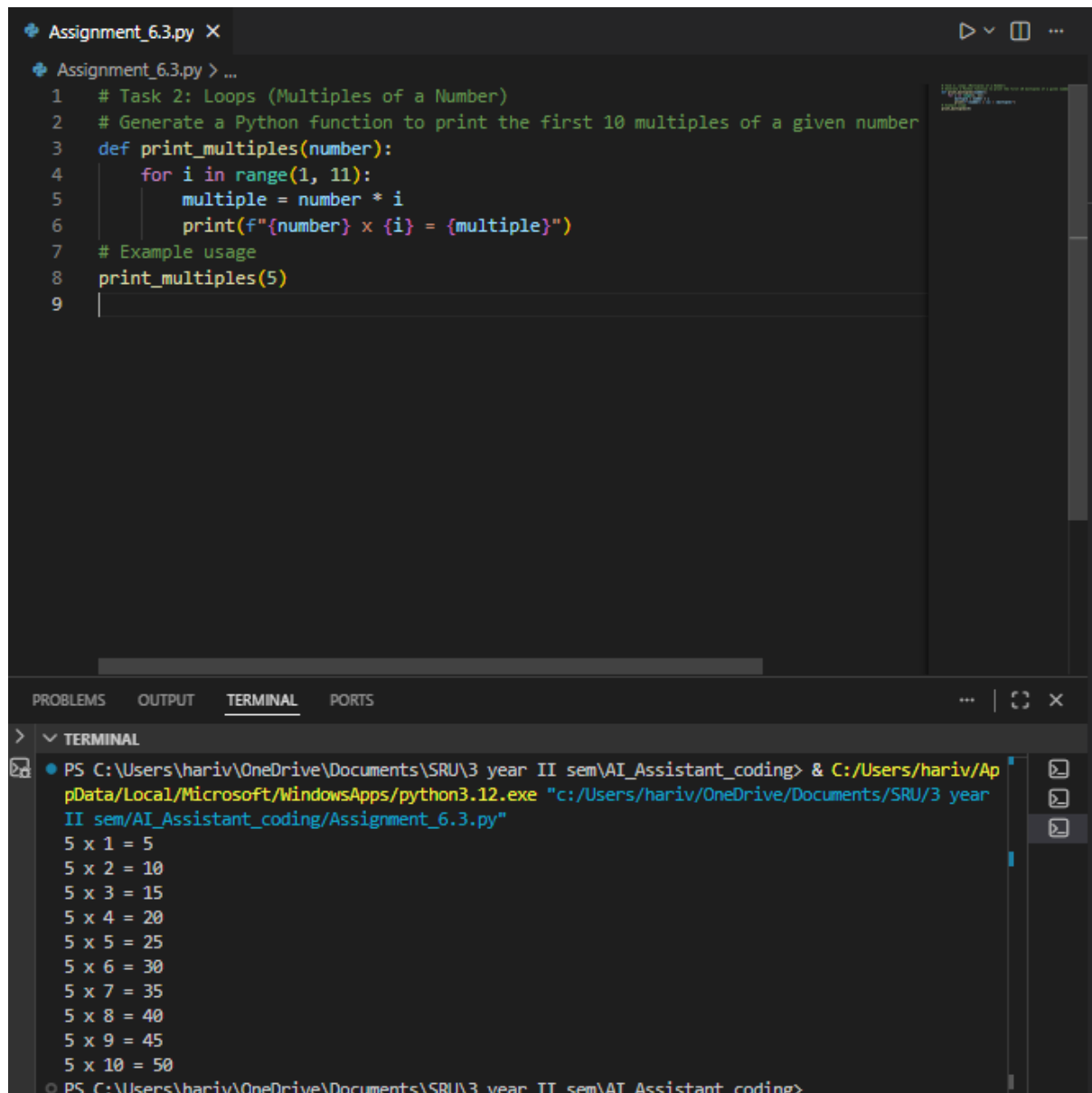
display\_details() method neatly outputs this information. Overall, the class design is clean, easy to understand, and works as expected when a student object is created and run.

## 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:



```
Assignment_6.3.py X
Assignment_6.3.py > ...
1 # Task 2: Loops (Multiples of a Number)
2 # Generate a Python function to print the first 10 multiples of a given number
3 def print_multiples(number):
4     for i in range(1, 11):
5         multiple = number * i
6         print(f"{number} x {i} = {multiple}")
7 # Example usage
8 print_multiples(5)
9 |

PROBLEMS OUTPUT TERMINAL PORTS
> TERMINAL
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/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 C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

### Explanation:

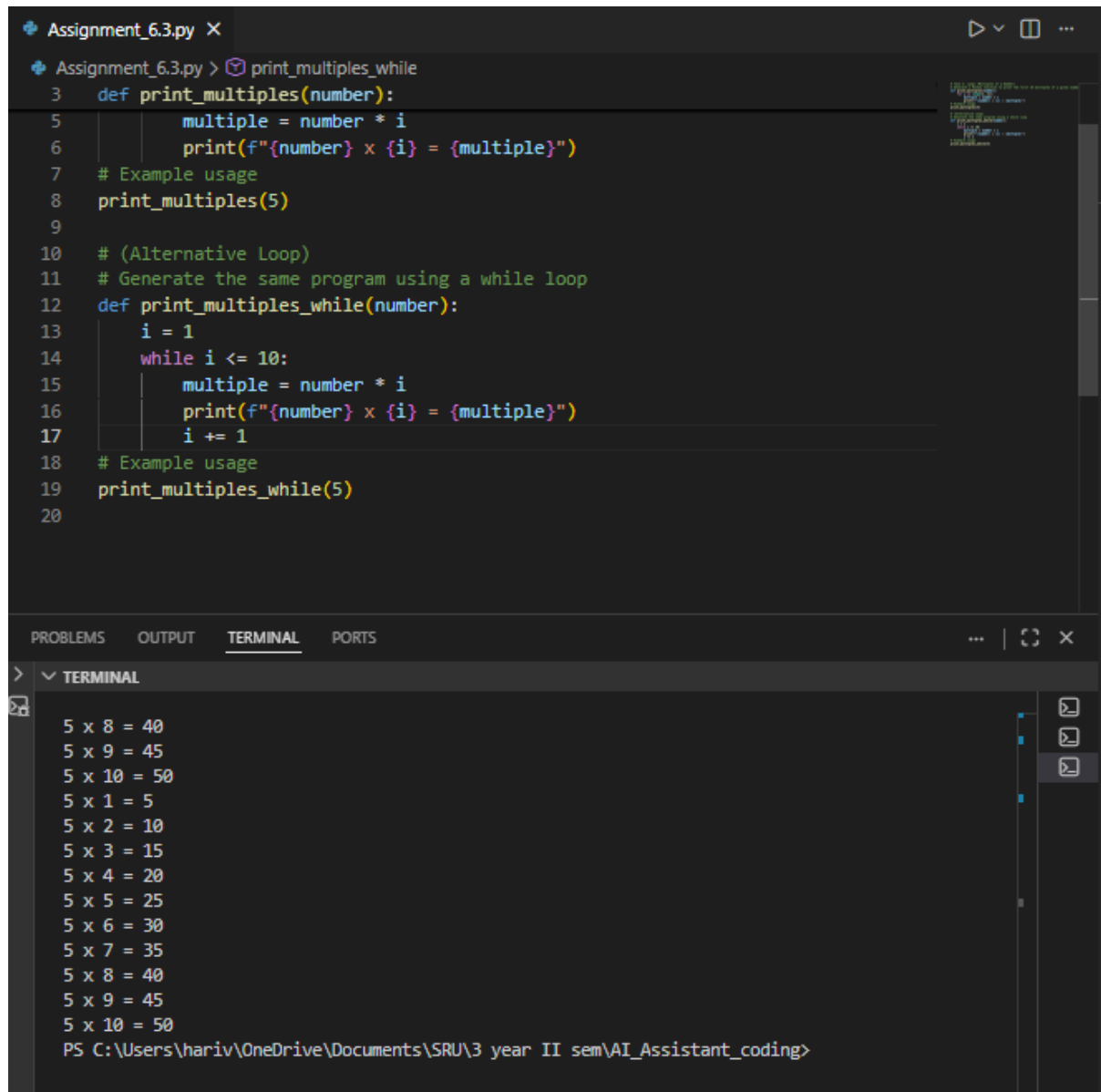
The AI-generated function uses a for loop to loop through values from 1 to 10 and display the multiples of the given number. During each loop cycle, the current value is multiplied by the input number to generate the result. The loop limits are clearly set, and the logic works

correctly to produce accurate output. Overall, the code is simple, efficient, and easy to read, making it well-suited for situations that require a fixed number of repetitions.

### Prompt (Alternative Loop):

Generate the same program using a while loop

### Code & Output:



```
Assignment_6.3.py X
Assignment_6.3.py > print_multiples_while
3 def print_multiples(number):
4     multiple = number * i
5     print(f"{number} x {i} = {multiple}")
6
7 # Example usage
8 print_multiples(5)
9
10 # (Alternative Loop)
11 # Generate the same program using a while loop
12 def print_multiples_while(number):
13     i = 1
14     while i <= 10:
15         multiple = number * i
16         print(f"{number} x {i} = {multiple}")
17         i += 1
18
19 # Example usage
20 print_multiples_while(5)
```

PROBLEMS OUTPUT **TERMINAL** PORTS

TERMINAL

```
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
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\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

### Explanation:

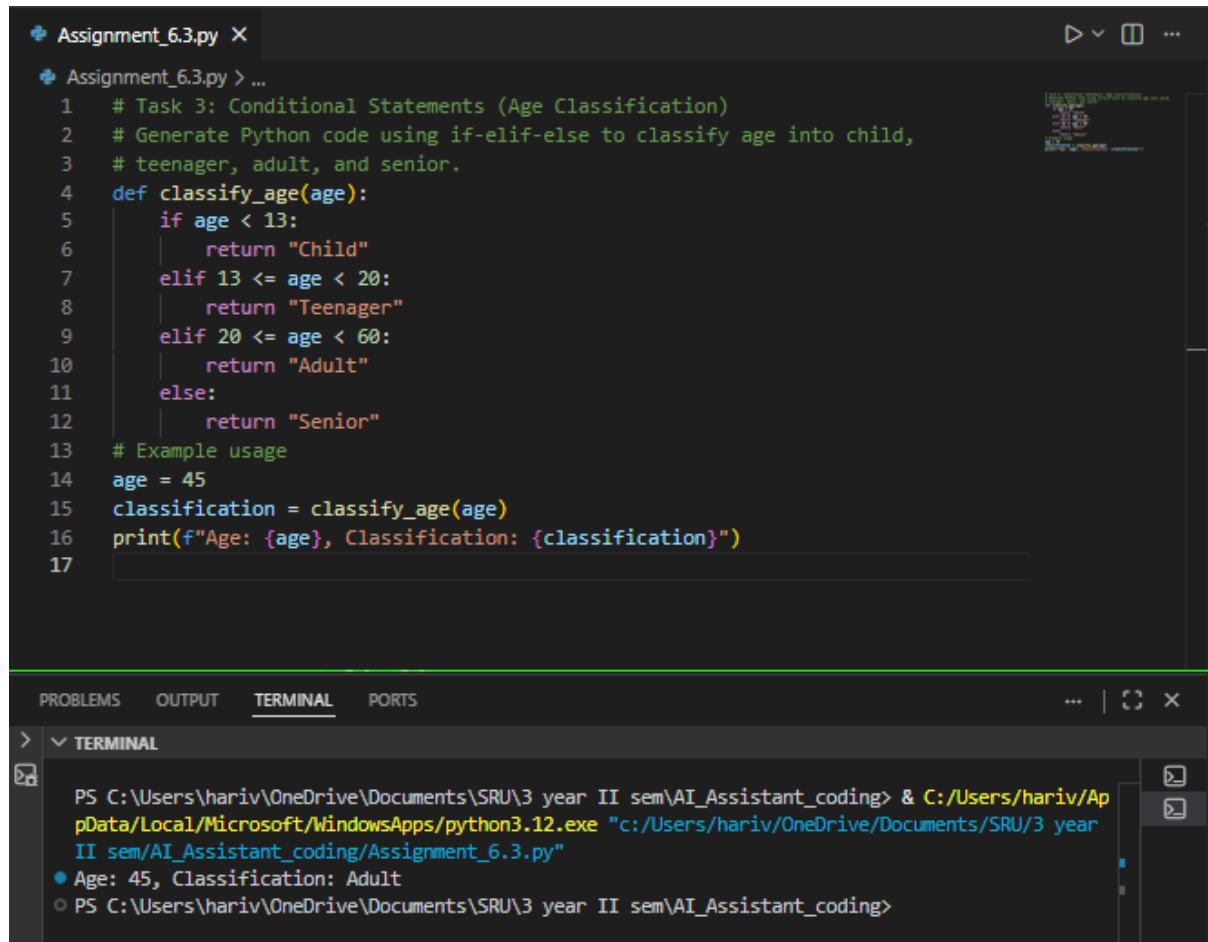
The while-loop version produces the same output as the for-loop version. While loops require manual control of the counter variable, making the for loop slightly cleaner and more readable for fixed iterations.

### 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:



```
Assignment_6.3.py X
Assignment_6.3.py > ...
1 # Task 3: Conditional Statements (Age Classification)
2 # Generate Python code using if-elif-else to classify age into child,
3 # teenager, adult, and senior.
4 def classify_age(age):
5     if age < 13:
6         return "Child"
7     elif 13 <= age < 20:
8         return "Teenager"
9     elif 20 <= age < 60:
10        return "Adult"
11    else:
12        return "Senior"
13 # Example usage
14 age = 45
15 classification = classify_age(age)
16 print(f"Age: {age}, Classification: {classification}")
17
```

PROBLEMS OUTPUT **TERMINAL** PORTS

```
> TERMINAL
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/Assignment_6.3.py"
● Age: 45, Classification: Adult
○ PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

#### Explanation:

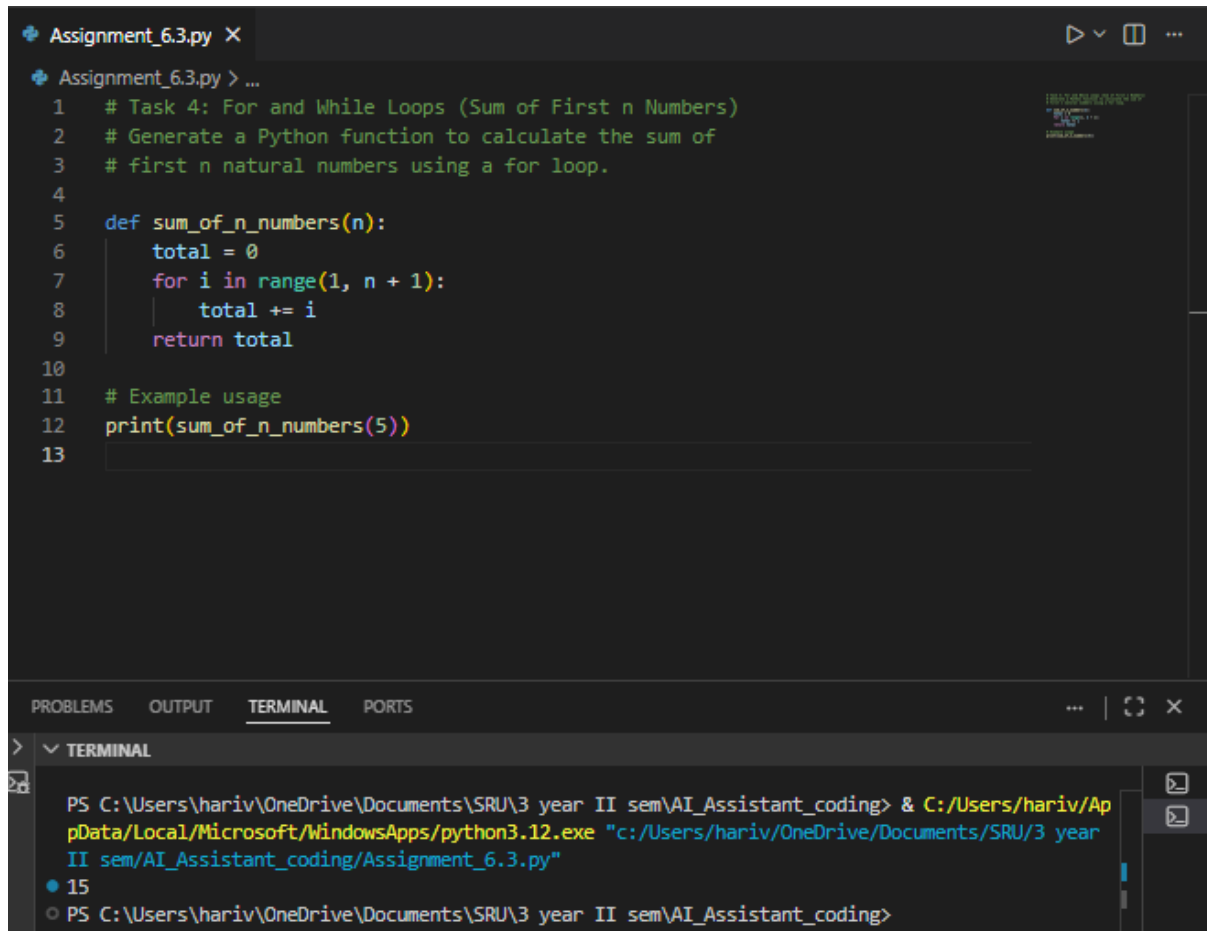
The AI-generated function applies nested if-elif-else statements to categorize different age groups. Each condition evaluates a particular age range in a logical sequence, ensuring that only one group is assigned for any given age. The approach is straightforward and accurate, making the logic easy to follow and verify. This clear structure makes the code beginner-friendly and practical for real-world age classification scenarios.

## 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:



```
Assignment_6.3.py X
Assignment_6.3.py > ...
1 # Task 4: For and While Loops (Sum of First n Numbers)
2 # Generate a Python function to calculate the sum of
3 # first n natural numbers using a for loop.
4
5 def sum_of_n_numbers(n):
6     total = 0
7     for i in range(1, n + 1):
8         total += i
9     return total
10
11 # Example usage
12 print(sum_of_n_numbers(5))
13
```

PROBLEMS OUTPUT **TERMINAL** PORTS

```
> TERMINAL
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/Assignment_6.3.py"
● 15
○ PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

### Explanation:

The AI-generated function finds the sum by looping through the numbers from 1 to n and adding each value to a running total. The loop is implemented correctly and gives the right result. This method is simple, easy to follow, and works well for small to medium values of n.

## Task 5: Classes (Bank Account Class)

### Prompt:

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

### Code & Output:

```
Assignment_6.3.py X
Assignment_6.3.py > ...
1  # Task 5: Classes (Bank Account Class)
2  # Generate a Python Bank Account class with methods to deposit, withdraw,
3  # and check balance.
4
5  class BankAccount:
6      def __init__(self, account_holder, initial_balance=0):
7          self.account_holder = account_holder
8          self.balance = initial_balance
9
10     def deposit(self, amount):
11         if amount > 0:
12             self.balance += amount
13             print(f"Deposited: {amount}. New Balance: {self.balance}")
14         else:
15             print("Deposit amount must be positive.")
16
17     def withdraw(self, amount):
18         if 0 < amount <= self.balance:
19             self.balance -= amount
20             print(f"Withdrew: {amount}. New Balance: {self.balance}")
21         else:
22             print("Insufficient balance or invalid withdrawal amount.")
23
24     def check_balance(self):
25         print(f"Account Holder: {self.account_holder}, Balance: {self.balance}")
26
27     # Example usage
28     account = BankAccount("John Doe", 1000)
29     account.check_balance()
30     account.deposit(500)
31     account.withdraw(200)
32     account.check_balance()
33
```

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/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 C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

### Explanation:

The AI-generated Bank Account class effectively showcases object-oriented programming concepts. The constructor sets the initial balance, while the class methods handle deposits, withdrawals, and balance inquiries. Built-in conditional checks ensure that withdrawals are blocked when there are insufficient funds. Overall, the code is well-structured, logically correct, and easy to expand, making it a good fit for a simple banking system.