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**

```python
#Use AI to complete a Student class with attributes and a method
#Class with constructor and display_details() method
class Student:
    def __init__(self, name, age, student_id):
        self.name = name
        self.age = age
        self.student_id = student_id

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Student ID: {self.student_id}")
# Example usage
student1 = Student("Alice", 20, "S12345")
student1.display_details()
student2 = Student("Bob", 22, "S67890")
student2.display_details()
student3 = Student("Charlie", 19, "S54321")
student3.display_details()
student4 = Student("Diana", 21, "S98765")
student4.display_details()

```

output:

```
Name: Alice

Age: 20

Student ID: S12345

Name: Bob

Age: 22

Student ID: S67890
```

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**

```
#Function to print the first 10 multiples of a given number
print("for loop multiples:")
def print_multiples(number):
    for i in range(1, 6):
        multiple = number * i
        print(f"{number} x {i} = {multiple}")
# Example usage
print_multiples(5)
#same code sing other controlled loops
print("while loop multiples:")
def print_multiples_while(number):
    i = 1
    while i <= 5:
        multiple = number * i
        print(f"{number} x {i} = {multiple}")
        i += 1
# Example usage
print_multiples_while(3)
print("do-while loop multiples:")
def print_multiples_do_while(number):
    i = 1
    while True:
        multiple = number * i
        print(f"{number} x {i} = {multiple}")
        i += 1
        if i > 5:
            break
# Example usage
print_multiples_do_while(7)
```

output:

```
for loop multiples:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5x 4 = 20
5 x 5= 25
While loop multiples:
3 x 1= 3
3 x 2 = 6
3 x 3= 9
3 x 4 = 12
3 x 45= 15
do-while loop multiples:
7 x 1 = 7
7 x 2= 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
```

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**

```
#Ask AI to write nested if-elif-else conditionals to classify age groups.
#Age classification function with appropriate conditions and with explanation
def classify_age(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"
# Example usage
ages = [5, 15, 30]
for age in ages:
    category = classify_age(age)
    print(f"Age: {age}, Category: {category}")
```

output:

```
Age: 5, Category: Child
Age: 15, Category: Teenager
Age: 30, Category: Adult
```

Explanation:

# The function classify_age takes an integer age as input and classifies it into different age groups using nested if-elif-else statements.

# It first checks if the age is negative, returning "Invalid age" if so.

# Then, it checks if the age is 12 or below to classify as "Child".

# Next, it checks if the age is between 13 and 19 to classify as "Teenager".

# After that, it checks if the age is between 20 and 64 to classify as "Adult".

# Finally, if none of the previous conditions are met, it classifies the age as "Senior" for ages 65 and above.

# The function is demonstrated with a list of example ages, printing out the corresponding category for each age.

# The function classify_age effectively uses conditional statements to determine the correct age group based on the provided age input.

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**

```
#Generate a sum_to_n() function to calculate sum of first n numbers

def sum_to_n(n):

  total = 0

  for i in range(1, n + 1):

    total += i

  return total

# Example usage

n = 10
```

```
result = sum_to_n(n)
print(f"The sum of the first {n} numbers is: {result}")
#other controlled looping
print("Using while loop:")
def sum_to_n_while(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total
# Example usage
n = 10
result = sum_to_n_while(n)
print(f"The sum of the first {n} numbers using while loop is: {result}")
print("Using do-while loop:")
def sum_to_n_do_while(n):
    total = 0
    i = 1
    while True:
        total += i
        i += 1
        if i > n:
            break
    return total
# Example usage
```

n = 10

result = sum_to_n_do_while(n)

print(f"The sum of the first {n} numbers using do-while loop is: {result}")

output:

The sum of the first 10 numbers is: 55

Using while loop:

The sum of the first 10 numbers using while loop is: 55

Using do-while loop:

The sum of the first 10 numbers using while loop is: 55

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**

```python
#build a BankAccount class with deposit, withdraw, and balance methods.
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return f"Deposited: ${amount:.2f}"
        else:
            return "Deposit amount must be positive."

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                return f"Withdrew: ${amount:.2f}"
            else:
                return "Insufficient funds."
        else:
            return "Withdrawal amount must be positive."

    def get_balance(self):
        return f"Current balance: ${self.balance:.2f}"
# Example usage
account = BankAccount(100)  # Create an account with an initial balance of $100
print(account.get_balance())  # Check balance
print(account.deposit(50))    # Deposit $50
print(account.get_balance())  # Check balance
print(account.withdraw(30))   # Withdraw $30
print(account.get_balance())  # Check balance
print(account.withdraw(150))  # Attempt to withdraw $150 (should fail)
print(account.get_balance())  # Check balance
print(account.deposit(-20))   # Attempt to deposit a negative amount (should fail)

```

output:

```
Current balance: $100.00

Deposited: $50.00

Current balance: $150.00

Withdrew: $30.00

Current balance: $120.00

Insufficient funds.

Current balance: $120.00

Deposit amount must be positive.
```