

# AI Assisted Coding

## Assignment 6.3

Name: P. Vineeth Kumar

Hall ticket no: 2303A51256

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:

The screenshot shows a code editor with a Python file named `Assignment_6.3.py` open. The code defines a `Student` class with an `__init__` method to initialize name, roll number, and branch, and a `display_details` method to print them. An example usage creates a `student1` object and calls its `display_details` method. To the right, a terminal window shows the code being run in a PowerShell environment (PS) on Windows. The output shows the creation of a `student1` object with name "Alice", roll number 101, and branch "Computer Science".

```
Assignment_6.3.py
Assignment_6.3.py > ...
1  "Task 1: Classes (Student Class)"
2  # Generate a Python Student class with attributes name, roll number, and
3  # branch, and a method to display student details
4  class Student:
5      def __init__(self, name, roll_number, branch):
6          self.name = name
7          self.roll_number = roll_number
8          self.branch = branch
9
10     def display_details(self):
11         print(f"Name: {self.name}")
12         print(f"Roll Number: {self.roll_number}")
13         print(f"Branch: {self.branch}")
14
15 # Example usage
16 student1 = Student("Alice", "101", "Computer Science")
17 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:

The screenshot shows a code editor interface with two panes. The left pane displays the Python file `Assignment_6.3.py` containing the following code:

```
File Edit Selection View Go Run Terminal Help ← →
Assignment_6.3.py ×
Assignment_6.3.py > ...
17
18 "Task 2: Loops (Multiples of a Number)"
19 # Generate a Python function to print the first 10 multiples of a given
20 # number using a For Loop
21 def print_multiples(number):
22     for i in range(1, 11):
23         multiple = number * i
24         print(f"{number} x {i} = {multiple}")
25 # Example usage:
26 print_multiples(5)
27
```

The right pane shows the terminal output of running the script:

```
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 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:

Assignment\_6.3.py

```
18 """ Task 2: Loops (Multiples of a Number) """
19 # Generate a Python function to print the first 10 multiples of a given
20 # number using a for loop
21 def print_multiples(number):
22     for i in range(1, 11):
23         multiple = number * i
24         print(f"{number} x {i} = {multiple}")
25 # Example usage
26 print_multiples(5)
27
28 "(Alternative loop)"
29 # Generate the same program using a while loop
30
31 def print_multiples_while(number):
32     i = 1
33     while i <= 10:
34         multiple = number * i
35         print(f"[number] x {i} = {multiple}")
36         i += 1
37 # Example usage
38 print_multiples_while(5)
```

Code

```
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
```

## 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:

The screenshot shows a dark-themed instance of Visual Studio Code. On the left, the Explorer sidebar displays two files: "Assignment\_6.3.py" and "Assignment\_6.3.py ...". The main editor area contains the following Python code:

```
File Edit Selection View Go Run Terminal Help ← → Assignment_6.3.py x Assignment_6.3.py ... 39 40 "Task 3: Conditional Statements (Age Classification)" 41 # Generate Python code using if-elif-else to classify age into child, 42 # teenager, adult, and senior. 43 def classify_age(age): 44     if age < 13: 45         return "Child" 46     elif 13 <= age < 20: 47         return "Teenager" 48     elif 20 <= age < 60: 49         return "Adult" 50     else: 51         return "Senior" 52 # Example usage 53 age = 45 54 classification = classify_age(age) 55 print(f"Age: {age}, Classification: {classification}")
```

To the right of the editor is a terminal window titled "Code" with the following output:

```
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>
```

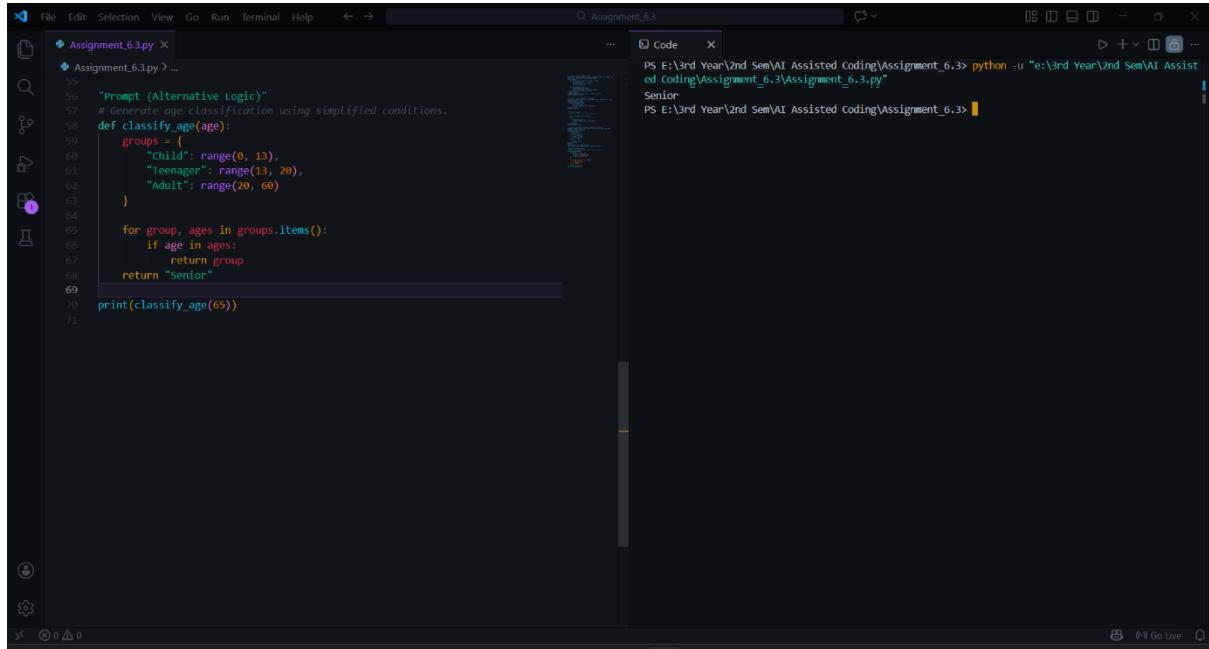
### **Explanation:**

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:**



A screenshot of a code editor showing a Python script named `Assignment_6.3.py`. The code defines a function `classify_age` that takes an age as input and returns a classification. It uses a dictionary to map age ranges to group names. The code is as follows:

```
File Edit Selection View Go Run Terminal Help ← →
Assignment_6.3.py ×
Assignment_6.3.py > ...
55
56 "Prompt (Alternative Logic)"
57 # Generate age classification using simplified conditions.
58 def classify_age(age):
59     groups = {
60         "Child": range(0, 13),
61         "Teenager": range(13, 20),
62         "Adult": range(20, 60)
63     }
64
65     for group, ages in groups.items():
66         if age in ages:
67             return group
68     return "Senior"
69
70 print(classify_age(65))
71
```

The output window shows the command `python -u "e:\3rd Year\2nd Sem\AI Assisted coding\Assignment_6.3\Assignment_6.3.py"` and the result `Senior`.

### **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:**

A screenshot of a code editor window titled "Assignment\_6.3". The left pane shows the file "Assignment\_6.3.py" with the following code:

```
Assignment_6.3.py
Assignment_6.3.py > ...
72 "Task 4: For and While Loops (Sum of First n Numbers)"
73 # Generate a Python function to calculate the sum of
74 # first n natural numbers using a for loop.
75 def sum_of_n_numbers(n):
76     total = 0
77     for i in range(1, n + 1):
78         total += i
79     return total
80 # Example usage
81 print(sum_of_n_numbers(5))
```

The right pane shows the terminal output:

```
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"
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:

A screenshot of a code editor window titled "Assignment\_6.3". The left pane shows the file "Assignment\_6.3.py" with the following code:

```
Assignment_6.3.py
Assignment_6.3.py > ...
83 "Prompt (Alternative Loop):"
84 # Generate the same functionality using a while loop.
85 def sum_of_n_numbers_while(n):
86     total = 0
87     i = 1
88     while i <= n:
89         total += i
90         i += 1
91     return total
92 # Example usage
93 print(sum_of_n_numbers_while(10))
```

The right pane shows the terminal output:

```
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 Assisted 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:

```
Assignment_6.3.py
Assignment_6.3.py > ...
96 "Task 5: Classes (Bank Account Class)"
97 # Generate a Python Bank Account class with methods to deposit, withdraw,
98 # and check balance.
99
100 class BankAccount:
101     def __init__(self, account_holder, initial_balance=0):
102         self.account_holder = account_holder
103         self.balance = initial_balance
104
105     def deposit(self, amount):
106         if amount > 0:
107             self.balance += amount
108             print(f"Deposited: {amount}. New Balance: {self.balance}")
109         else:
110             print("Deposit amount must be positive.")
111
112     def withdraw(self, amount):
113         if 0 < amount <= self.balance:
114             self.balance -= amount
115             print(f"Withdrew: {amount}. New Balance: {self.balance}")
116         else:
117             print("Insufficient balance or invalid withdrawal amount.")
118
119     def check_balance(self):
120         print(f"Account Holder: {self.account_holder}, Balance: {self.
121             balance}")
122
123 # Example usage
124 account = BankAccount("John Doe", 1000)
125 account.check_balance()
126 account.deposit(500)
127 account.withdraw(200)
128 account.check_balance()

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"
Account Holder: John Doe, Balance: 1000
Deposited: 500, New Balance: 1500
Withdraw: 200, New Balance: 1300
Account Holder: John Doe, Balance: 1300
```

### 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.