AI ASSISTED CODING

ASSIGNMENT 6.3

Name: Koudagani Sahithya

Ht.No: 2403A52063

Batch: 02

TASK 1

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

PROMPT:

Use AI to generate a Python Student class with attributes and one method, run it to check output, and analyze the code

CODE:

```python
class Student:
    """Represents a student with name, age, and grade."""

    def __init__(self, name, age, grade):
        """Initializes a new Student object."""
        self.name = name
        self.age = age
        self.grade = grade

    def display_info(self):
        """Displays the student's information."""
        print(f"Name: {self.name}, Age: {self.age}, Grade: {self.grade}")

# Example usage:
student1 = Student("Alice", 16, "A")
student1.display_info()
```

OUTPUT:

```
Enter student name: Alice
Enter student age: 16
Enter student grade: A
Name: Alice, Age: 16, Grade: A
```

OBSERVATION:

The code defines a Student class with attributes for name, age, and grade. It takes user input to create a student object, converting age to an integer. Finally, it uses a method to display the student's details, as shown in the output.

CONCLUSION:

we have successfully defined a Student class in Python, which can store a student's name, age, and grade. The code allows for interactive input from the user to create

a Student object and then displays the stored information using a dedicated method. This demonstrates the basic principles of object-oriented programming with classes, attributes, and methods.

TASK 2

• 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

PROMPT:

Write a function that prints the first 10 multiples of a given number using a loop. Check and analyze the generated code. Again generate the same program using a different looping construct (e.g., while instead of for).

CODE:

```python
def print_multiples_while(number):
  """Prints the first 10 multiples of a given number using a while loop."""
  print(f"Multiples of {number}:")
  count = 1
  while count <= 10:
    print(number * count)
    count += 1

# Example usage with user input:
num_to_print = int(input("Enter the number to find multiples of: "))
print_multiples_while(num_to_print)
```

OUTPUT:

```
Enter the number to find multiples of: 7
Multiples of 7:
7
14
21
28
35
42
49
56
63
70
```

OBSERVATION:

The code defines a print_multiples_while function using a while loop to print the first 10 multiples of a number. It takes user input for the number, converts it to an integer, and then calls the function to display the multiples.

CONCLUSION:

The code successfully implements a function using a while loop to find and print the first 10 multiples of a

number. By incorporating user input, the code becomes interactive, allowing the user to specify the number and see the results directly, demonstrating a practical application of while loops and user interaction in Python.

TASK 3

• 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

PROMPT:

Write a Python program with nested if-elif-else conditionals to classify age groups. Check and analyze the generated code. Then, rewrite the program using other conditional approaches (e.g., match-case or logical operators)

CODE:

```python
def classify_age_nested_if(age):
  """Classifies age into groups using nested if-elif-else."""
  if age < 0:
    print("Invalid age")
  else: # Nested conditional block starts here
    if age < 13:
      print("Child")
    elif age < 20:
      print("Teenager")
    elif age < 65:
      print("Adult")
    else:
      print("Senior")

# Example usage with user input:
age_input = int(input("Enter the age to classify: "))
classify_age_nested_if(age_input)
```

OUTPUT:

```
Enter the age to classify: 13
Teenager
```

OBSERVATION:

The code defines a function classify_age_nested_if that uses nested if-elif-else statements to categorize age. It takes user input for age, handles invalid negative input, and correctly classifies ages into Child, Teenager, Adult, or Senior groups.

CONCLUSION:

In conclusion, the code successfully demonstrates age group classification using nested if-elif-else statements and handles user input. It effectively categorizes ages based on defined ranges, illustrating a fundamental conditional control flow structure in Python.

TASK 4

- 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

PROMPT:

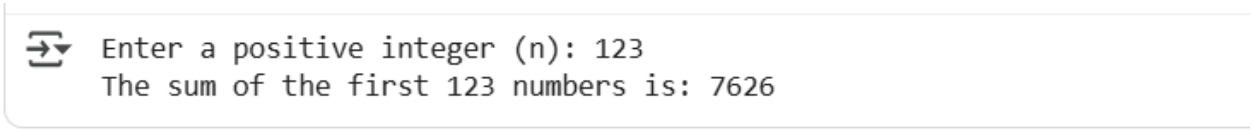
Generate a sum_to_n() function in Python to calculate the sum of the first n numbers using a loop? Then show how it can be written using a different looping construct.

CODE:

```
def sum_to_n_for(n):
  """Calculates the sum of the first n numbers using a for loop."""
  total_sum = 0
  for i in range(1, n + 1):
    total_sum += i
  return total_sum

# Example usage with user input:
num = int(input("Enter a positive integer (n): "))
result = sum_to_n_for(num)
print(f"The sum of the first {num} numbers is: {result}")
```

OUTPUT:



OBSERVATION:

The code defines a function sum_to_n_for using a for loop to calculate the sum of the first 'n' numbers. It takes user input for 'n', iterates from 1 to 'n', accumulates the sum, and returns the result, which is then displayed

CONCLUSION:

The code effectively calculates the sum of the first 'n' positive integers using a for loop and user input. It demonstrates a fundamental iterative approach to summation, providing a clear and functional solution to the task

TASK 5

• Use AI to build a BankAccount class with deposit, withdraw, and balance methods.

- Analyze the generated code
- Add comments and explain code

PROMPT:

Can you build a BankAccount class in Python with deposit, withdraw, and balance methods? After generating the code, analyze it, add comments, and explain how it works.

CODE:

```python
class BankAccount:
    """Represents a simple bank account."""

    def __init__(self, account_holder, initial_balance=0):
        """Initializes a new bank account."""
        self.account_holder = account_holder
        self.balance = initial_balance
        print(f"Account created for {self.account_holder} with initial balance ${self.balance:.2f}")

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

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

    def get_balance(self):
        """Returns the current account balance."""
        return self.balance
```

```python
# Example usage with user input:
account_name = input("Enter account holder name: ")
initial_deposit = float(input("Enter initial deposit amount: "))

account1 = BankAccount(account_name, initial_deposit)

while True:
    print("\nChoose an action:")
    print("1. Deposit")
    print("2. Withdraw")
    print("3. Check Balance")
    print("4. Exit")

    choice = input("Enter choice (1-4): ")

    if choice == '1':
        amount = float(input("Enter deposit amount: "))
        account1.deposit(amount)
    elif choice == '2':
        amount = float(input("Enter withdrawal amount: "))
        account1.withdraw(amount)
    elif choice == '3':
        print(f"Current balance: ${account1.get_balance():.2f}")
    elif choice == '4':
        print("Exiting.")
        break
    else:
        print("Invalid choice. Please enter a number between 1 and 4.")
```

OUTPUT:

```
Enter account holder name: Alice
Enter initial deposit amount: 50000
Account created for Alice with initial balance $50000.00

Choose an action:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter choice (1-4): 2
Enter withdrawal amount: 3000
Withdrew: $3000.00

Choose an action:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter choice (1-4): 3
Current balance: $47000.00

Choose an action:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter choice (1-4): 1
Enter deposit amount: 4
Deposited: $4.00

Choose an action:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter choice (1-4):
```

OBSERVATION:

The code defines a BankAccount class with deposit, withdraw, and balance methods. It interactively takes user input for account actions. The output shows successful deposit, withdrawal, and balance checks based on user choices, demonstrating basic bank transactions.

CONCLUSION:

The code provides a functional simulation of a bank account using object-oriented programming. It allows users to perform core transactions like deposit and withdrawal and check their balance interactively, demonstrating class methods and user input handling.