

# ASSIGNMENT - 6.3

**Syed Althaf**

**2303A52496**

**Batch-50**

## TASK-01:

### Question:

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:

Create a Python Student class with attributes `name`, `roll_number`, and `branch`. Add a constructor (`__init__`) and a method `display_details()` to print student information.

### Generated 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("Student Details:")
    print("Name:", self.name)
    print("Roll Number:", self.roll_number)
    print("Branch:", self.branch)
```

# Example usage

```
student1 = Student("Syed Althaf", "CSE2023A01", "CSE-AIML")
student1.display_details()
```

## Output:

```
syeda@DESKTOP-JPPV585 MINGW64 ~/OneDrive/Desktop/AI_Assistant_Coding (master)
$ C:/Users/syeda/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/syeda/OneDrive/Desktop/AI_Assistant_Coding/Lab_6.3.py"
Student Details:
Name: Syed Althaf
Roll Number: CSE2023A01
Branch: CSE-AIML
```

## Explanation:

This code defines a Student class that stores a student's name, roll number, and branch.

The `__init__` method sets these values when a student object is created.

The `display_details()` method prints the student's information, and the example shows how to

create and display one student's details.

## TASK-02:

### Question:

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 (2.1):

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

### Generated Code:

```
def print_multiples(number):
    for i in range(1, 11):
        print(f"{number} x {i} = {number * i}")
```

# Example usage

```
print_multiples(5)
```

## Output (2.1):

```
$ C:/Users/syeda/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/syeda/OneDrive/Desktop/AI Assistant_Coding/Lab_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

syeda@DESKTOP-JPPVS85 MINGW64 ~/OneDrive/Desktop/AI Assistant_Coding (master)
```

## Explanation:

This function prints the first 10 multiples of a given number.

It uses a for loop from 1 to 10 and multiplies the number with each value.

Calling print\_multiples(5) displays multiples of 5 from 5 to 50.

## Prompt (2.2):

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

## Generated Code:

```
def print_multiples(number):
    i = 1
    while i <= 10:
        print(f"{number} x {i} = {number * i}")
        i += 1
```

```
# Example usage
print_multiples(7)
```

## Output (2.2):

```
syeda@DESKTOP-JPPVS85 MINGW64 ~/OneDrive/Desktop/AI Assistant_Coding (master)
$ C:/Users/syeda/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/syeda/OneDrive/Desktop/AI Assistant_Coding/Lab_6.3.py"
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

## Explanation:

This function prints the first 10 multiples of a given number using a while loop. It starts with  $i = 1$  and keeps multiplying the number until  $i$  reaches 10. Each multiple is printed, and  $i$  is increased by 1 in every loop iteration.

## TASK-03:

### Question:

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 (3.1):

Write a Python function that classifies a person's age into groups such as child, teenager, adult, and senior using if-elif-else statements.

### Generated Code:

```
def classify_age(age):
    if age < 0:
        return "Invalid age"
    elif age <= 12:
        return "Child"
    elif age <= 19:
        return "Teenager"
    elif age <= 59:
        return "Adult"
    else:
        return "Senior"
```

```
# Example usage
print(classify_age(10))
print(classify_age(16))
print(classify_age(30))
print(classify_age(65))
```

### Output (3.1):

```
syeda@DESKTOP-JPPVS85 MINGW64 ~/OneDrive/Desktop/AI Assistant_Coding (master)
$ C:/Users/syeda/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/syeda/OneDrive/Desktop/AI Assistant_Coding/b_6.3.py"
Child
Teenager
Adult
Senior
syeda@DESKTOP-JPPVS85 MINGW64 ~/OneDrive/Desktop/AI Assistant_Coding (master)
$
```

### Explanation:

This function checks a person's age and returns a category like Child, Teenager, Adult, or Senior.

It uses if-elif-else conditions to compare the age with different ranges.

The example tests multiple ages and prints their corresponding categories.

### Prompt (3.2):

Implement the same age classification using an alternative approach such as simplified

conditional statements or dictionary-based logic.

Explain how this approach works and how it differs from the if-elif-else method.

### Generated Code:

```
def classify_age(age):
    age_groups = {
        "Child": range(0, 13),
        "Teen": range(13, 20),
        "Adult": range(20, 60),
        "Senior": range(60, 150)
    }

    for category, age_range in age_groups.items():
        if age in age_range:
```

```
return category
```

```
# Example
```

```
print(classify_age(25)) # Adult
```

```
print(classify_age(10)) # Child
```

## Output (3.2):

```
syeda@DESKTOP-JPPVS85 MINGW64 ~/OneDrive/Desktop/AI Assistant_Coding (master)
$ C:/Users/syeda/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/syeda/OneDrive/Desktop/AI Assistant_Coding
b_6.3.py"
Adult
Child

syeda@DESKTOP-JPPVS85 MINGW64 ~/OneDrive/Desktop/AI Assistant_Coding (master)
$
```

## Explanation:

This function classifies a person's age using a **dictionary of age ranges** instead of multiple if-elif statements.

It checks which age range the given value falls into and returns the matching category. This approach is flexible and easy to update because age ranges and categories are stored separately from the logic.

## TASK-04:

### Question:

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 (4.1):

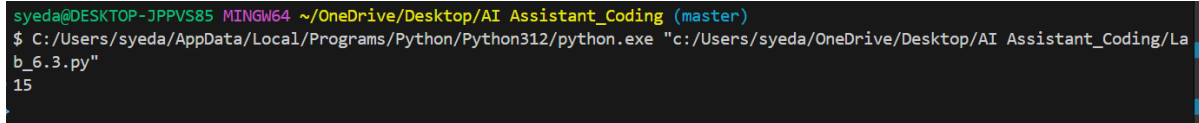
Write a Python function `sum_to_n(n)` that calculates the sum of the first n natural numbers using a for loop.

### Generated Code:

```
def sum_to_n(n):
    total = 0
    for i in range(1, n + 1):
```

```
    total += i
return total
```

## Output (4.1):



```
syeda@DESKTOP-JPPVS85 MINGW64 ~/OneDrive/Desktop/AI Assistant_Coding (master)
$ C:/Users/syeda/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/syeda/OneDrive/Desktop/AI Assistant_Coding/Lab_6.3.py"
15
```

## Explanation:

This function calculates the sum of the first n natural numbers.

If n is less than 1, it returns 0 to avoid invalid input.

Otherwise, it uses a for loop to add numbers from 1 to n and prints the final sum.

## Prompt (4.2):

Rewrite the sum\_to\_n(n) function using an alternative approach such as a while loop or a

mathematical formula.

Explain how this approach differs from the for loop implementation.

## Generated Code:

```
def sum_to_n(n):
```

```
    total = 0
```

```
    i = 1
```

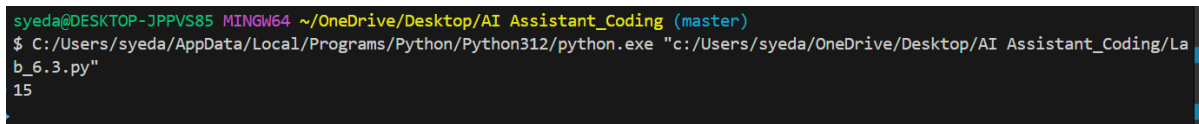
```
    while i <= n:
```

```
        total += i
```

```
        i += 1
```

```
    return total
```

## Output (4.2):



```
syeda@DESKTOP-JPPVS85 MINGW64 ~/OneDrive/Desktop/AI Assistant_Coding (master)
$ C:/Users/syeda/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/syeda/OneDrive/Desktop/AI Assistant_Coding/Lab_6.3.py"
15
```

## Explanation:

This code finds the sum of the first n natural numbers using a mathematical formula.

If n is less than 1, it returns 0; otherwise, it directly calculates the sum using  $n(n + 1)/2$ .

Compared to the loop-based method, this approach is faster and more efficient because it runs in constant time  $O(1)$ .

## **TASK-05:**

### **Question:**

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:**

Demonstrate deposit and withdrawal operations and display the updated balance. Analyze the class structure and logic, add meaningful comments, and explain how the code works.

### **Generated Code:**

```
class BankAccount:
```

```
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance
```

```
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited: ₹{amount}")
        else:
            print("Deposit amount must be positive.")
```

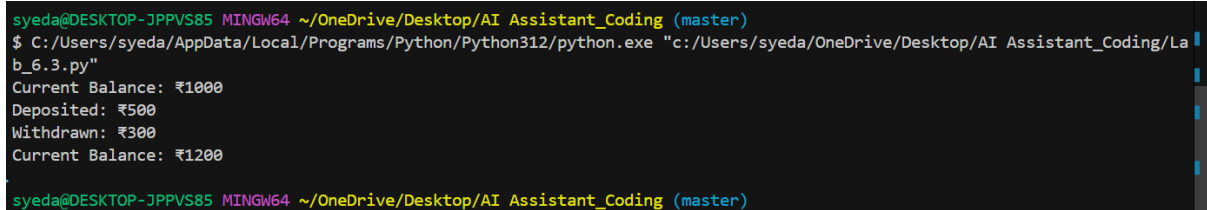
```
    def withdraw(self, amount):
        if amount <= 0:
            print("Withdrawal amount must be positive.")
        elif amount > self.balance:
            print("Insufficient balance!")
        else:
            self.balance -= amount
            print(f"Withdrawn: ₹{amount}")
```

```
    def display_balance(self):
```

```
print(f"Current Balance: ₹{self.balance}")
```

```
account = BankAccount("Syed Althaf", 1000)
account.display_balance()
account.deposit(500)
account.withdraw(300)
account.display_balance()
```

## Output:

A terminal window with a dark background and light-colored text. The prompt is 'syeda@DESKTOP-JPPVS85 MINGW64 ~/OneDrive/Desktop/AI Assistant\_Coding (master)'. The command executed is '\$ C:/Users/syeda/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/syeda/OneDrive/Desktop/AI Assistant\_Coding/La b\_6.3.py"'. The output shows: 'Current Balance: ₹1000', 'Deposited: ₹500', 'Withdrawn: ₹300', and 'Current Balance: ₹1200'. The prompt at the bottom is 'syeda@DESKTOP-JPPVS85 MINGW64 ~/OneDrive/Desktop/AI Assistant\_Coding (master)'.

```
syeda@DESKTOP-JPPVS85 MINGW64 ~/OneDrive/Desktop/AI Assistant_Coding (master)
$ C:/Users/syeda/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/syeda/OneDrive/Desktop/AI Assistant_Coding/La
b_6.3.py"
Current Balance: ₹1000
Deposited: ₹500
Withdrawn: ₹300
Current Balance: ₹1200
syeda@DESKTOP-JPPVS85 MINGW64 ~/OneDrive/Desktop/AI Assistant_Coding (master)
```

## Explanation:

This code defines a **BankAccount** class to manage basic banking operations. It stores the account holder's name and balance, and provides methods to deposit money, withdraw money (with balance checks), and view the current balance. The example shows creating an account, performing transactions, and printing the final balance.