AI ASSISTED CODING

NAME:SANIYA

ROLL NO:2403A51OE7

Lab 6: Al-Based Code Completion Classes, Loops, and Conditionals

Lab Objectives:

To explore Al-powered auto-completion features for core Python constructs.
To analyze how Al suggests logic for class definitions, loops, and conditionals.
To evaluate the completeness and correctness of code generated by Al assistants.

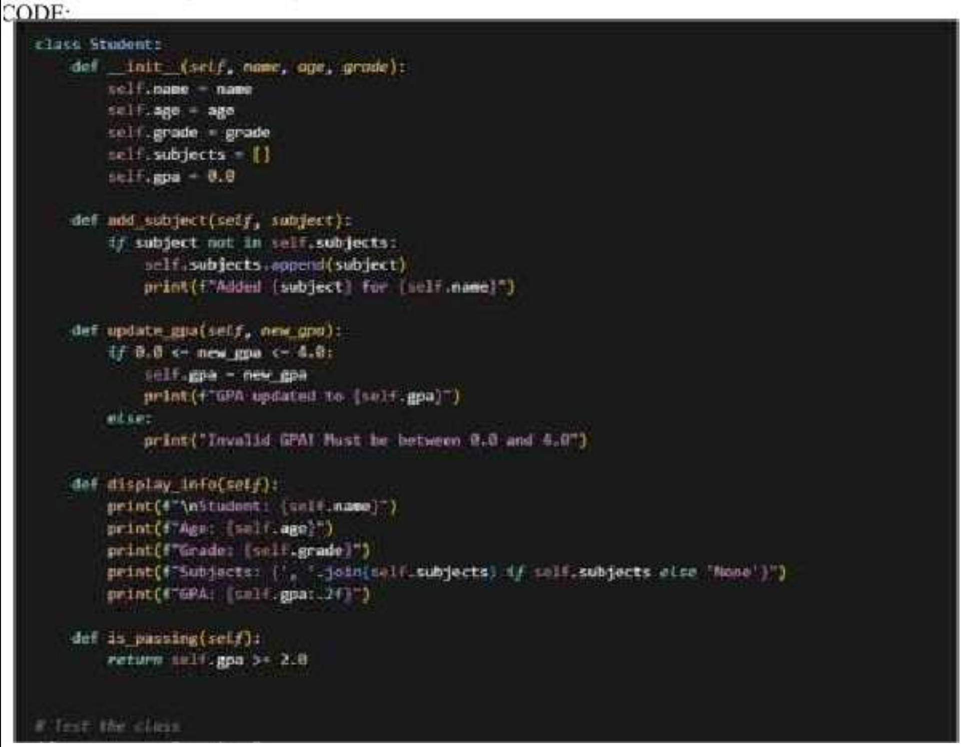
Lab Outcomes (LOS):
After completing this lab, students will be able to:

Use Al tools to generate and complete class definitions and methods.
Understand and assess Al-suggested loops for iterative tasks.
Generate conditional statements through prompt-driven suggestions.
Critically evaluate Al-assisted code for correctness and clarity.

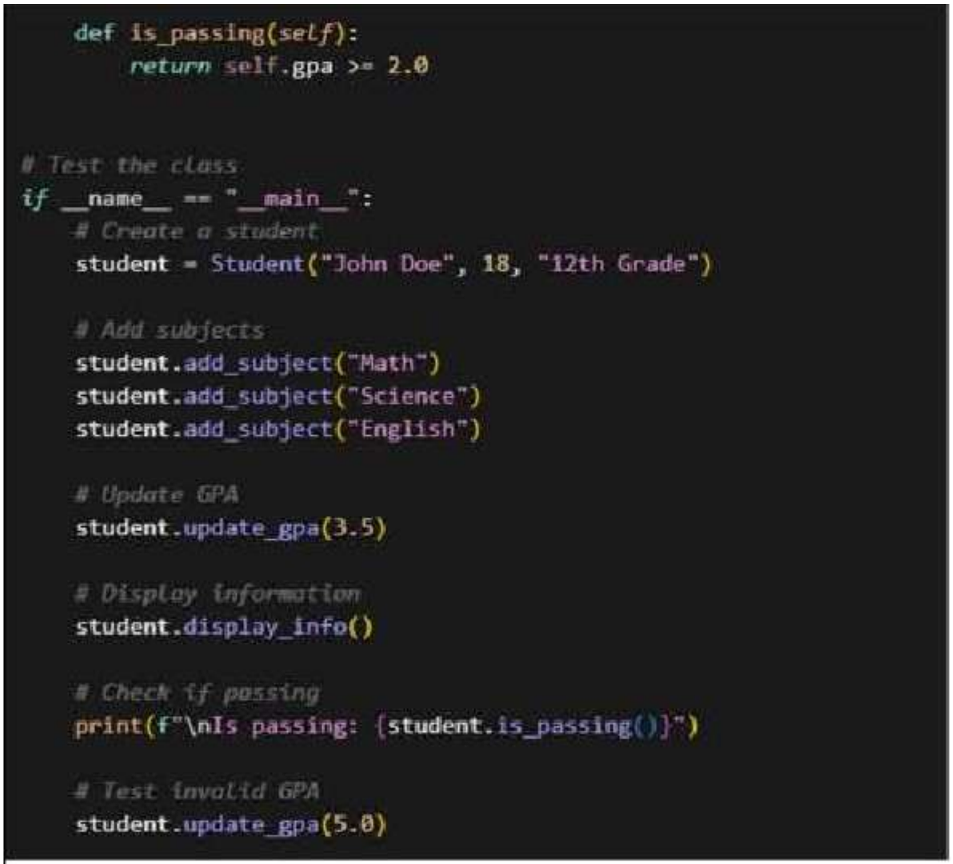
Task Description#l (Classes)
Use Al to complete a Student class with attributes and a method.
Check output
Analyze the code generated by Al tool

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

CODE:



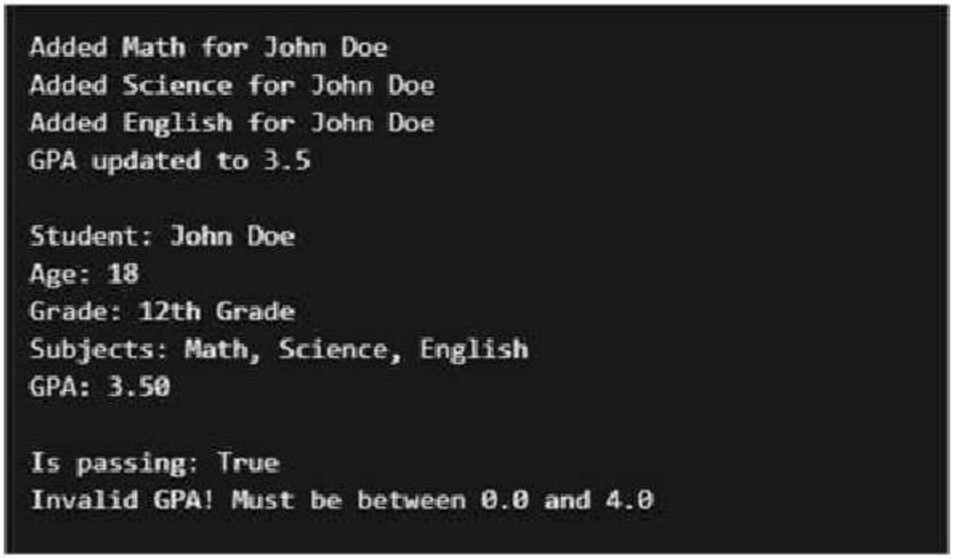
```
class Student:
    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade
        self.subjects = []
        self.gpa = 0.0

    def add_subject(self, subject):
        if subject not in self.subjects:
            self.subjects.append(subject)
            print(f"Added {subject} for {self.name}")

    def update_gpa(self, new_gpa):
        if 0.0 <= new_gpa <= 4.0:
            self.gpa = new_gpa
            print(f"GPA updated to {self.gpa}")
        else:
            print("Invalid GPA! Must be between 0.0 and 4.0")

    def display_info(self):
        print(f"\nStudent: {self.name}")
        print(f"Age: {self.age}")
        print(f"Grade: {self.grade}")
        print(f"Subjects: {', '.join(self.subjects) if self.subjects else 'None'}")
        print(f"GPA: {self.gpa:.2f}")

    def is_passing(self):
        return self.gpa >= 2.0


# Test the class
```



```
    def is_passing(self):
        return self.gpa >= 2.0


# Test the class
if __name__ == "__main__":
    # Create a student
    student = Student("John Doe", 18, "12th Grade")

    # Add subjects
    student.add_subject("Math")
    student.add_subject("Science")
    student.add_subject("English")

    # Update GPA
    student.update_gpa(3.5)

    # Display information
    student.display_info()

    # Check if passing
    print(f"\nIs passing: {student.is_passing()}")

    # Test invalid GPA
    student.update_gpa(5.0)
```

Expected Output#1

- Class with constructor and display_details() method



```
Added Math for John Doe
Added Science for John Doe
Added English for John Doe
GPA updated to 3.5

Student: John Doe
Age: 18
Grade: 12th Grade
Subjects: Math, Science, English
GPA: 3.50

Is passing: True
Invalid GPA! Must be between 0.0 and 4.0
```

Task Description#2 (Loops)

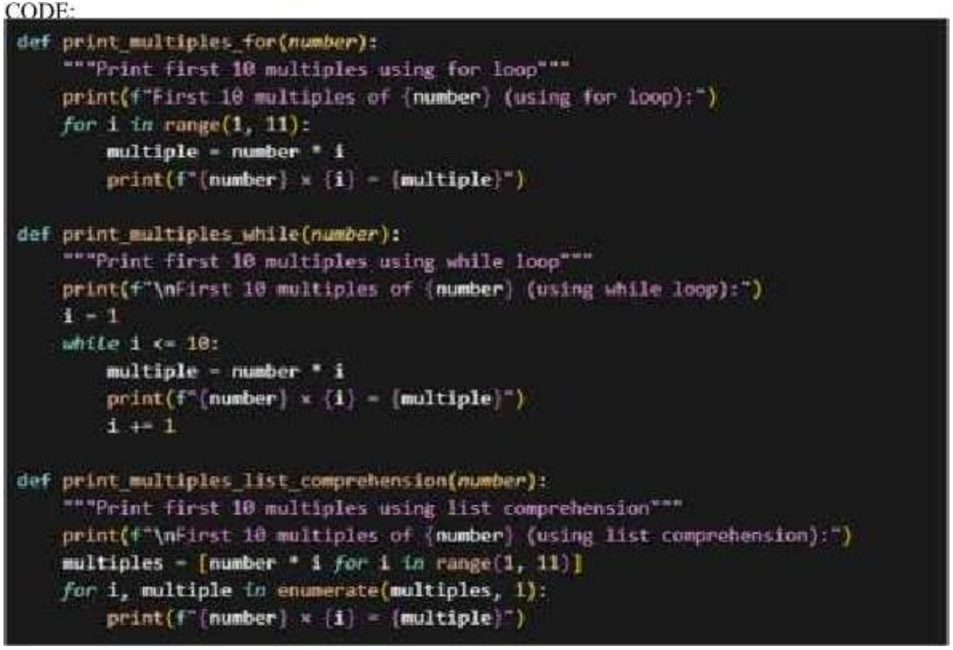
- Prompt AI to complete a function that prints the first 10 multiples of a number using a loop.
- Analyze the generated code
- Ask AI to generate code using other controlled looping

PROMPT:

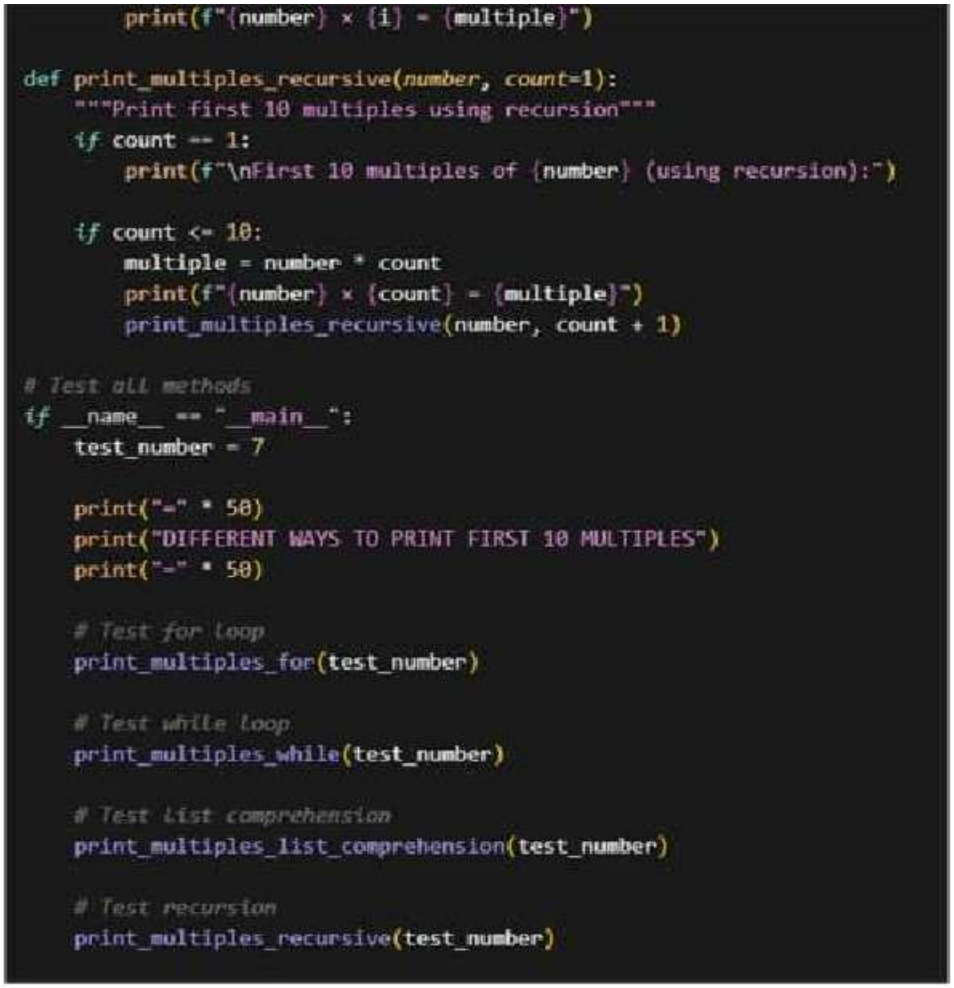
Use AI to complete a Student class with attributes and a method.

- Check output
- Analyze the code generated by AI tool

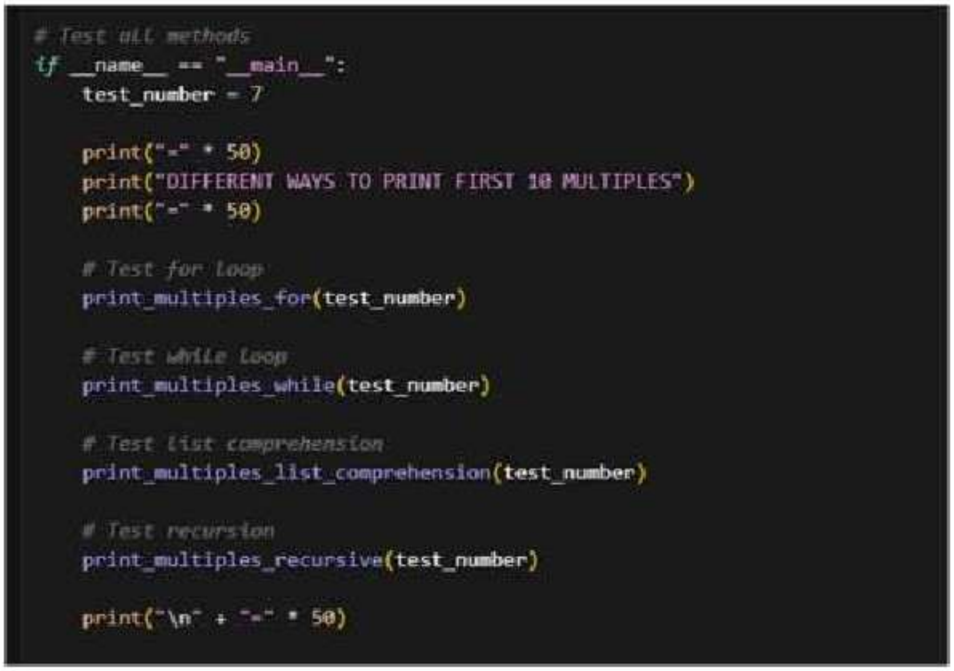
CODE:



```
def print_multiples_for(number):
    """Print first 10 multiples using for loop"""
    print(f"First 10 multiples of {number} (using for loop):")
    for i in range(1, 11):
        multiple = number * i
        print(f"{number} × {i} = {multiple}")

def print_multiples_while(number):
    """Print first 10 multiples using while loop"""
    print(f"\nFirst 10 multiples of {number} (using while loop):")
    i = 1
    while i <= 10:
        multiple = number * i
        print(f"{number} × {i} = {multiple}")
        i += 1

def print_multiples_list_comprehension(number):
    """Print first 10 multiples using list comprehension"""
    print(f"\nFirst 10 multiples of {number} (using list comprehension):")
    multiples = [number * i for i in range(1, 11)]
    for i, multiple in enumerate(multiples, 1):
        print(f"{number} × {i} = {multiple}")
```



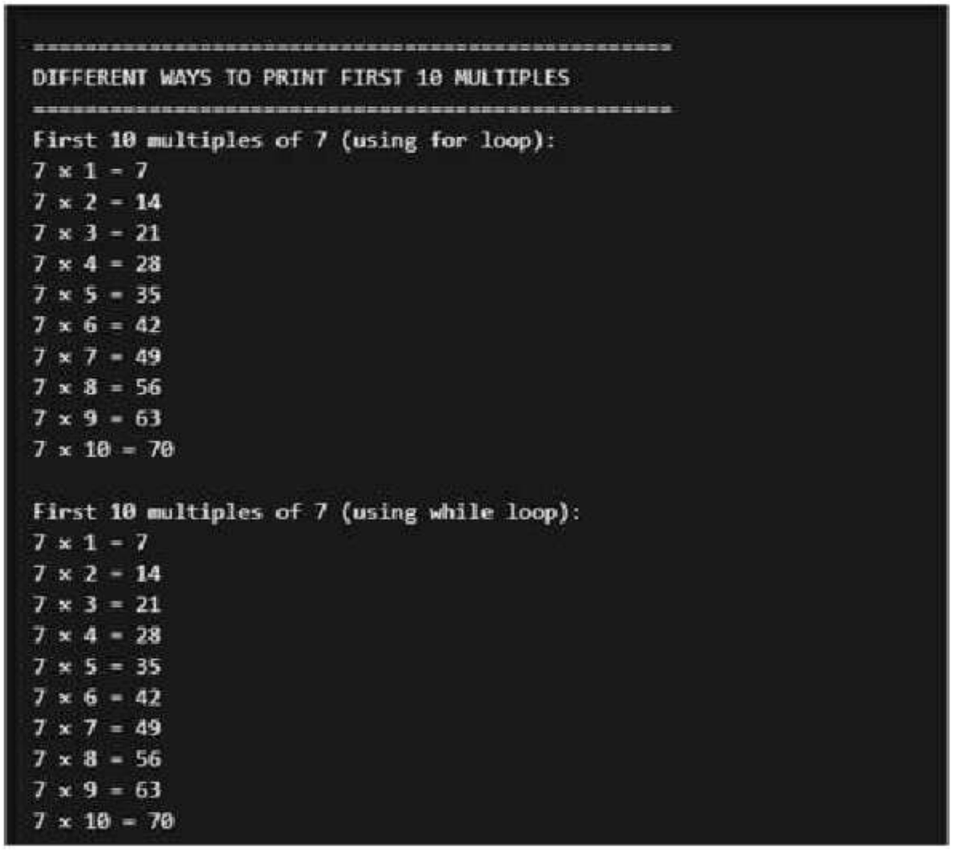
```python
        print(f"{number} × {i} = {multiple}")

def print_multiples_recursive(number, count=1):
    """Print first 10 multiples using recursion"""
    if count == 1:
        print(f"\nFirst 10 multiples of {number} (using recursion):")

    if count <= 10:
        multiple = number * count
        print(f"{number} × {count} = {multiple}")
        print_multiples_recursive(number, count + 1)

# Test all methods
if __name__ == "__main__":
    test_number = 7

    print("=" * 50)
    print("DIFFERENT WAYS TO PRINT FIRST 10 MULTIPLES")
    print("=" * 50)

    # Test for loop
    print_multiples_for(test_number)

    # Test while loop
    print_multiples_while(test_number)

    # Test list comprehension
    print_multiples_list_comprehension(test_number)

    # Test recursion
    print_multiples_recursive(test_number)
```



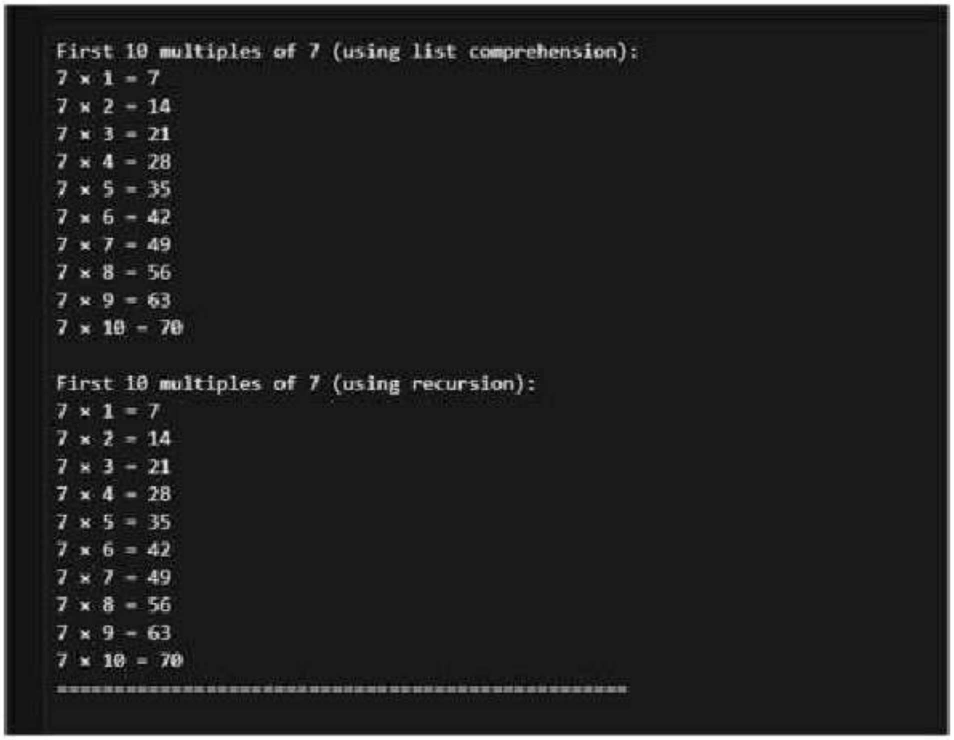
```python
# Test all methods
if __name__ == "__main__":
    test_number = 7

    print("=" * 50)
    print("DIFFERENT WAYS TO PRINT FIRST 10 MULTIPLES")
    print("=" * 50)

    # Test for loop
    print_multiples_for(test_number)

    # Test while loop
    print_multiples_while(test_number)

    # Test list comprehension
    print_multiples_list_comprehension(test_number)

    # Test recursion
    print_multiples_recursive(test_number)

    print("\n" + "=" * 50)
```

Expected Output#2

- Correct loop-based implementation



```
======================================================
DIFFERENT WAYS TO PRINT FIRST 10 MULTIPLES
======================================================
First 10 multiples of 7 (using for loop):
7 × 1 = 7
7 × 2 = 14
7 × 3 = 21
7 × 4 = 28
7 × 5 = 35
7 × 6 = 42
7 × 7 = 49
7 × 8 = 56
7 × 9 = 63
7 × 10 = 70

First 10 multiples of 7 (using while loop):
7 × 1 = 7
7 × 2 = 14
7 × 3 = 21
7 × 4 = 28
7 × 5 = 35
7 × 6 = 42
7 × 7 = 49
7 × 8 = 56
7 × 9 = 63
7 × 10 = 70
```



```
First 10 multiples of 7 (using list comprehension):
7 × 1 = 7
7 × 2 = 14
7 × 3 = 21
7 × 4 = 28
7 × 5 = 35
7 × 6 = 42
7 × 7 = 49
7 × 8 = 56
7 × 9 = 63
7 × 10 = 70

First 10 multiples of 7 (using recursion):
7 × 1 = 7
7 × 2 = 14
7 × 3 = 21
7 × 4 = 28
7 × 5 = 35
7 × 6 = 42
7 × 7 = 49
7 × 8 = 56
7 × 9 = 63
7 × 10 = 70
======================================================
```

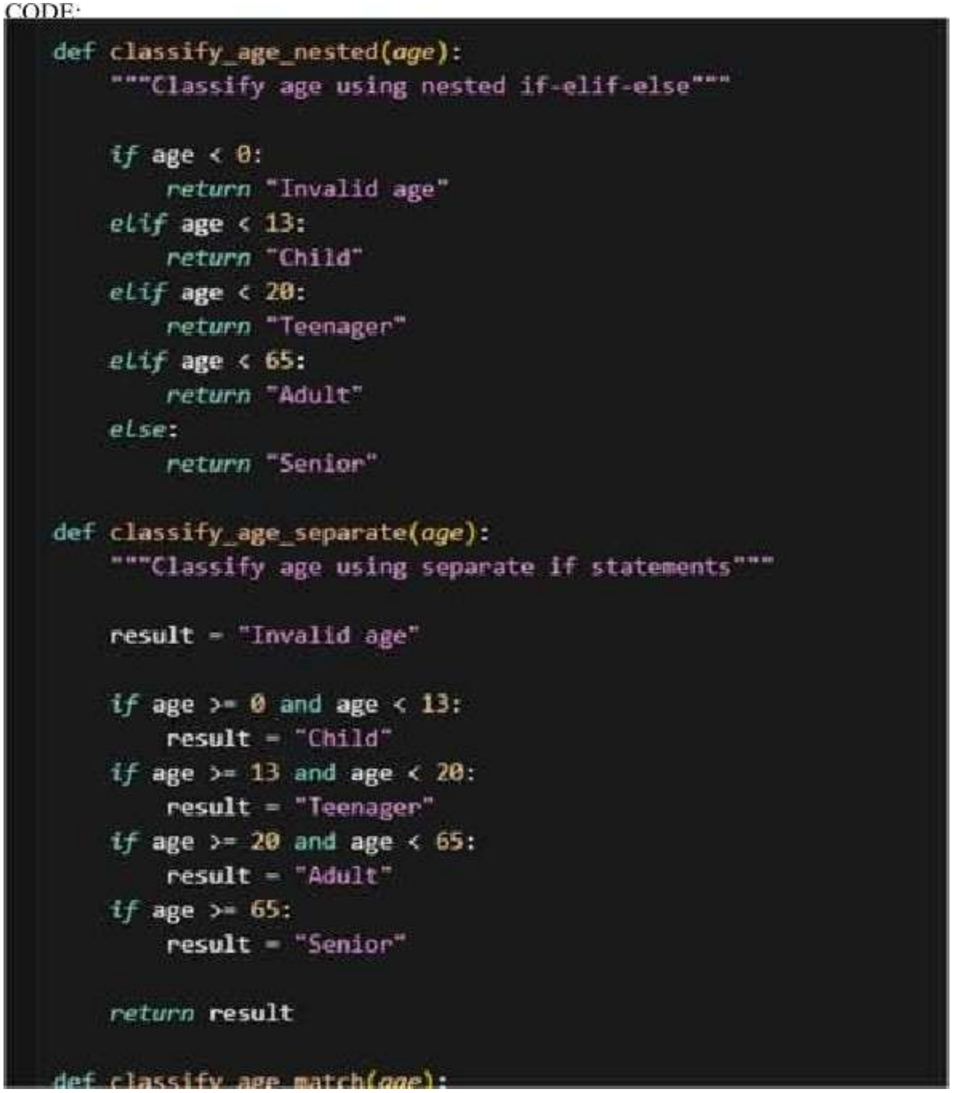
Task Description#3 (Conditional Statements)

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

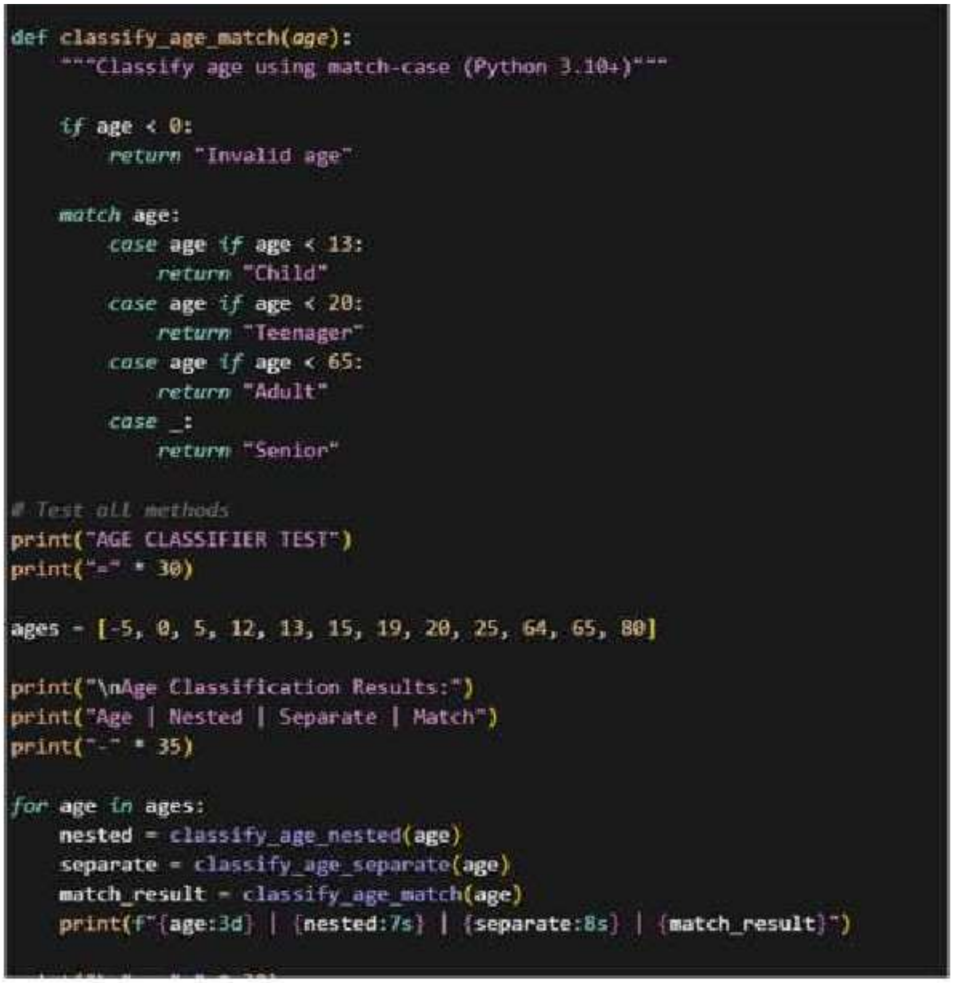
PROMT:

Write a Python program using nested if-elif-else statements to classify people into age groups (child, teenager, adult, senior). Then analyze the code and explain how it works. After that, rewrite the program using a different type of conditional structure (like separate if conditions or match-case).

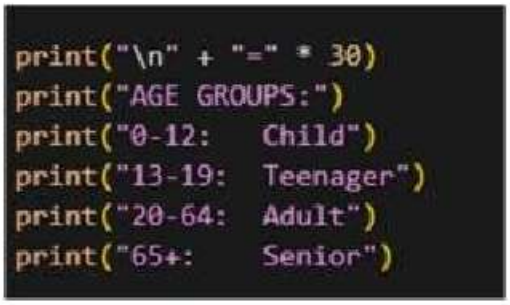
CODE:



```
def classify_age_nested(age):
    """Classify age using nested if-elif-else"""

    if age < 0:
        return "Invalid age"
    elif age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 65:
        return "Adult"
    else:
        return "Senior"

def classify_age_separate(age):
    """Classify age using separate if statements"""

    result = "Invalid age"

    if age >= 0 and age < 13:
        result = "Child"
    if age >= 13 and age < 20:
        result = "Teenager"
    if age >= 20 and age < 65:
        result = "Adult"
    if age >= 65:
        result = "Senior"

    return result

def classify_age_match(age):
```



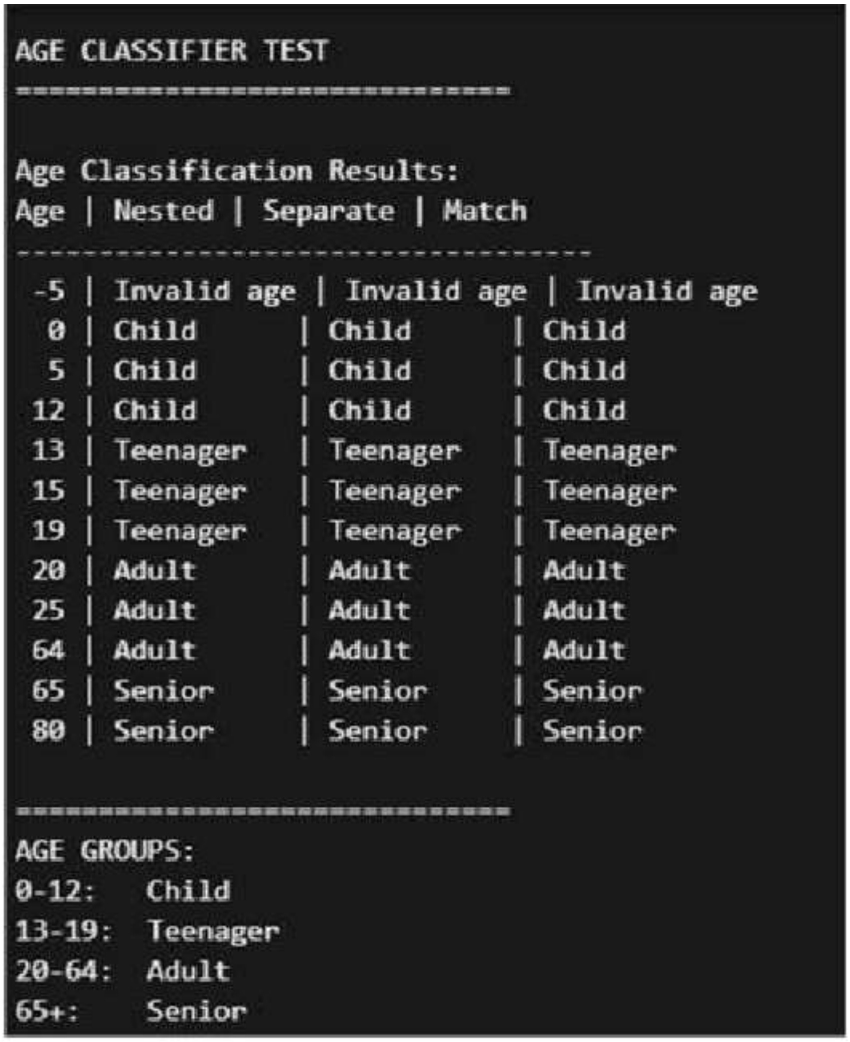
```
def classify_age_match(age):
    """Classify age using match-case (Python 3.10+)"""

    if age < 0:
        return "Invalid age"

    match age:
        case age if age < 13:
            return "Child"
        case age if age < 20:
            return "Teenager"
        case age if age < 65:
            return "Adult"
        case _:
            return "Senior"

# Test all methods
print("AGE CLASSIFIER TEST")
print("=" * 30)

ages = [-5, 0, 5, 12, 13, 15, 19, 20, 25, 64, 65, 80]

print("\nAge Classification Results:")
print("Age | Nested | Separate | Match")
print("-" * 35)

for age in ages:
    nested = classify_age_nested(age)
    separate = classify_age_separate(age)
    match_result = classify_age_match(age)
    print(f"{age:3d} | {nested:7s} | {separate:8s} | {match_result}")
```



```
print("\n" + "=" * 30)
print("AGE GROUPS:")
print("0-12:   Child")
print("13-19:  Teenager")
print("20-64:  Adult")
print("65+:    Senior")
```

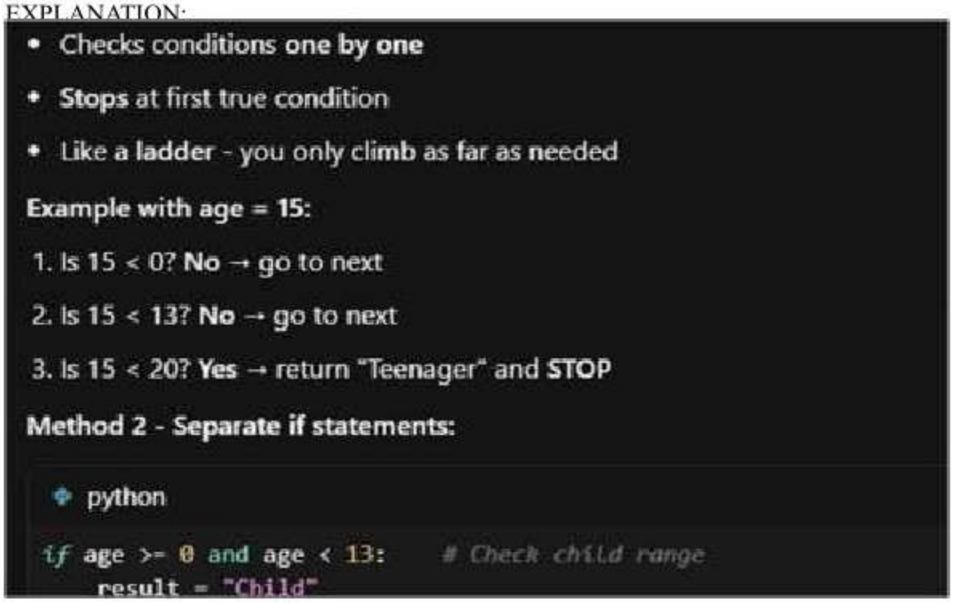
Expected Output#3

- Age classification function with appropriate conditions and with explanation



```
AGE CLASSIFIER TEST
================================

Age Classification Results:
Age | Nested | Separate | Match
------------------------------------
 -5 | Invalid age | Invalid age | Invalid age
  0 | Child        | Child        | Child
  5 | Child        | Child        | Child
 12 | Child        | Child        | Child
 13 | Teenager     | Teenager     | Teenager
 15 | Teenager     | Teenager     | Teenager
 19 | Teenager     | Teenager     | Teenager
 20 | Adult        | Adult        | Adult
 25 | Adult        | Adult        | Adult
 64 | Adult        | Adult        | Adult
 65 | Senior       | Senior       | Senior
 80 | Senior       | Senior       | Senior

================================
AGE GROUPS:
0-12:   Child
13-19:  Teenager
20-64:  Adult
65+:    Senior
```

EXPLANATION:



- Checks conditions **one by one**
- **Stops** at first true condition
- Like **a ladder** - you only **climb** as **far** as **needed**

Example with age = 15:

1. Is 15 < 0? **No** → go to next
2. Is 15 < 13? **No** → go to next
3. Is 15 < 20? **Yes** → return "Teenager" and **STOP**

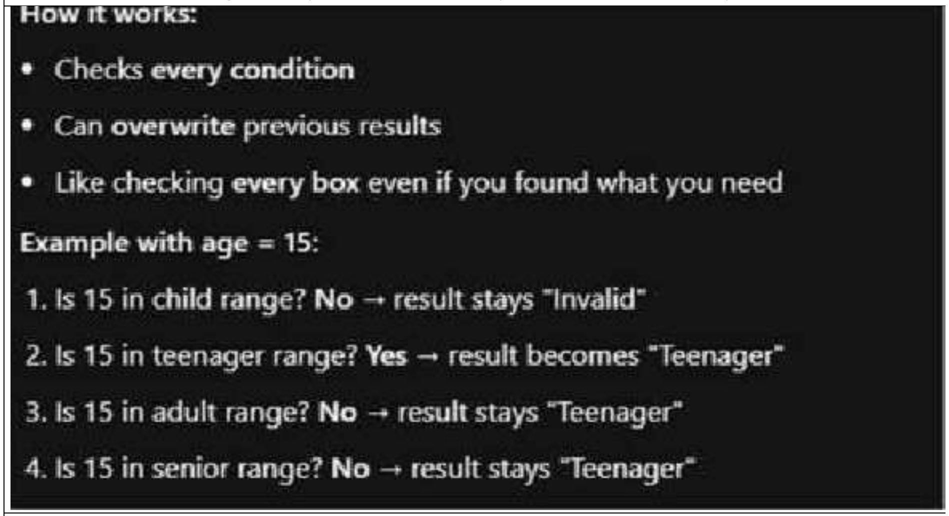
Method 2 - Separate if statements:

python

```python
if age >= 0 and age < 13:    # Check child range
    result = "Child"
```

How it works:

- Checks every condition
- Can overwrite previous results
- Like checking every box even if you found what you need



How it works:

- Checks **every condition**
- Can **overwrite** previous results
- Like checking **every box** even if you found what you need

Example with age = 15:

1. Is 15 in child range? **No** → result stays "Invalid"
2. Is 15 in teenager range? **Yes** → result becomes "Teenager"
3. Is 15 in adult range? **No** → result stays "Teenager"
4. Is 15 in senior range? **No** → result stays "Teenager"

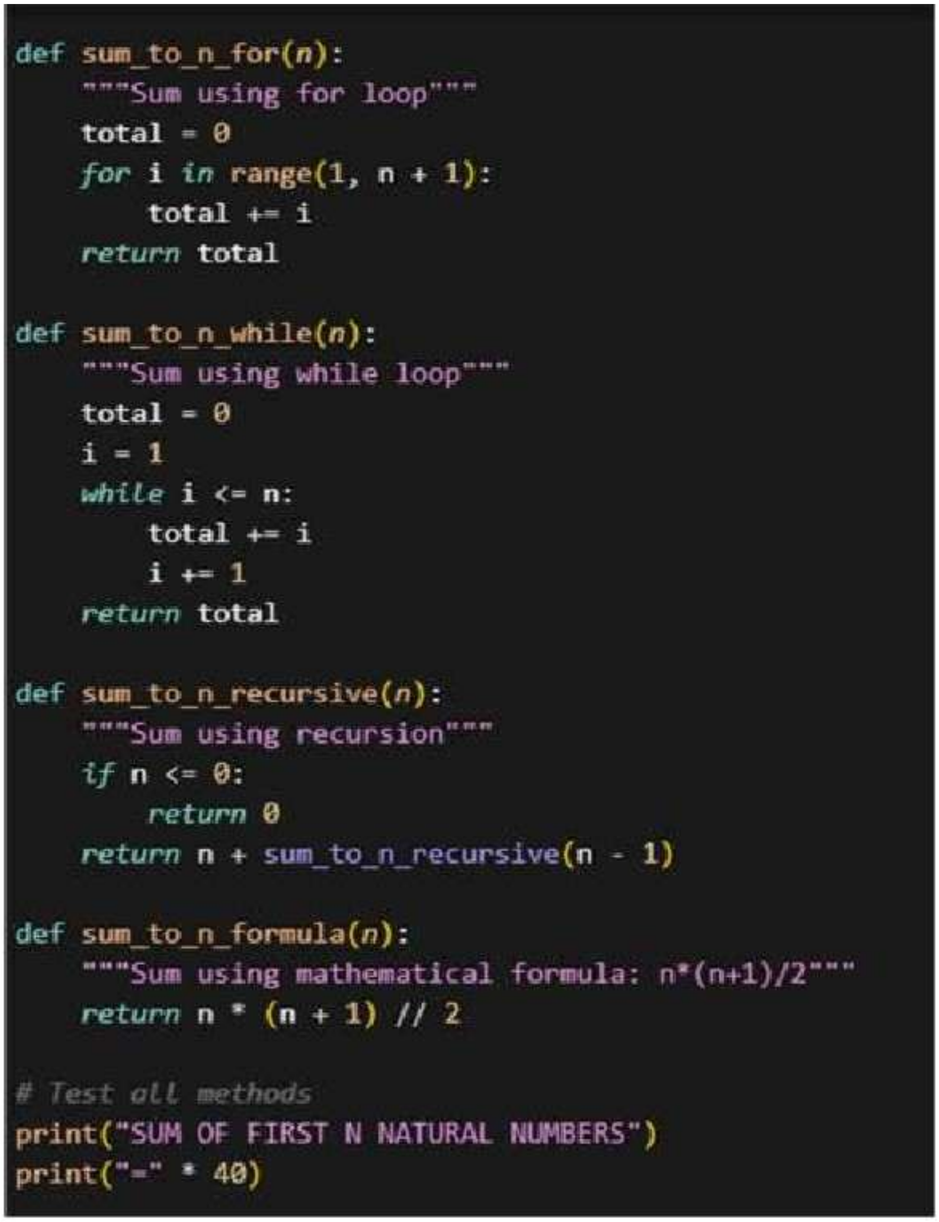
2. Is 15 in teenager range? Yes result becomes "Teenager"
3. Is 15 in adult range? No —s result stays "Teenager"
4. Is 15 in senior range? No —v result stays "Teenager-

Task Description#4 (For and While loops)

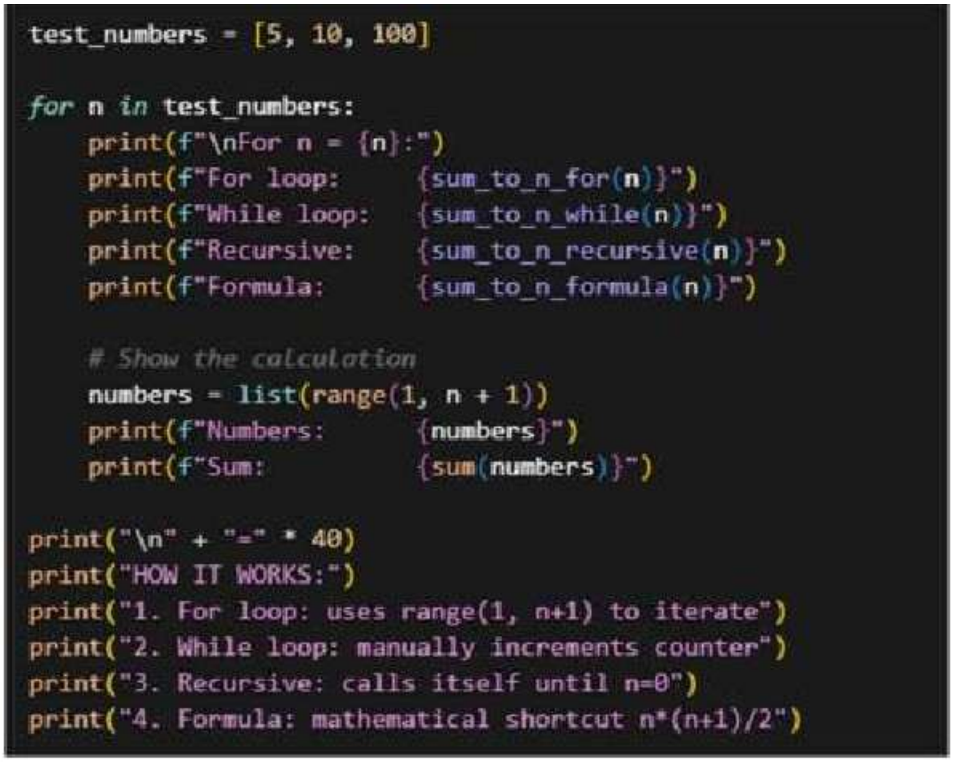
Generate a sum_to n() function to calculate sum offirst n numbers
Analyze the generated code
Get suggestions from Al with other controlled looping

PROOMT:

Write a Python function sum_to_n(n) that calculates the sum of the first n natural numbers using a loop. Explain how the code works. Then, suggest and show other ways to do it using different loops (like while loop, for loop, etc.).



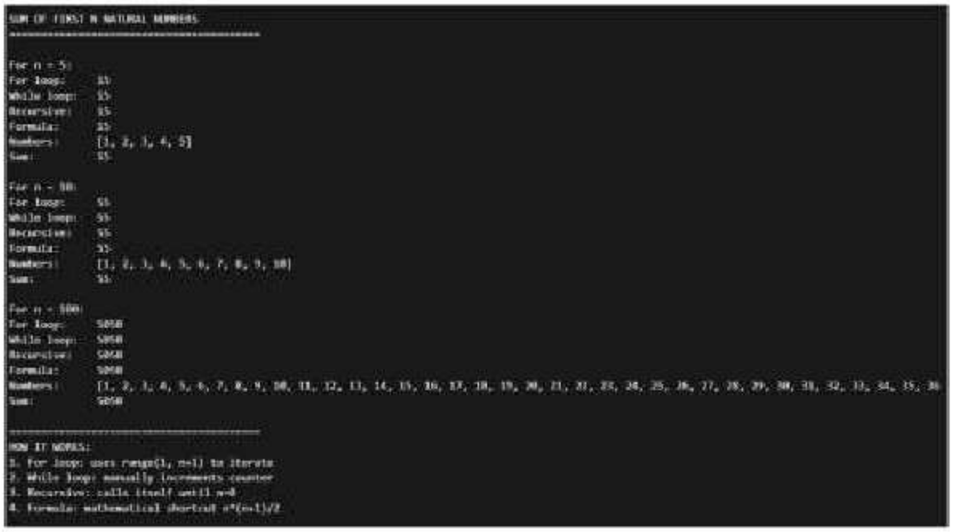
```python
def sum_to_n_for(n):
    """Sum using for loop"""
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

def sum_to_n_while(n):
    """Sum using while loop"""
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total

def sum_to_n_recursive(n):
    """Sum using recursion"""
    if n <= 0:
        return 0
    return n + sum_to_n_recursive(n - 1)

def sum_to_n_formula(n):
    """Sum using mathematical formula: n*(n+1)/2"""
    return n * (n + 1) // 2

# Test all methods
print("SUM OF FIRST N NATURAL NUMBERS")
print("=" * 40)
```



```python
test_numbers = [5, 10, 100]

for n in test_numbers:
    print(f"\nFor n = {n}:")
    print(f"For loop:     {sum_to_n_for(n)}")
    print(f"While loop:   {sum_to_n_while(n)}")
    print(f"Recursive:    {sum_to_n_recursive(n)}")
    print(f"Formula:      {sum_to_n_formula(n)}")

    # Show the calculation
    numbers = list(range(1, n + 1))
    print(f"Numbers:      {numbers}")
    print(f"Sum:          {sum(numbers)}")

print("\n" + "=" * 40)
print("HOW IT WORKS:")
print("1. For loop: uses range(1, n+1) to iterate")
print("2. While loop: manually increments counter")
print("3. Recursive: calls itself until n=0")
print("4. Formula: mathematical shortcut n*(n+1)/2")
```

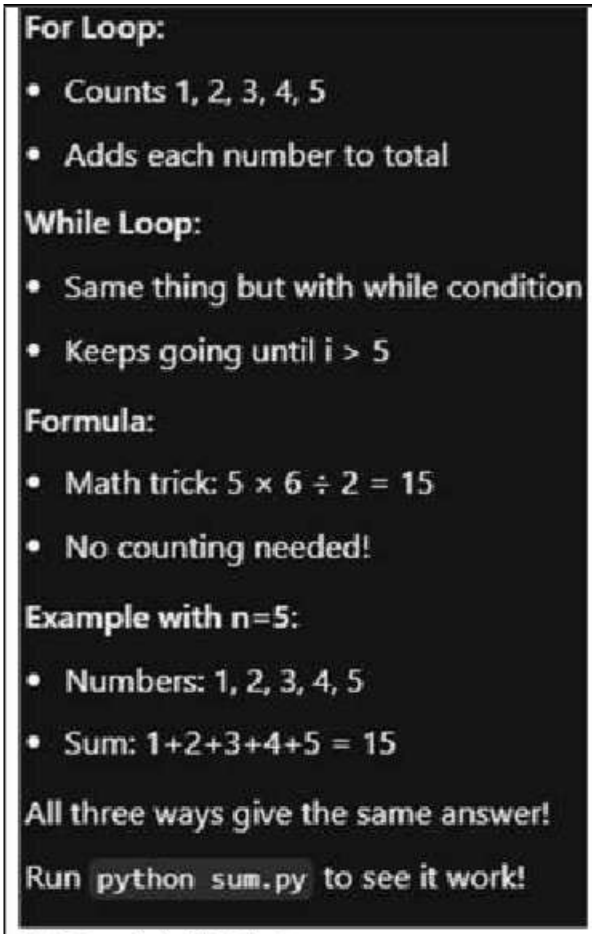
Expected Output#4

- Python code with explanation



```
SUM OF FIRST N NATURAL NUMBERS
========================================

For n = 5:
For loop:     15
While loop:   15
Recursive:    15
Formula:      15
Numbers:      [1, 2, 3, 4, 5]
Sum:          15

For n = 10:
For loop:     55
While loop:   55
Recursive:    55
Formula:      55
Numbers:      [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum:          55

For n = 100:
For loop:     5050
While loop:   5050
Recursive:    5050
Formula:      5050
Numbers:      [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36
Sum:          5050

========================================
HOW IT WORKS:
1. For loop: uses range(1, n+1) to iterate
2. While loop: manually increments counter
3. Recursive: calls itself until n=0
4. Formula: mathematical shortcut n*(n+1)/2
```

EXPLANATION:



For Loop:

- Counts 1, 2, 3, 4, 5
- Adds each number to total

While Loop:

- Same thing but with while condition
- Keeps going until i > 5

Formula:

- Math trick: 5 × 6 ÷ 2 = 15
- No counting needed!

Example with n=5:

- Numbers: 1, 2, 3, 4, 5
- Sum: 1+2+3+4+5 = 15

All three ways give the same answer!

Run `python sum.py` to see it work!

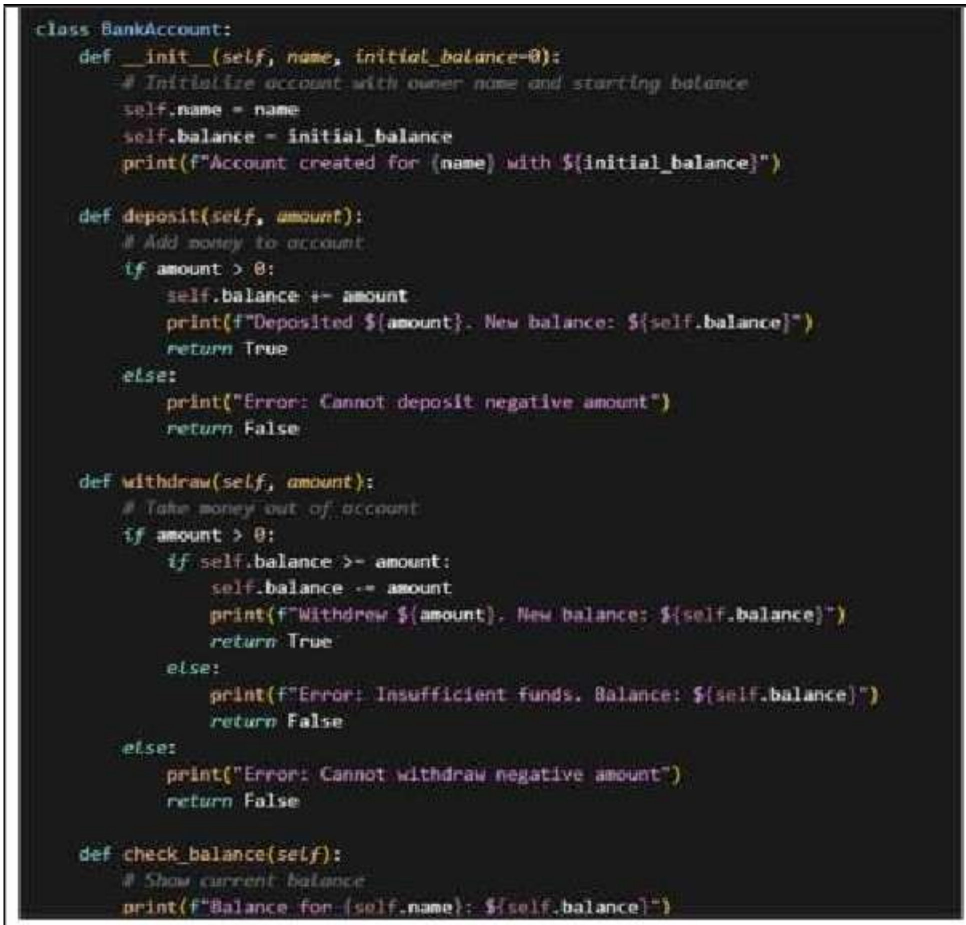
Task Description#5 (Class)

- Use AI to build a BankAccount class with deposit, withdraw, and balance methods.
- Analyze the generated code
- Add comments and explain code

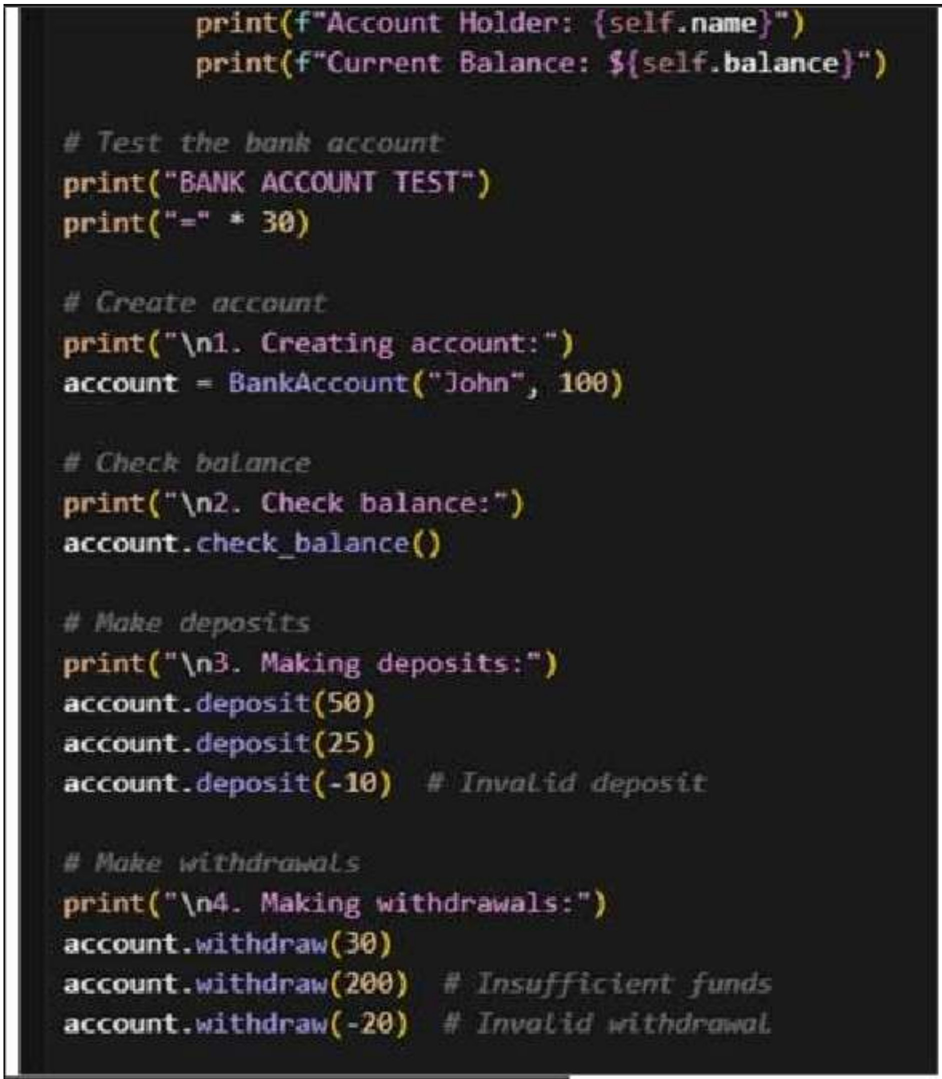
PROMT:

Write a Python class BankAccount with methods to deposit money, withdraw money, and check balance. Explain how the code works. Then add comments in the code to make it easy to understand.

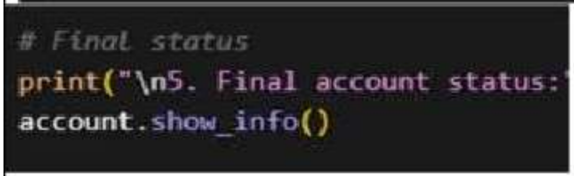
CODE:



```python
class BankAccount:
    def __init__(self, name, initial_balance=0):
        # Initialize account with owner name and starting balance
        self.name = name
        self.balance = initial_balance
        print(f"Account created for {name} with ${initial_balance}")

    def deposit(self, amount):
        # Add money to account
        if amount > 0:
            self.balance += amount
            print(f"Deposited ${amount}. New balance: ${self.balance}")
            return True
        else:
            print("Error: Cannot deposit negative amount")
            return False

    def withdraw(self, amount):
        # Take money out of account
        if amount > 0:
            if self.balance >= amount:
                self.balance -= amount
                print(f"Withdrew ${amount}. New balance: ${self.balance}")
                return True
            else:
                print(f"Error: Insufficient funds. Balance: ${self.balance}")
                return False
        else:
            print("Error: Cannot withdraw negative amount")
            return False

    def check_balance(self):
        # Show current balance
        print(f"Balance for {self.name}: ${self.balance}")
```



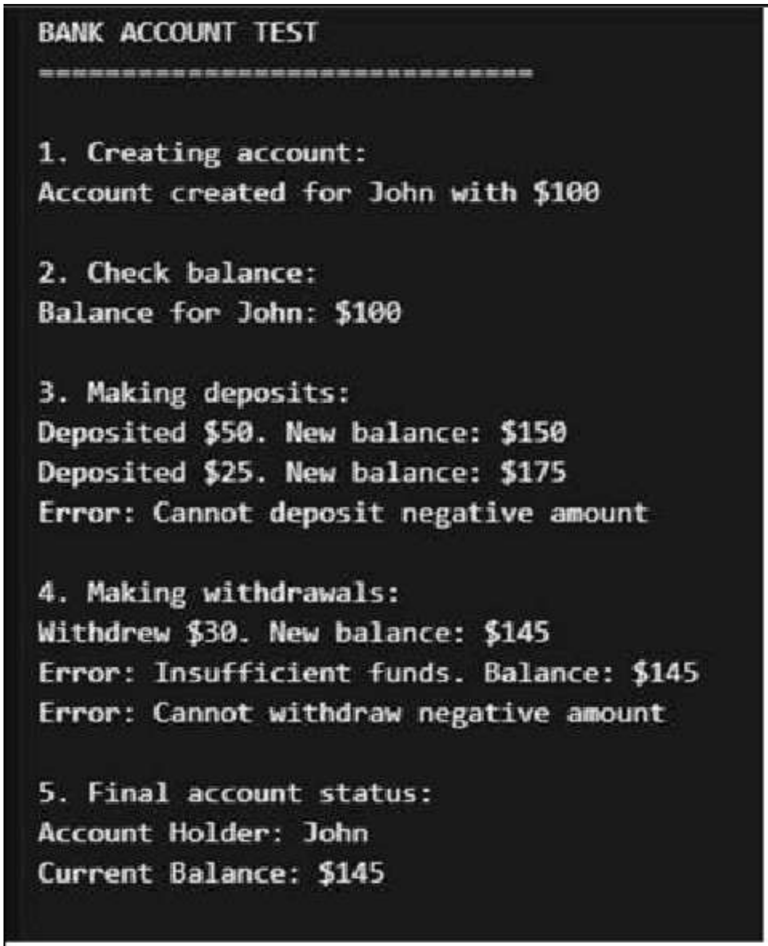
```
        print(f"Account Holder: {self.name}")
        print(f"Current Balance: ${self.balance}")

# Test the bank account
print("BANK ACCOUNT TEST")
print("=" * 30)

# Create account
print("\n1. Creating account:")
account = BankAccount("John", 100)

# Check balance
print("\n2. Check balance:")
account.check_balance()

# Make deposits
print("\n3. Making deposits:")
account.deposit(50)
account.deposit(25)
account.deposit(-10)  # Invalid deposit

# Make withdrawals
print("\n4. Making withdrawals:")
account.withdraw(30)
account.withdraw(200)  # Insufficient funds
account.withdraw(-20)  # Invalid withdrawal
```



```
# Final status
print("\n5. Final account status:
account.show_info()
```

Expected Output#5

- Python code with explanation



```
BANK ACCOUNT TEST
==============================

1. Creating account:
Account created for John with $100

2. Check balance:
Balance for John: $100

3. Making deposits:
Deposited $50. New balance: $150
Deposited $25. New balance: $175
Error: Cannot deposit negative amount

4. Making withdrawals:
Withdrew $30. New balance: $145
Error: Insufficient funds. Balance: $145
Error: Cannot withdraw negative amount

5. Final account status:
Account Holder: John
Current Balance: $145
```