

## LAB ASSIGNMENT-6.3

### <AI ASSISTED CODING>

#### ➤ **Task Description#1**(Classes)

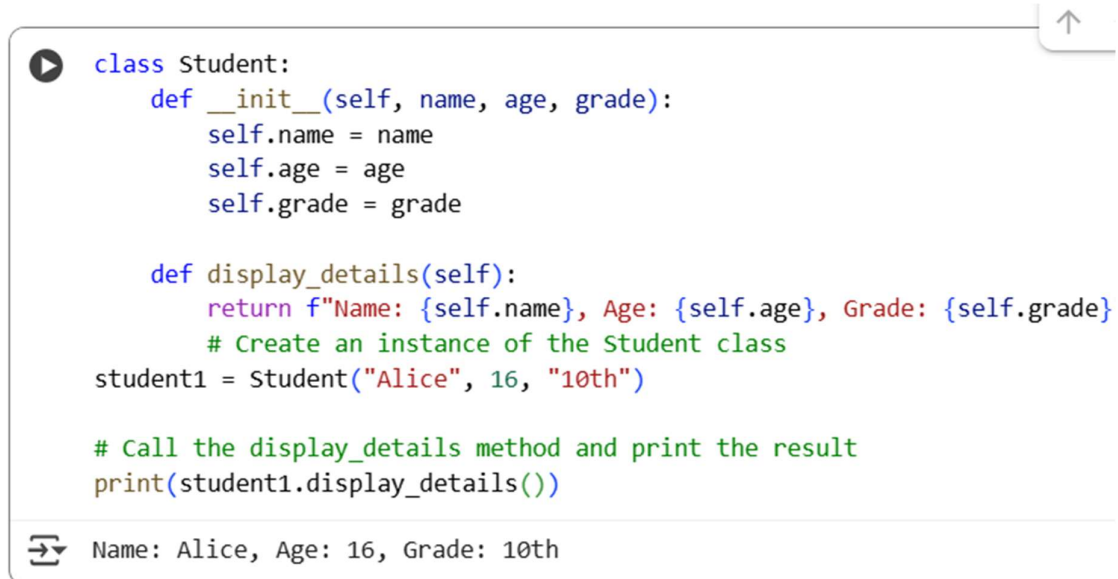
- Use AI to complete a Student class with attributes and a method.
- Check output
- Analyze the code generated by AI tool

#### ➤ **Expected output :**

- Class with constructor and display\_details() method.

#### ➤ **Prompt :**

Create a Python class named Student with attributes and a method .Class with constructor and display\_details() method give a code.



```
class Student:
    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade

    def display_details(self):
        return f"Name: {self.name}, Age: {self.age}, Grade: {self.grade}"
# Create an instance of the Student class
student1 = Student("Alice", 16, "10th")

# Call the display_details method and print the result
print(student1.display_details())
```

➤ Name: Alice, Age: 16, Grade: 10th

#### ➤ **Analyze :**

This code defines a blueprint for creating student objects and then creates one such object named student1 and displays its details using the display\_details method.

#### ➤ **Observation :**

This code is that it defines a way to represent a student with their name, age, and grade, and provides a method to easily get these details in a readable format. It then shows how to create a specific student and print their information.

#### ➤ **Task Description#2** (Loops)

- Prompt AI to complete a function that prints the first 10

multiples of a number using a loop.

- Analyze the generated code
- Ask AI to generate code using other controlled looping

➤ **Expected output :**

- Correct loop-based implementation.

➤ **Prompt :**

Write a Python function that takes a number as input and prints its first 10 multiples using a loop.

```
def print_multiples(number):  
    """  
    Prints the first 10 multiples of a given number.  
  
    Args:  
        number: The number for which to print multiples.  
    """  
    for i in range(1, 11):  
        print(f"{number} x {i} = {number * i}")  
  
# Example usage:  
print_multiples(5)
```

```
➡ 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
```

➤ **Analyze :**

This function takes a number and then prints out the multiplication table for that number from 1 times the number up to 10 times the number.

- **Generate code using other controlled looping**

```

def print_multiples_while(number):
    """
    Prints the first 10 multiples of a given number using a while loop.

    Args:
        number: The number for which to print multiples.
    """
    i = 1
    while i <= 10:
        print(f"{number} x {i} = {number * i}")
        i += 1

# Example usage:
print_multiples_while(7)

```

```

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

```

### ➤ Observation :

This code (cell 82a00610) is that it uses a while loop to achieve the same result as the previous for loop example – printing the first 10 multiples of a number. It requires manual initialization of a counter ( $i = 1$ ) and manual incrementing ( $i += 1$ ) within the loop.

### ➤ Task Description#3 (Conditional Statements)

- Ask AI to write nested if-elif-else conditionals to classify age groups.
- Analyze the generated code
- Ask AI to generate code using other conditional statements.

### ➤ Expected output :

- Age classification function with appropriate conditions and with explanation.

### ➤ Prompt :

Write a Python program using nested if-elif-else statements to classify a person's age into groups like "Child", "Teen", "Adult", or "Senior" based on their age.

```
def classify_age_conditional_logic(age):  
    if age < 0:  
        return "Invalid age"  
    else:  
        if age >= 0 and age < 13:  
            return "Child"  
        if age >= 13 and age < 20:  
            return "Teen"  
        if age >= 20 and age < 65:  
            return "Adult"  
        if age >= 65:  
            return "Senior"  
        print(classify_age(8))  
    print(classify_age(17))  
    print(classify_age(35))  
    print(classify_age(70))  
    print(classify_age(-5))
```

```
⇒ Teen  
Adult  
Senior  
Invalid age
```

➤ **Analyze :**

This function first checks for an invalid negative age. If the age is valid (0 or greater), it then proceeds to check which age group the person falls into based on a series of ranges using nested conditional statements.

➤ **Generate code using other conditional statements.**

```

def classify_age_conditional_logic(age):
    if age < 0:
        return "Invalid age"
    else:
        if age >= 0 and age < 13:
            return "Child"
        if age >= 13 and age < 20:
            return "Teen"
        if age >= 20 and age < 65:
            return "Adult"
        if age >= 65:
            return "Senior"
    print(classify_age(8))
print(classify_age(17))
print(classify_age(35))
print(classify_age(70))
print(classify_age(-5))

```

```

Teen
Adult
Senior
Invalid age

```

#### ➤ **Observation :**

This code (cell 8c1facff) is that it uses a series of separate if statements with and to check for age ranges, rather than the elif chain used in the previous example (cell 267fa196). While it achieves the same outcome for valid ages, the structure is different and relies on each if condition being evaluated independently if the initial else block is entered.

#### ➤ **Task Description#4** (For and While loops)

- Generate a sum\_to\_n() function to calculate sum of first n numbers
- Analyze the generated code
- Get suggestions from AI with other controlled looping.

#### ➤ **Expected output :**

- Python code with explanation.

#### ➤ **Prompt :**

Write a Python function called sum\_to\_n(n) that returns the sum of the first n natural numbers using a loop.

```

▶ def sum_to_n(n):
    total_sum = 0
    # Natural numbers start from 1
    for i in range(1, n + 1):
        total_sum += i
    return total_sum

# Example usage:
print(sum_to_n(10))
print(sum_to_n(5)) |

```

```

⇒ 55
   15

```

#### ➤ **Analyze :**

This function takes a number  $n$ , adds up all the whole numbers from 1 to  $n$ , and gives you the final sum.

#### ➤ **Generate with other controlled looping.**

```

▶ def sum_to_n_while(n):
    total_sum = 0
    i = 1
    while i <= n:
        total_sum += i
        i += 1
    return total_sum

# Example usage:
print(sum_to_n_while(10)) # Sum of first 10 natural numbers
print(sum_to_n_while(5)) # Sum of first 5 natural numbers

```

```

⇒ 55
   15

```

#### ➤ **Obseravtion :**

This code (cell 70f920b4) is that it calculates the sum of the first  $n$  natural numbers using a while loop. Like the while loop for multiples, it requires explicit initialization and incrementing of a counter variable ( $i$ ).

#### ➤ **Task Description#5** (Class)

- Use AI to build a BankAccount class with deposit, withdraw, and balance

methods.

- Analyze the generated code
- Add comments and explain code.

➤ **Expected output :**

- Python code with explanation.

➤ **Prompt :**

Write a Python class called BankAccount with methods to deposit, withdraw, and check balance. The class should store the account balance and update it based on transactions.

```
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

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

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew: ${amount}. New balance: ${self.balance}")
        elif amount > self.balance:
            print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        print(f"Current balance: ${self.balance}")

# Example usage:
account = BankAccount(1000)
account.deposit(500)
account.withdraw(200)
account.check_balance()
account.withdraw(2000) # Example of insufficient funds
```

```
➤ Deposited: $500. New balance: $1500
➤ Withdrew: $200. New balance: $1300
➤ Current balance: $1300
➤ Insufficient funds.
```

➤ **Analyze :**

The end demonstrates how to create a BankAccount object, deposit money, withdraw money, and check the balance.

➤ **Add comments and explain code.**

```
class BankAccount:
    # Constructor to initialize the account with an optional initial balance
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

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

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

    # Method to check the current balance of the account
    def check_balance(self):
        print(f"Current balance: ${self.balance}")

# Example usage:
account = BankAccount(1000)
account.deposit(500)
```

It creates a **bank account** in Python that can:

- Start with some money
- Add money (deposit)
- Take out money (withdraw)
- Show how much money is left (check balance).

➤ **Obseravtion :**

This code provides a basic simulation of a bank account with functions to add money (deposit), take money out (withdraw), and see how much money is currently in the account (check balance).



