AI Assistant Coding

Assignement-6.3

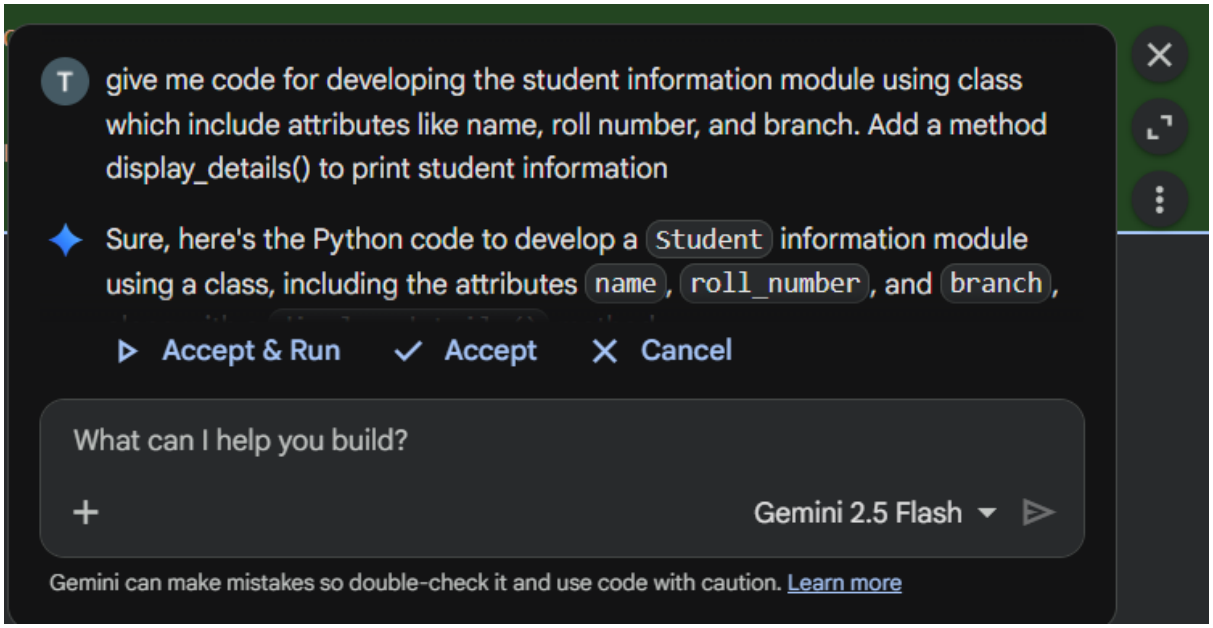
Name: Aman Sarkar

Roll No: 2303A51273

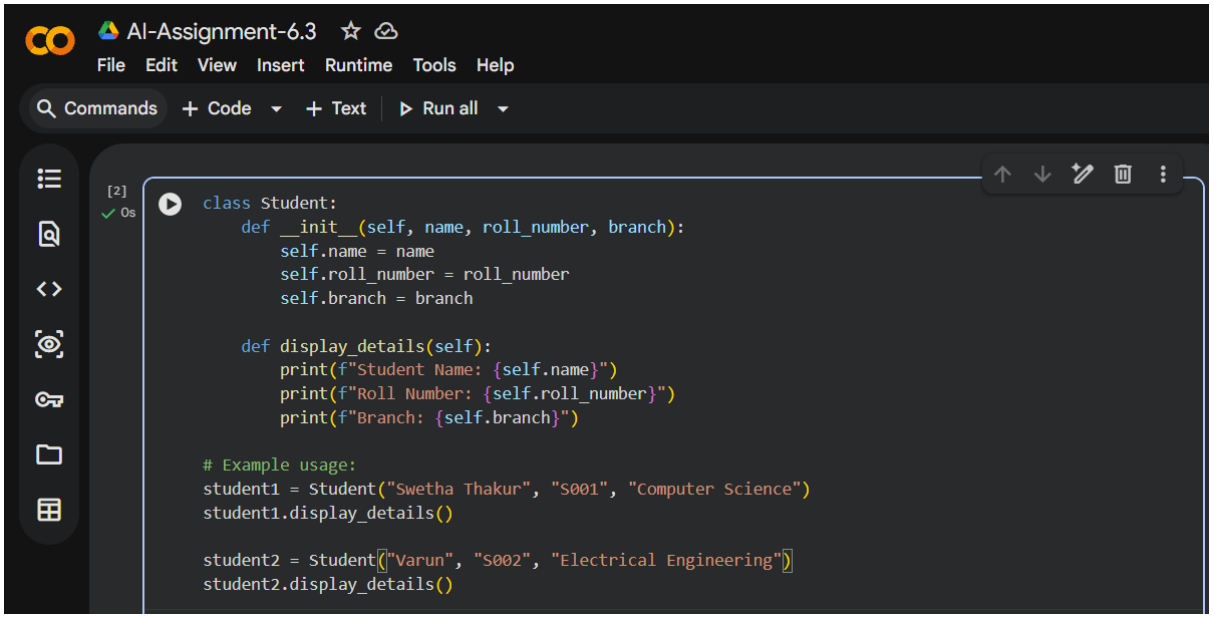
Batch: 05

Task Description #1: Classes (Student Class)

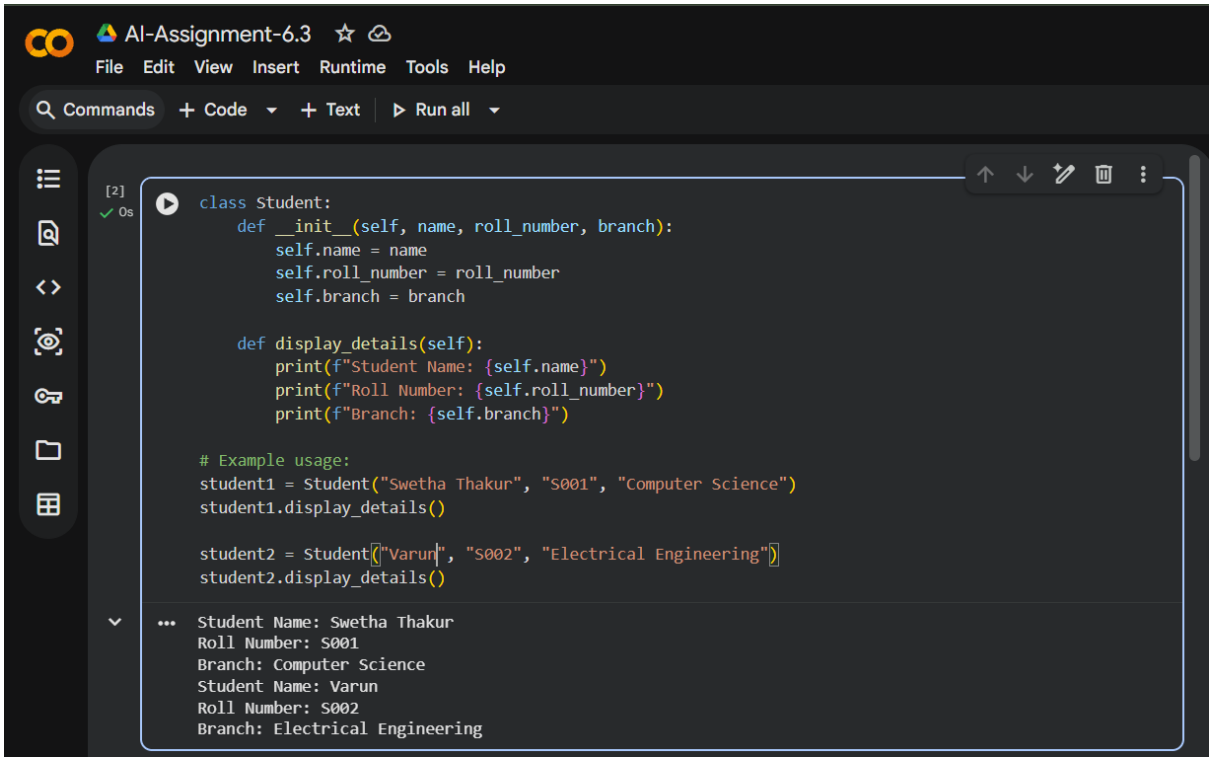
Prompt:



Code:



Output:



Code Explanation:

Class Definition

- class Student: → Defines a class named Student, which acts as a blueprint for creating student objects.

Constructor (__init__ method)

- def __init__(self, name, roll_number, branch): → Special method called automatically when a new object is created.

_Parameters:

  - self → Refers to the current object (instance of the class).
  - name, roll_number, branch → Attributes passed during object creation.

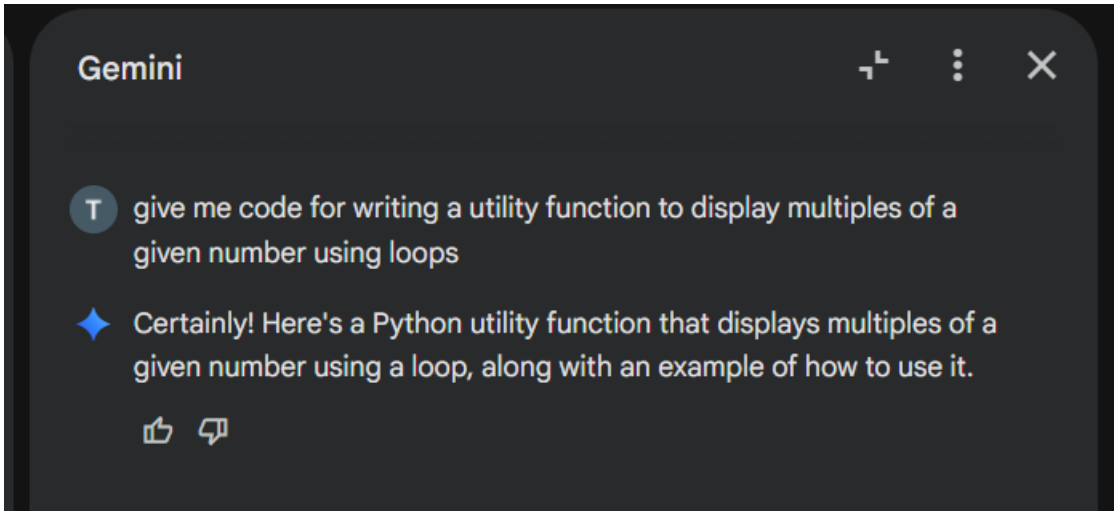
- Inside the constructor:
- self.name = name → Assigns the given name to the object.
- self.roll_number = roll_number → Assigns the roll number.
- self.branch = branch → Assigns the branch.

Instance Method

- def display_details(self): → Defines a method to display student details.
- Uses f-strings for formatted output:
- print(f"Student Name: {self.name}")
- print(f"Roll Number: {self.roll_number}")
- print(f"Branch: {self.branch}")

Task Description #2: Loops (Multiples of a Number)

Prompt:



Code:

```python
def display_multiples(number, count):
    """
    Displays a specified number of multiples for a given number.

    Args:
        number (int): The base number.
        count (int): The number of multiples to display.
    """
    print(f"Multiples of {number}:")
    for i in range(1, count + 1):
        multiple = number * i
        print(multiple, end=" ")
    print() # For a new line after all multiples

# Example usage:
display_multiples(7, 5) # Display 5 multiples of 7
display_multiples(3, 10) # Display 10 multiples of 3
```

Output:

```
Multiples of 7:
7 14 21 28 35
Multiples of 3:
3 6 9 12 15 18 21 24 27 30
```

Code Explanation:

Function Definition

- def display_multiples(number, count):
- Defines a function named display_multiples.

Parameters:

- number → the base number whose multiples will be displayed.
- count → how many multiples to display.

Printing Header

- print(f"Multiples of {number}:")
- Prints a header showing which number's multiples are being displayed.
- Uses an f-string for formatting.

Loop for Multiples

- for i in range(1, count + 1):
    - Loops from 1 up to count (inclusive).
    - Example: if count = 5, loop runs for i = 1, 2, 3, 4, 5.
- Inside the loop:

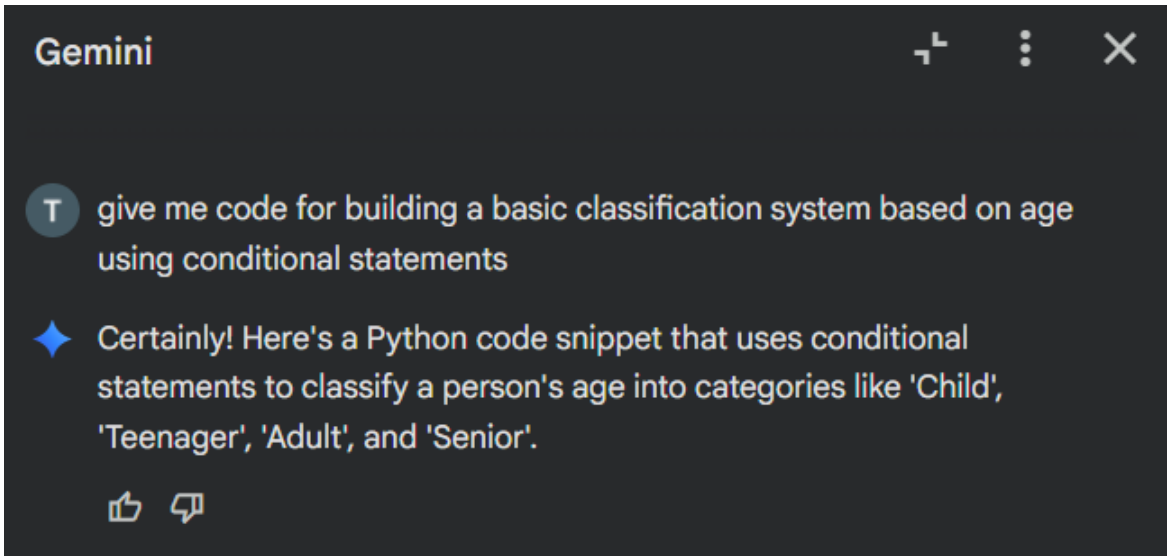
- multiple = number * i → calculates the multiple.
- print(multiple, end=" ") → prints the multiple on the same line separated by spaces (because of end=" ").

New Line After Loop

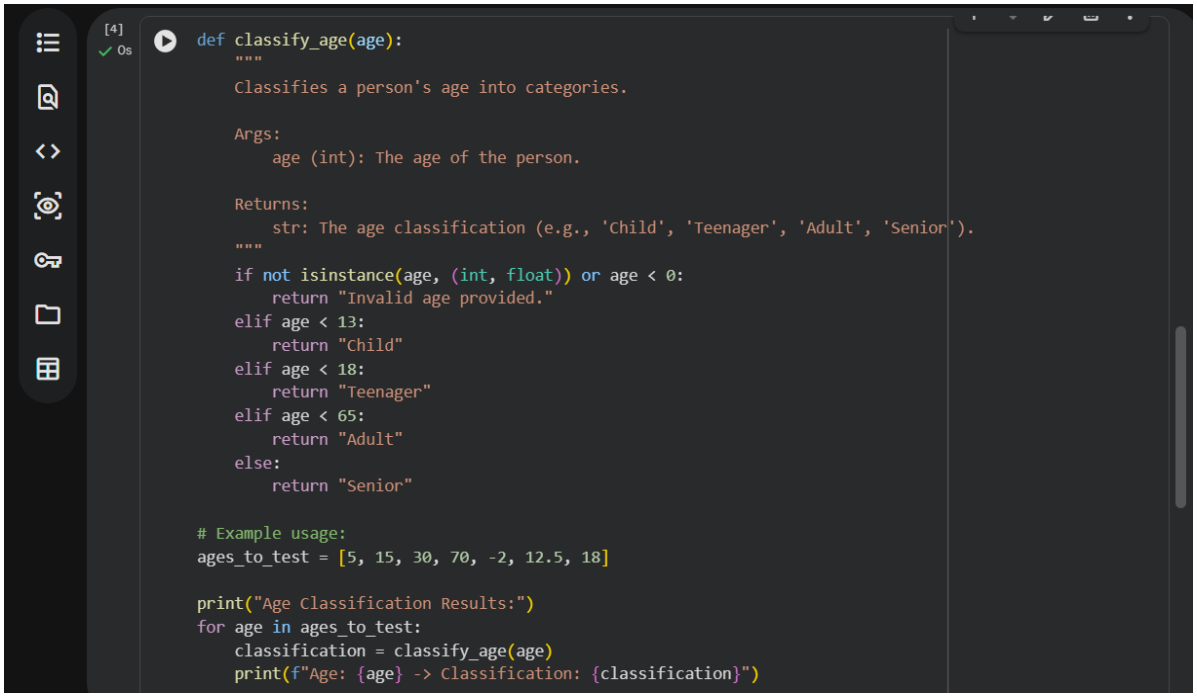
- print() → ensures the output moves to a new line after all multiples are printed.

Task Description #3: Conditional Statements (Age Classification)

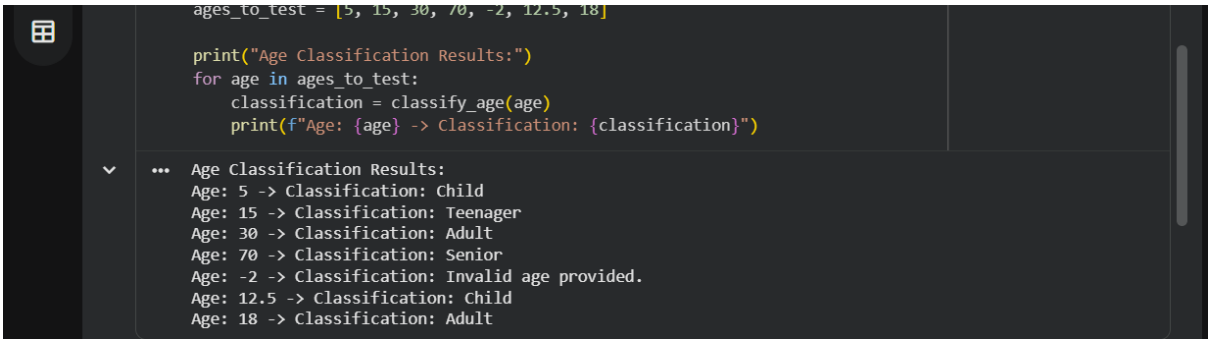
Prompt:



Code:



Output:



```
ages_to_test = [5, 15, 30, 70, -2, 12.5, 18]

print("Age Classification Results:")
for age in ages_to_test:
    classification = classify_age(age)
    print(f"Age: {age} -> Classification: {classification}")
```

```
Age Classification Results:
Age: 5 -> Classification: Child
Age: 15 -> Classification: Teenager
Age: 30 -> Classification: Adult
Age: 70 -> Classification: Senior
Age: -2 -> Classification: Invalid age provided.
Age: 12.5 -> Classification: Child
Age: 18 -> Classification: Adult
```

Code Explanation:

Function Definition

- def classify_age(age):
- Defines a function named classify_age.
- Takes one parameter: age (expected to be an integer or float).

Input Validation

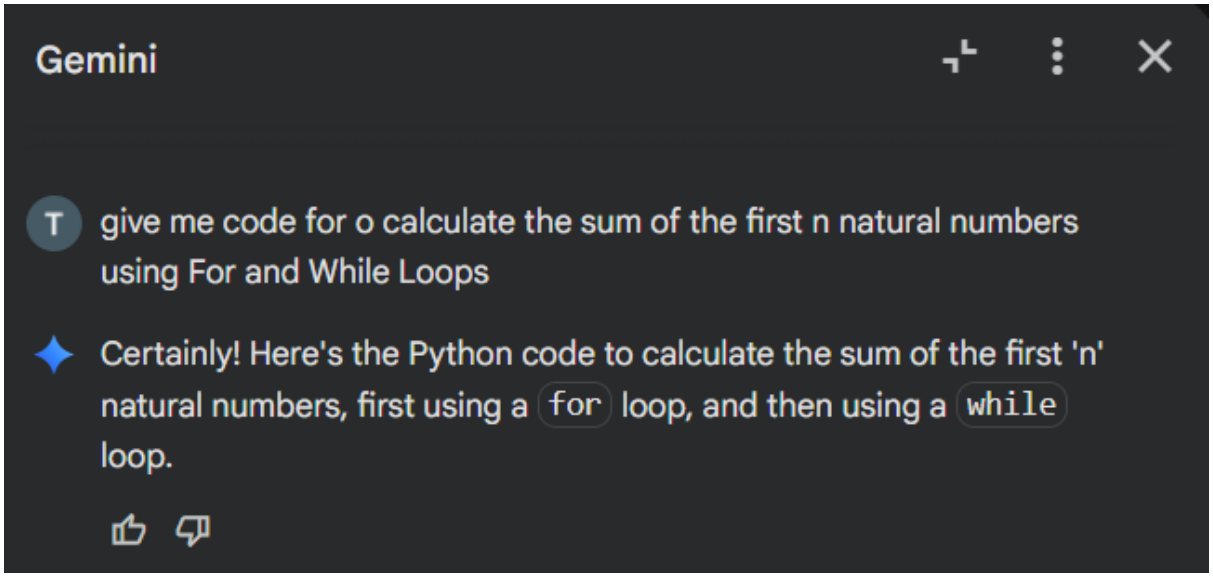
- if not isinstance(age, (int, float)) or age < 0:
- Checks if age is not a number (int/float) OR is negative.
- If true → returns "Invalid age provided.".

Age Classification Logic

- elif age < 13: → Returns "Child" if age is less than 13.
- elif age < 18: → Returns "Teenager" if age is between 13 and 17.
- elif age < 65: → Returns "Adult" if age is between 18 and 64.
- else: → Returns "Senior" if age is 65 or above.

Task Description #4: For and While Loops (Sum of First n Numbers)

Prompt:



Code:(For loop)

```
[5]
✓ 0s
def sum_natural_numbers_for(n):
    """
    Calculates the sum of the first n natural numbers using a for loop.

    Args:
        n (int): The number of natural numbers to sum.

    Returns:
        int: The sum of the first n natural numbers.
    """
    if n < 0:
        return "Input must be a non-negative integer."
    total_sum = 0
    for i in range(1, n + 1):
        total_sum += i
    return total_sum

# Example usage for 'for' loop:
n_for = 10
sum_for = sum_natural_numbers_for(n_for)
print(f"Sum of the first {n_for} natural numbers (using for loop): {sum_for}")

n_for_negative = -5
sum_for_negative = sum_natural_numbers_for(n_for_negative)
print(f"Sum for {n_for_negative} (using for loop): {sum_for_negative}")
```

Output:

```
n_for_negative = -5
sum_for_negative = sum_natural_numbers_for(n_for_negative)
print(f"Sum for {n_for_negative} (using for loop): {sum_for_negative}")

Sum of the first 10 natural numbers (using for loop): 55
Sum for -5 (using for loop): Input must be a non-negative integer.
```

Code:(While loop)

```
[6]
✓ 0s
def sum_natural_numbers_while(n):
    """
    Calculates the sum of the first n natural numbers using a while loop.

    Args:
        n (int): The number of natural numbers to sum.

    Returns:
        int: The sum of the first n natural numbers.
    """
    if n < 0:
        return "Input must be a non-negative integer."
    total_sum = 0
    counter = 1
    while counter <= n:
        total_sum += counter
        counter += 1
    return total_sum

# Example usage for 'while' loop:
n_while = 7
sum_while = sum_natural_numbers_while(n_while)
print(f"Sum of the first {n_while} natural numbers (using while loop): {sum_while}")

n_while_zero = 0
sum_while_zero = sum_natural_numbers_while(n_while_zero)
print(f"Sum for {n_while_zero} (using while loop): {sum_while_zero}")
```

Output:

```
n_while_zero = 0
sum_while_zero = sum_natural_numbers_while(n_while_zero)
print(f"Sum for {n_while_zero} (using while loop): {sum_while_zero}")

...
Sum of the first 7 natural numbers (using while loop): 28
Sum for 0 (using while loop): 0
```

Code Explanation:

Function 1: sum_natural_numbers_for(n)

- Purpose: Calculates the sum of the first n natural numbers using a for loop.
- Steps:
    1. Input validation:
        - If n < 0, returns "Input must be a non-negative integer.".
    2. Initialize total_sum = 0.
    3. Loop:
        - for i in range(1, n + 1): → iterates from 1 to n.
        - Adds each number to total_sum.
    4. Returns the final sum.

Example usage:

- n_for = 10 → Sum = 55.

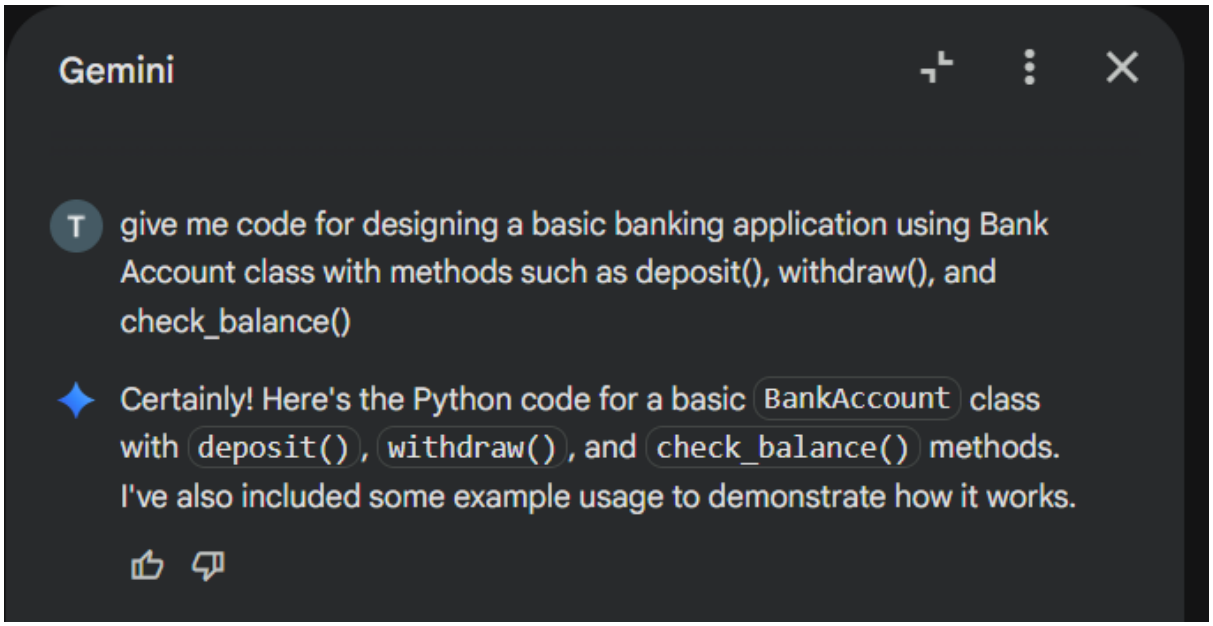
- n_for_negative = -5 → Returns "Input must be a non-negative integer.".

🔷Function 2: sum_natural_numbers_while(n)

- Purpose: Calculates the sum of the first n natural numbers using a while loop.
- Steps:
    1. Input validation:
        - If n < 0, returns "Input must be a non-negative integer.".
    2. Initialize total_sum = 0 and counter = 1.
    3. Loop:
        - while counter <= n: → runs until counter exceeds n.
        - Adds counter to total_sum.
        - Increments counter by 1.

Task Description #5: Classes (Bank Account Class)

Prompt:



Code:

```
[7]
✓ 0s
    def __init__(self, account_holder_name, initial_balance=0):
        if initial_balance < 0:
            raise ValueError("Initial balance cannot be negative.")
        self.account_holder_name = account_holder_name
        self.balance = initial_balance
        print(f"Account created for {self.account_holder_name} with initial balance: ${self.balance:.2f

    def deposit(self, amount):
        if amount <= 0:
            print("Deposit amount must be positive.")
            return False
        self.balance += amount
        print(f"Deposited: ${amount:.2f}. New balance: ${self.balance:.2f}")
        return True

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

    def check_balance(self):
        print(f"Account Balance for {self.account_holder_name}: ${self.balance:.2f}")
        return self.balance
```

Output:

```
[7]
✓ 0s
print("\n--- Transactions for Jane Smith ---")
account2.check_balance()
account2.deposit(750)
account2.withdraw(300)
account2.check_balance()
```

```
--- Creating Accounts ---
Account created for John Doe with initial balance: $1000.00
Account created for Jane Smith with initial balance: $0.00

--- Transactions for John Doe ---
Account Balance for John Doe: $1000.00
Deposited: $500.00. New balance: $1500.00
Withdrew: $200.00. New balance: $1300.00
Insufficient funds.
Deposit amount must be positive.
Account Balance for John Doe: $1300.00

--- Transactions for Jane Smith ---
Account Balance for Jane Smith: $0.00
Deposited: $750.00. New balance: $750.00
Withdrew: $300.00. New balance: $450.00
Account Balance for Jane Smith: $450.00
450
```

Code Explanation:

Class Definition

- class BankAccount: → Defines a BankAccount class to simulate basic banking operations.

Constructor (__init__)

- def __init__(self, account_holder_name, initial_balance=0):
- Initializes a new account with:
    - account_holder_name → Name of the account holder.
    - initial_balance → Starting balance (default = 0).
- Validation:
    - If initial_balance < 0, raises a ValueError.
- Sets attributes:
    - self.account_holder_name = account_holder_name
    - self.balance = initial_balance
- Prints confirmation message with formatted balance.

Deposit Method

- def deposit(self, amount):
- Validates deposit:
    - If amount <= 0, prints error and returns False.
- Otherwise:
- Adds amount to self.balance.
- Prints deposit confirmation and new balance.
- Returns True.

Withdraw Method

- def withdraw(self, amount):
- Validates withdrawal:
    - If amount <= 0, prints error and returns False.
    - If amount > self.balance, prints "Insufficient funds" and returns False.
- Otherwise:
- Deducts amount from self.balance.
- Prints withdrawal confirmation and new balance.
- Returns True.

Check Balance Method

- def check_balance(self): Prints current balance for the account holder.