

ASSIGNMENT 6.3

NAME: Karthikeya Uthuri

HALLTICKETNO: 2303A51306

BATCH: 05

TASK1:

Scenario

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 method `display_details()` to print the student information.

Create a sample object, call the method, and show the output.

Also give a short analysis of the code.

OUTPUT:

```
● 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"Student Name: {self.name}")  
        print(f"Roll Number: {self.roll_number}")  
        print(f"Branch: {self.branch}")  
  
# Create a student object  
student1 = Student("Alice Smith", "CS001", "Computer Science")  
  
# Display student details  
student1.display_details()  
... Student Name: Alice Smith  
Roll Number: CS001  
Branch: Computer Science
```

TASK 2:

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: Write a Python function that prints the **first 10 multiples** of a given number using a **for loop**. Explain the loop logic briefly

OUTPUT:

```
[2] ✓ os
    def print_multiples_for_loop(number):
        print(f"Multiples of {number} (using for loop):")
        for i in range(1, 11): # Loop 10 times, from 1 to 10
            print(number * i)

[3] ✓ os
    # Example usage:
    print_multiples_for_loop(7)

    ...
    Multiples of 7 (using for loop):
    7
    14
    21
    28
    35
    42
    49
    56
    63
    70
```

```
[0] ✓ os
    def print_multiples_while_loop(number):
        print(f"Multiples of {number} (using while loop):")
        count = 1
        while count <= 10:
            print(number * count)
            count += 1

[1] ✓ os
    # Example usage:
    print_multiples_while_loop(7)

    ...
    Multiples of 7 (using while loop):
    7
    14
    21
    28
    35
    42
    49
    56
    63
    70
```

Task Description #3: Conditional Statements (Age Classification)

Scenario

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:

Write a Python program that classifies a person based on age using **if-elif-else** statements (child, teenager, adult, senior)

OUTPUT:

```
[10] 08 def classify_age_dict_based(age):
    if age < 0:
        return "Invalid Age"

    # Define age ranges and their classifications using a list of tuples for ordered checking
    age_ranges = [
        (12, "Child"),
        (19, "Teenager"),
        (64, "Adult")
    ]

    for limit, classification in age_ranges:
        if age <= limit:
            return classification

    return "Senior" # If age is greater than all defined limits
```

```
1 08 # Example usage:
ages = [-5, 5, 15, 30, 70, 12, 19, 64, 65]

print("Age Classification using dictionary-based logic:")
for age in ages:
    print(f"Age: {age}, Classification: {classify_age_dict_based(age)}")

*** Age Classification using dictionary-based logic:
Age: -5, Classification: Invalid Age
Age: 5, Classification: Child
Age: 15, Classification: Teenager
Age: 30, Classification: Adult
Age: 70, Classification: Senior
Age: 12, Classification: Child
Age: 19, Classification: Teenager
Age: 64, Classification: Adult
Age: 65, Classification: Senior
```

```

[8] ✓ os def classify_age_if_elif_else(age):
    if age < 0:
        return "Invalid Age"
    elif age <= 12:
        return "Child"
    elif age < 19:
        return "Teenager"
    elif age <= 64:
        return "Adult"
    else:
        return "Senior"

[9] ✓ os # Example usage:
ages = [5, 5, 15, 30, 70, 12, 19, 64, 65]

print("Age Classification using if-elif-else:")
for age in ages:
    print(f"Age: {age}, Classification: {classify_age_if_elif_else(age)}")

Age Classification using if-elif-else:
Age: 5, Classification: Invalid Age
Age: 5, Classification: Child
Age: 15, Classification: Teenager
Age: 30, Classification: Adult
Age: 70, Classification: Senior
Age: 12, Classification: Child
Age: 19, Classification: Teenager
Age: 64, Classification: Adult
Age: 65, Classification: Senior

```

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

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.

Formula

PROMPT:

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

Display the result and explain the code briefly

OUTPUT:

```

def sum_to_n_while_loop(n):
    if n < 1:
        return "Input must be a positive integer"
    total_sum = 0
    count = 1
    while count <= n:
        total_sum += count
        count += 1
    return total_sum

def sum_to_n_formula(n):
    if n < 1:
        return "Input must be a positive integer"
    # Gauss's formula for the sum of the first n natural numbers
    return n * (n + 1) // 2

```

```
# Example usage for while loop and formula:
number = 5
result_while = sum_to_n_while_loop(number)
print(f"The sum of the first {number} natural numbers (using while loop) is: {result_while}")
result_formula = sum_to_n_formula(number)
print(f"The sum of the first {number} natural numbers (using formula) is: {result_formula}")

number = 10
result_while = sum_to_n_while_loop(number)
print(f"The sum of the first {number} natural numbers (using while loop) is: {result_while}")
result_formula = sum_to_n_formula(number)
print(f"The sum of the first {number} natural numbers (using formula) is: {result_formula}")

number = 0
result_while = sum_to_n_while_loop(number)
print(f"The sum of the first {number} natural numbers (using while loop) is: {result_while}")
result_formula = sum_to_n_formula(number)
print(f"The sum of the first {number} natural numbers (using formula) is: {result_formula}")

...
The sum of the first 5 natural numbers (using while loop) is: 15
The sum of the first 5 natural numbers (using formula) is: 15
The sum of the first 10 natural numbers (using while loop) is: 55
The sum of the first 10 natural numbers (using formula) is: 55
The sum of the first 0 natural numbers (using while loop) is: Input must be a positive integer
The sum of the first 0 natural numbers (using formula) is: Input must be a positive integer
```

Task Description #5: Classes (Bank Account Class)

Scenario

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:

Create a Python **BankAccount** class with attributes like `account_number` and `balance`.
Add methods `deposit()`, `withdraw()`, and `check_balance()`.
Show sample object creation and method calls

OUTPUT:

```
[17] ✓ 0s
  class BankAccount:
      def __init__(self, account_holder, initial_balance=0):
          # Constructor to initialize a new bank account
          # account_holder: Name of the account owner
          # initial_balance: Starting balance, defaults to 0
          self.account_holder = account_holder
          self.balance = initial_balance
          print(f"Account for {self.account_holder} created with initial balance: ${self.balance:.2f}")

      def deposit(self, amount):
          # Method to deposit money into the account
          # amount: The amount to be deposited
          if amount > 0:
              self.balance += amount
              print(f"Deposited: ${amount:.2f}. New balance: ${self.balance:.2f}")
          else:
              print("Deposit amount must be positive.")

      def withdraw(self, amount):
          # Method to withdraw money from the account
          # amount: The amount to be withdrawn
          if amount > 0:
              if self.balance >= amount:
                  self.balance -= amount
                  print(f"Withdrew: ${amount:.2f}. New balance: ${self.balance:.2f}")
              else:
                  print("Insufficient funds.")
          else:
              print("Withdrawal amount must be positive.")

  ✓ 10:29 AM Python 3
```

```
[17] ✓ 0s
  def check_balance(self):
      # Method to display the current account balance
      print(f"Current balance for {self.account_holder}: ${self.balance:.2f}")

  # ---- Example Usage ---
  #print("\n--- Creating and managing Bank Accounts ---")

  # Create a new bank account
  account1 = BankAccount("John Doe", 1000)

  # Check initial balance
  account1.check_balance()

  # Perform a deposit
  account1.deposit(500)

  # Perform a withdrawal
  account1.withdraw(200)

  # Try to withdraw more than available
  account1.withdraw(1500)

  # Try invalid deposit/withdrawal amounts
  account1.deposit(-100)
  account1.withdraw(0)

  # Check final balance
  account1.check_balance()
  print("\n--- Another Account Example ---")

  ✓ 10:29 AM Python 3
```

```
[17] ✓ 0s
  account1.deposit(-100)
  account1.withdraw(0)

  # Check final balance
  account1.check_balance()

  print("\n--- Another Account Example ---")
  account2 = BankAccount("Jane Smith") # No initial balance specified, defaults to 0
  account2.deposit(250.75)
  account2.check_balance()

  ...
  --- Creating and managing Bank Accounts ---
  Account for John Doe created with initial balance: $1000.00
  Current balance for John Doe: $1000.00
  Deposited: $500.00. New balance: $1500.00
  Withdraw: $200.00. New balance: $1300.00
  Insufficient funds.
  Deposit amount must be positive.
  Withdrawal amount must be positive.
  Current balance for John Doe: $1300.00

  --- Another Account Example ---
  Account for Jane Smith created with initial balance: $0.00
  Deposited: $250.75. New balance: $250.75
  Current balance for Jane Smith: $250.75

  ✓ 10:29 AM Python 3
```

