

## ASSIGNMENT- 6.3

**Name:** V. Trinayani

**HT.No:** 2303A51264

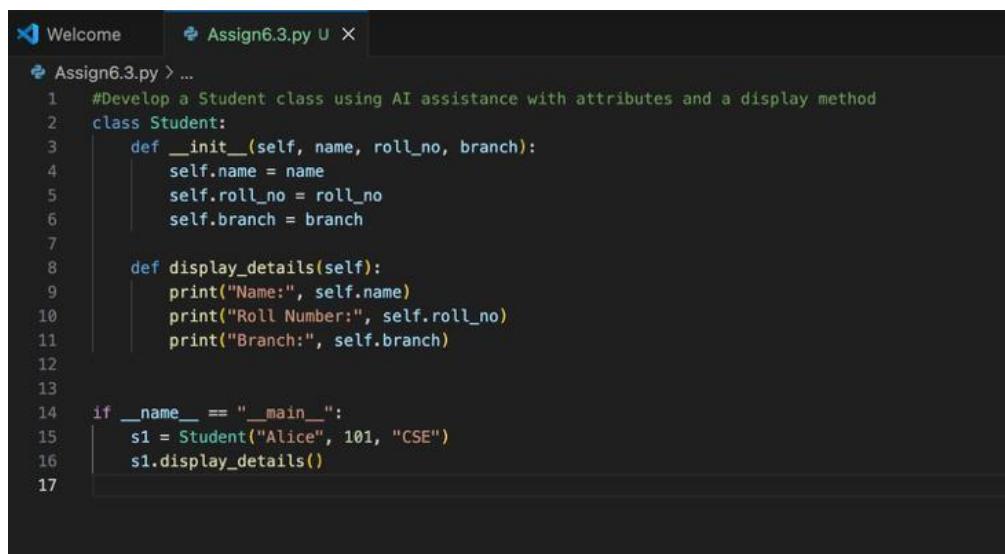
**Batch:** 19

### **Task 1: Classes – Student Class**

Develop a Student class using AI assistance with attributes and a display method

**Prompt: #Generate a Python Student class with name, roll number, and branch. Include a method to display student details..**

### **Code:**



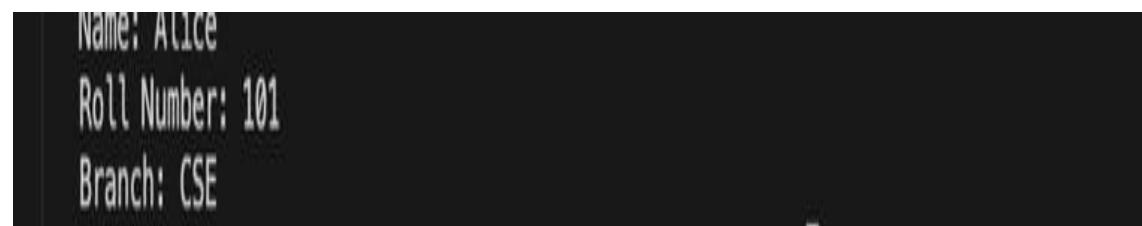
The screenshot shows a code editor window with two tabs: "Welcome" and "Assign6.3.py". The "Assign6.3.py" tab is active and displays the following Python code:

```
#Develop a Student class using AI assistance with attributes and a display method
class Student:
    def __init__(self, name, roll_no, branch):
        self.name = name
        self.roll_no = roll_no
        self.branch = branch

    def display_details(self):
        print("Name:", self.name)
        print("Roll Number:", self.roll_no)
        print("Branch:", self.branch)

if __name__ == "__main__":
    s1 = Student("Alice", 101, "CSE")
    s1.display_details()
```

### **Result:**



The screenshot shows a terminal window displaying the output of the Python code. The output consists of three lines of text: "Name: Alice", "Roll Number: 101", and "Branch: CSE".

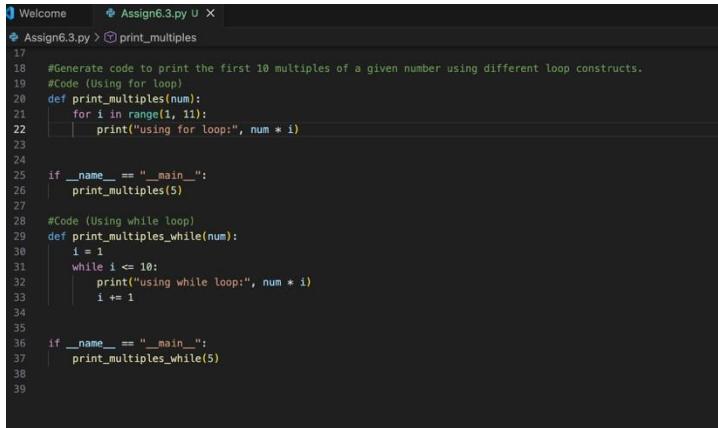
### **Observation:**

The AI-generated class structure is clear and logically organized. The constructor correctly initializes attributes, and the display method outputs student details in a readable format. The code is simple, correct, and suitable for beginner-level object-oriented programming.

**Task 2:** Loops – Multiples of a Number. Generate code to print the first 10 multiples of a given number using different loop constructs.

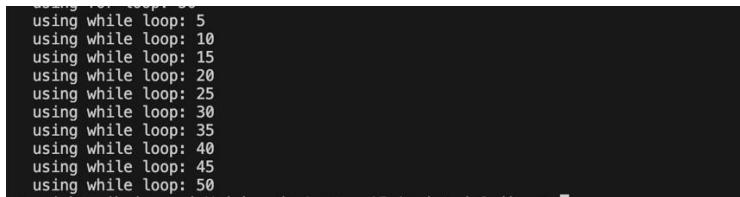
**Prompt:** #Generate Python code to print the first 10 multiples of a number using a loop.

**Code:**

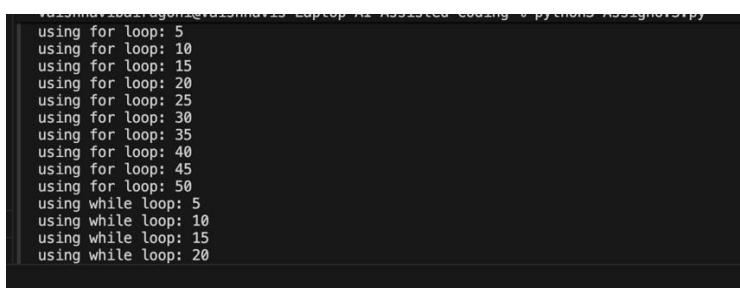


```
Welcome   Assign6.3.py X
Assign6.3.py > print_multiples
17
18     #Generate code to print the first 10 multiples of a given number using different loop constructs.
19     #Code (Using for loop)
20     def print_multiples(num):
21         for i in range(1, 11):
22             print("using for loop:", num * i)
23
24
25     if __name__ == "__main__":
26         print_multiples(5)
27
28     #Code (Using while loop)
29     def print_multiples_while(num):
30         i = 1
31         while i <= 10:
32             print("using while loop:", num * i)
33             i += 1
34
35
36     if __name__ == "__main__":
37         print_multiples_while(5)
38
39
```

**Result:**



```
using while loop: 5
using while loop: 10
using while loop: 15
using while loop: 20
using while loop: 25
using while loop: 30
using while loop: 35
using while loop: 40
using while loop: 45
using while loop: 50
```



```
using for loop: 5
using for loop: 10
using for loop: 15
using for loop: 20
using for loop: 25
using for loop: 30
using for loop: 35
using for loop: 40
using for loop: 45
using for loop: 50
using while loop: 5
using while loop: 10
using while loop: 15
using while loop: 20
```

**Observation:**

Both loop implementations correctly generate the required output. The for-loop version is more concise and readable, while the while-loop version provides better insight into loop control and iteration. AI suggestions for both approaches are correct and efficient.

**Task 3:** Conditional Statements – Age Classification. Classify a person's age into categories using conditional statements.

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

**Code:**

```
# Assign6.3.py U 
# Assign6.3.py > ...
40  #Classify a person's age into categories using conditional statements.
41  #Code (if-elif-else)
42  def classify_age(age):
43      if age < 13:
44          return "Child"
45      elif age < 20:
46          return "Teenager"
47      elif age < 60:
48          return "Adult"
49      else:
50          return "Senior"
51
52
53  if __name__ == "__main__":
54      print(classify_age(25))
55
56  #Code (Simplified logic using dictionary)
57  def classify_age_simple(age):
58      if age < 13:
59          return "Child"
60      if age < 20:
61          return "Teenager"
62      if age < 60:
63          return "Adult"
64      return "Senior"
65
```

**Result:**

```
using for loop: 45
using for loop: 50
using while loop: 5
using while loop: 10
using while loop: 15
using while loop: 20
using while loop: 25
using while loop: 30
using while loop: 35
using while loop: 40
using while loop: 45
using while loop: 50
Adult
```

**Observation:**

The AI-generated conditions correctly classify age groups. The if-elif-else structure is clear and readable, while the simplified version reduces nesting and improves clarity. Both approaches are logically sound.

**Task 4:** For and While Loops – Sum of First n Numbers. Calculate the sum of the first n natural numbers using different approaches.

**Prompt:** #Generate Python code to find the sum of the first n natural numbers using loops.

## Code:

```
Assign6.3.py U X
Assign6.3.py > ...
65
66 #Task-4:Calculate the sum of the first n natural numbers using different approaches
67 #Code (for loop)
68 def sum_to_n(n):
69     total = 0
70     for i in range(1, n + 1):
71         total += i
72     return total
73
74
75 if __name__ == "__main__":
76     print(sum_to_n(10))
77
78
79 #Code (while loop)
80 def sum_to_n_while(n):
81     total = 0
82     i = 1
83     while i <= n:
84         total += i
85         i += 1
86     return total
87
```

## Result:

55

## Observation

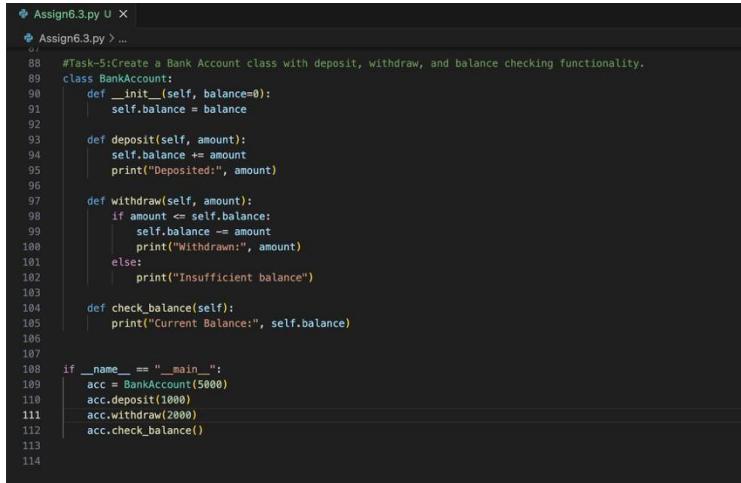
Both loop-based solutions produce the correct result. The for-loop version is more concise, while the while-loop version offers explicit control over iteration. AI-generated logic is correct and easy to understand.

## Task 5: Classes – Bank Account Class

Create a Bank Account class with deposit, withdraw, and balance checking functionality.

**Prompt: #Generate a Python Bank Account class with deposit, withdraw, and check balance methods.**

**Code:**



```
Assign6.3.py U ...
Assign6.3.py > ...
88 #Task-5:Create a Bank Account class with deposit, withdraw, and balance checking functionality.
89 class BankAccount:
90     def __init__(self, balance=0):
91         self.balance = balance
92
93     def deposit(self, amount):
94         self.balance += amount
95         print("Deposited:", amount)
96
97     def withdraw(self, amount):
98         if amount <= self.balance:
99             self.balance -= amount
100            print("Withdrawn:", amount)
101        else:
102            print("Insufficient balance")
103
104     def check_balance(self):
105         print("Current Balance:", self.balance)
106
107
108 if __name__ == "__main__":
109     acc = BankAccount(5000)
110     acc.deposit(1000)
111     acc.withdraw(2000)
112     acc.check_balance()
113
114
```

**Result:**

```
using while loop: 15
using while loop: 20
using while loop: 25
using while loop: 30
using while loop: 35
using while loop: 40
using while loop: 45
using while loop: 50
Adult
55
Deposited: 1000
Withdrawn: 2000
Current Balance: 4000
```

**Observation:**

The AI-generated class structure is well organized and logically correct. Methods perform expected operations, and balance updates are accurate. The code is readable, maintainable, and suitable for a basic banking application.