Assignment-6.3

Name:Y.Triveni

HT.NO:2303A51349

BTNO:20

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:

```python
#Task 1
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("\nStudent Details:")
        print("Name:", self.name)
        print("Roll Number:", self.roll_number)
        print("Branch:", self.branch)


# Taking input dynamically
name = input("Enter student name: ")
roll_number = int(input("Enter roll number: "))
branch = input("Enter branch: ")

# Creating object
student = Student(name, roll_number, branch)

# Displaying details
student.display_details()
```

Result:

```
Enter student name: manu
Enter roll number: 123
Enter branch: cse

Student Details:
Name: manu
Roll Number: 123
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:

```python
#Task 2
#For
def print_multiples_for(num):
    print("First 10 multiples using for loop:")
    for i in range(1, 11):
        print(num * i)
number = int(input("Enter a number: "))
print_multiples_for(number)
#while
def print_multiples_while(num):
    print("First 10 multiples using while loop:")
    i = 1
    while i <= 10:
        print(num * i)
        i += 1
number = int(input("Enter a number: "))
print_multiples_while(number)
```

Result:

```
Enter a number: 2
First 10 multiples using for loop:
2
4
6
8
10
12
14
16
18
20
Enter a number: 1
First 10 multiples using while loop:
1
2
3
4
5
6
7
8
9
10
```

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:

```
#Task 3

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"
age = int(input("Enter age: "))
print("Age Group:", classify_age(age))
#AI-Generated Alternative Using Simplified Conditional Logic
def classify_age_simple(age):
    if age < 0:
        return "Invalid age"
    if age <= 12:
        return "Child"
    if age <= 19:
        return "Teenager"
    if age <= 59:
        return "Adult"
    return "Senior"
age = int(input("Enter age: "))
print("Age Group:", classify_age_simple(age))
```

Result:

```
Enter age: 3
Age Group: Child
Enter age: 78
Age Group: Senior
```

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:

```python
#Task 4
def sum_to_n_for(n):
  sum=0
  for i in range(1,n+1):
    sum+=i
  return sum
number = int(input("Enter a number: "))
print("Sum of numbers up to", number, "using for loop:", sum_to_n_for(number))
def sum_to_n_while(n):
    sum=0
    i=1
    while i<=n:
      sum+=i
      i+=1
    return sum
number = int(input("Enter a number: "))
print("Sum of numbers up to", number, "using while loop:", sum_to_n_while(number))
```

Result:

```
Enter a number: 3
Sum of numbers up to 3 using for loop: 6
Enter a number: 4
Sum of numbers up to 4 using while loop: 10
```

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:

```
'''Generate a Python program that defines a BankAccount class with attributes for account holder name and balance.
Include methods deposit(), withdraw(), and check_balance().
The deposit method should allow only positive amounts.
The withdraw method should check for sufficient balance.
Add meaningful comments and demonstrate the class using user input."'''
class BankAccount:
    def __init__(self, account_holder_name, initial_balance=0):
        self.account_holder_name = account_holder_name
        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}")
        else:
            print("Insufficient funds or invalid withdrawal amount.")

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

# Example usage
account = BankAccount("John Doe", 100)
account.check_balance()
account.deposit(50)
account.withdraw(30)
account.check_balance()
account.withdraw(150)  # Attempt to withdraw more than the balance
account.deposit(-20)   # Attempt to deposit a negative amount
```

Result:

```
Account holder: John Doe
Current balance: $100
Deposited $50. New balance: $150
Withdrew $30. New balance: $120
Account holder: John Doe
Current balance: $120
Insufficient funds or invalid withdrawal amount.
Current balance: $120
Insufficient funds or invalid withdrawal amount.
Deposit amount must be positive.
```

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.