Assignment-6

2303A51018

Batch-28

Task Description #1 (AI-Based Code Completion for Loops)

Task: Use an AI code completion tool to generate a loop-based

Prompt

Wite a pyhon code to print all even numbers between 1 and N

using a loop.”

Expected Output:

- AI-generated loop logic.
- Identification of loop type used (for or while).
- Validation with sample inputs.

Prompt

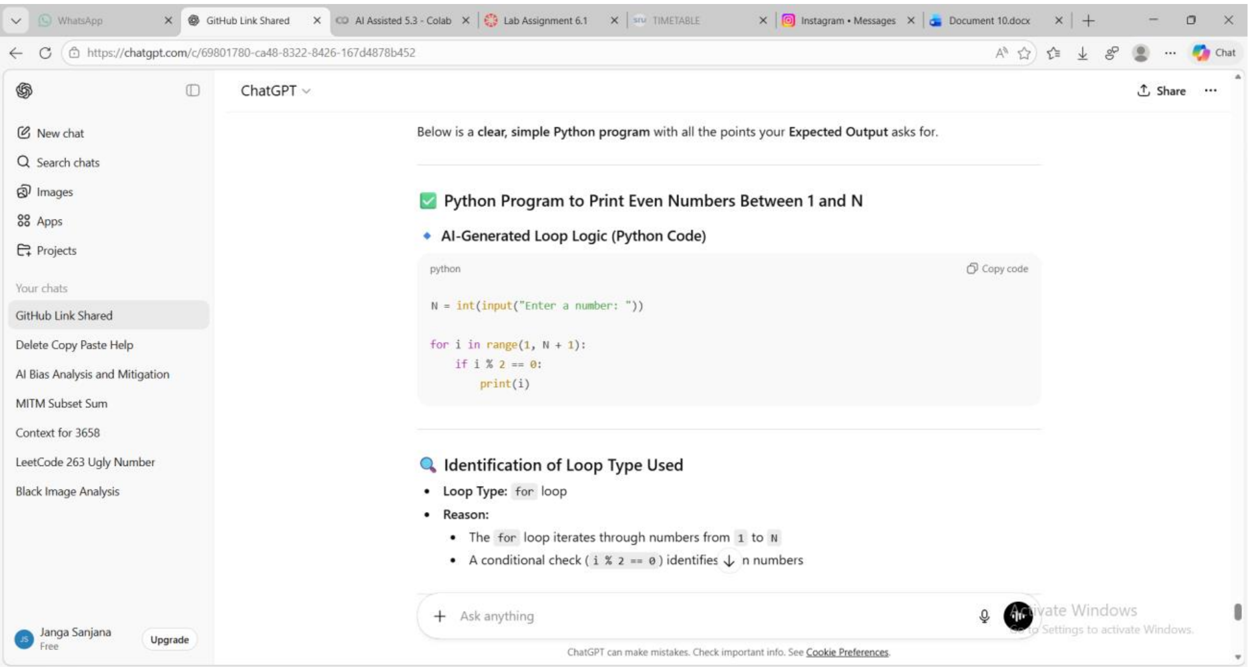
“Generate Python code to print all even numbers between 1 and N

using a loop.”

Expected Output:

- AI-generated loop logic.
- Identification of loop type used (for or while).
- Validation with sample inputs.

Code



Below is a **clear, simple Python program** with all the points your **Expected Output** asks for.

✅ Python Program to Print Even Numbers Between 1 and N

◆ AI-Generated Loop Logic (Python Code)

python

```python
N = int(input("Enter a number: "))

for i in range(1, N + 1):
    if i % 2 == 0:
        print(i)
```

🔍 Identification of Loop Type Used

- **Loop Type:** `for` loop
- **Reason:**
  - The `for` loop iterates through numbers from `1` to `N`
  - A conditional check (`i % 2 == 0`) identifies ↓ n numbers

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

Output

📌

```
2
4
6
```

Explanation

Identification of Loop Type Used

- **Loop Type:** `for` loop
- **Reason:**
  - The `for` loop iterates through numbers from 1 to N
  - A conditional check (`i % 2 == 0`) identifies even numbers

Conclusion

- The program correctly prints all even numbers between 1 and N
- A **for loop** is used to implement the logic
- Output is validated using different input values

Task Description #2 (AI-Based Code Completion for Loop with

Conditionals)

Task: Use an AI code completion tool to combine loops and

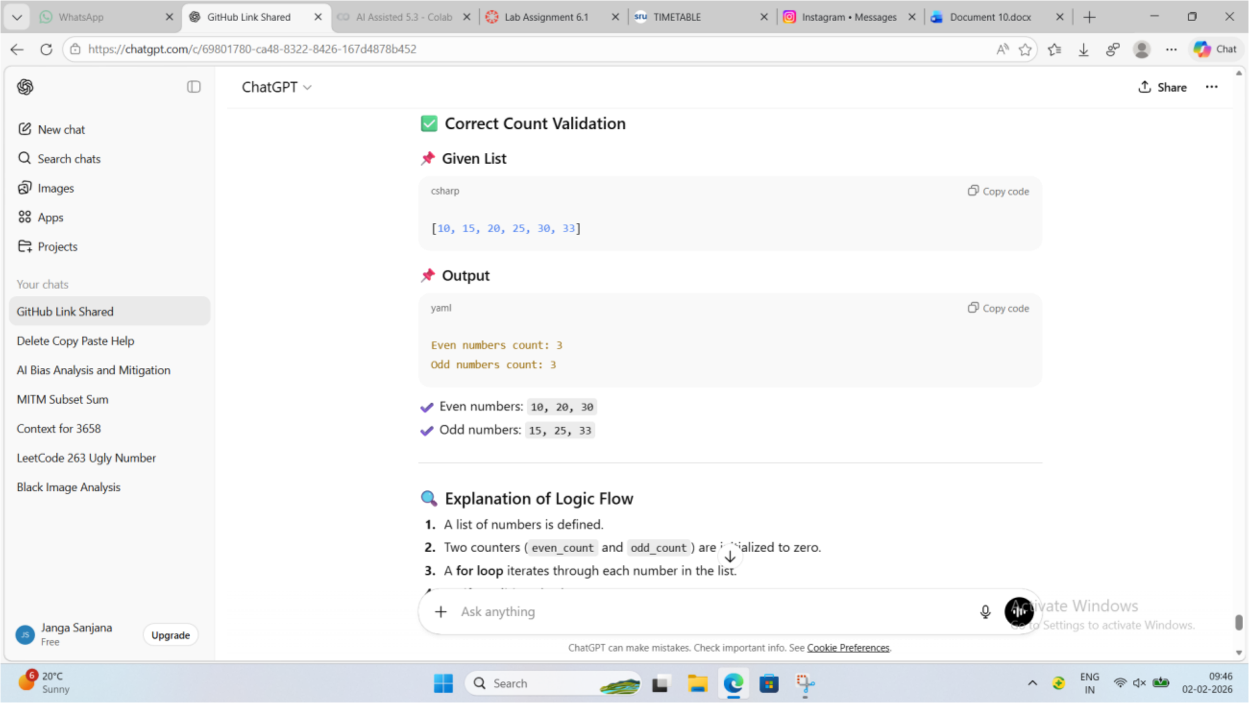
Conditionals.

Prompt

AI-Generated Code (Using Loop and `if` Condition)

```
numbers = [10, 15, 20, 25, 30, 33]

even_count = 0
odd_count = 0

for num in numbers:
    if num % 2 == 0:
        even_count += 1
    else:
        odd_count += 1

print("Even numbers count:", even_count)
print("Odd numbers count:", odd_count)
```

Code



✅ **Correct Count Validation**

📌 **Given List**

```csharp
[10, 15, 20, 25, 30, 33]
```

📌 **Output**

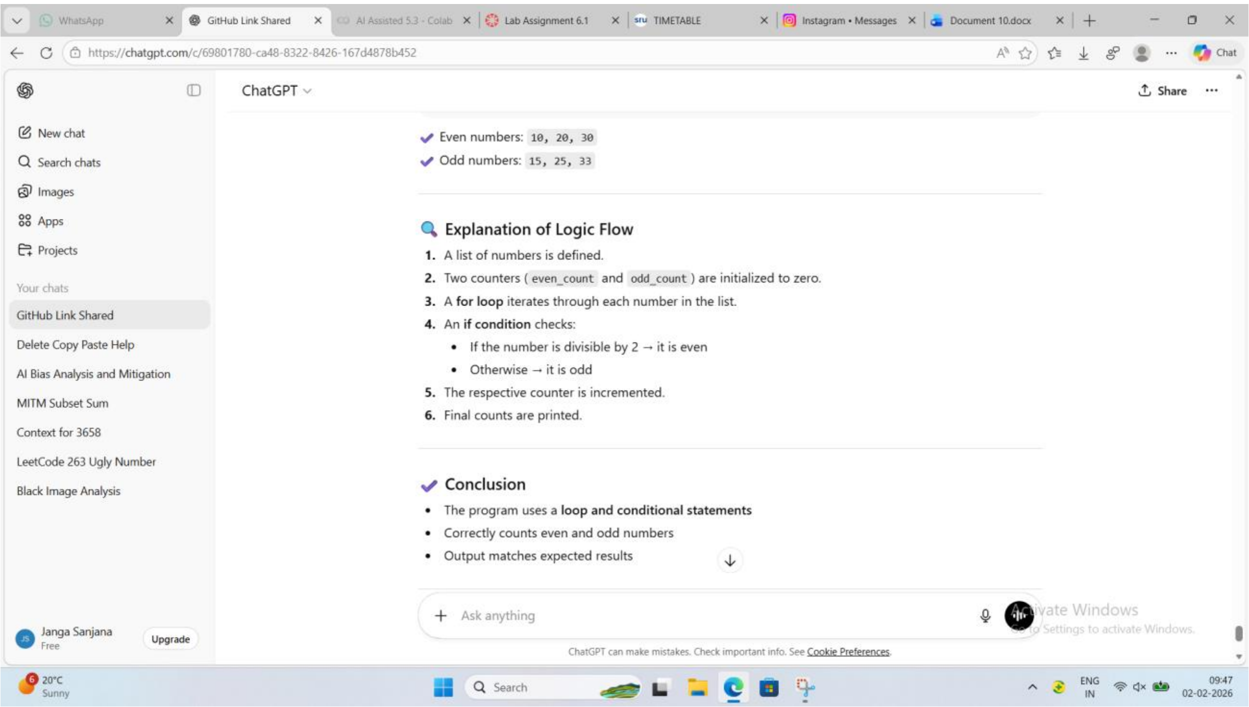
```yaml
Even numbers count: 3
Odd numbers count: 3
```

✔ Even numbers: `10, 20, 30`

✔ Odd numbers: `15, 25, 33`

🔍 **Explanation of Logic Flow**

1. A list of numbers is defined.
2. Two counters (`even_count` and `odd_count`) are initialized to zero.
3. A **for loop** iterates through each number in the list.



✔ Even numbers: `10, 20, 30`

✔ Odd numbers: `15, 25, 33`

🔍 **Explanation of Logic Flow**

1. A list of numbers is defined.
2. Two counters (`even_count` and `odd_count`) are initialized to zero.
3. A **for loop** iterates through each number in the list.

4. An **if condition** checks:
   - If the number is divisible by 2 → it is even
   - Otherwise → it is odd
5. The respective counter is incremented.
6. Final counts are printed.

✔ **Conclusion**

- The program uses a **loop and conditional statements**
- Correctly counts even and odd numbers
- Output matches expected results

Task Description #3 (AI-Based Code Completion for Class

Attributes Validation)

Task: Use an AI tool to complete a Python class that validates user input

Prompt

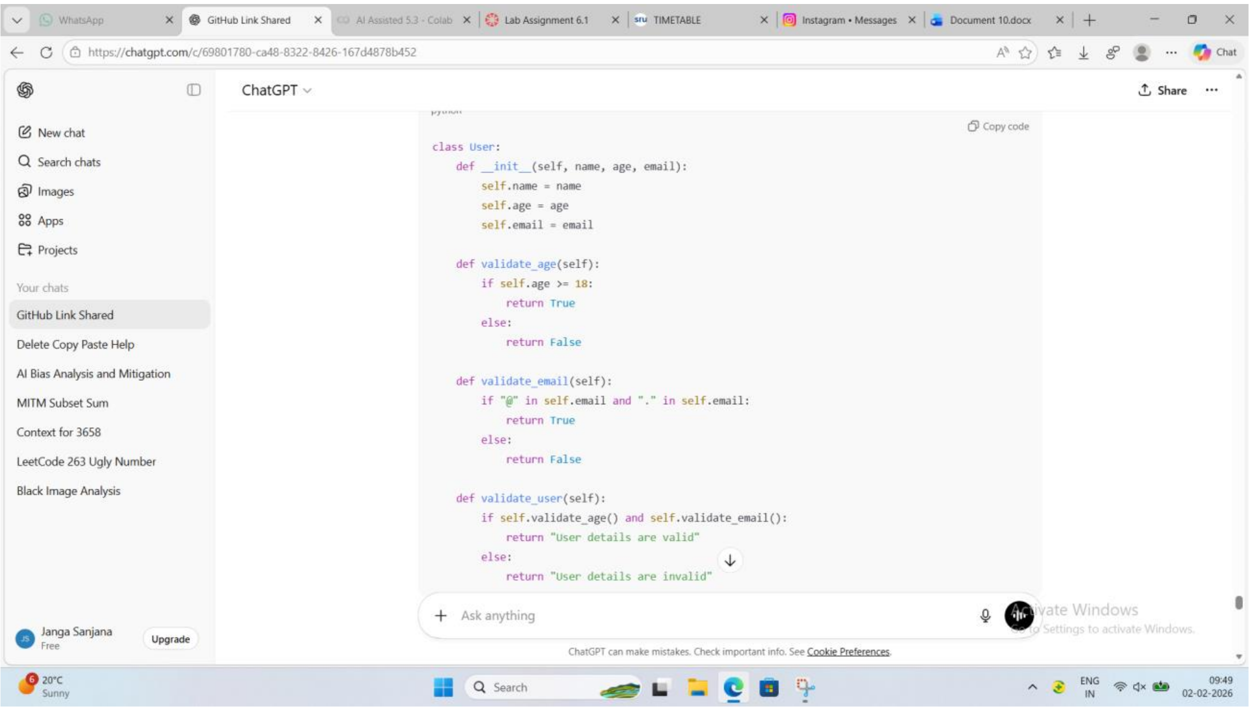
“Generate a Python class User that validates age and email using

conditional statements.”

Expected Output:

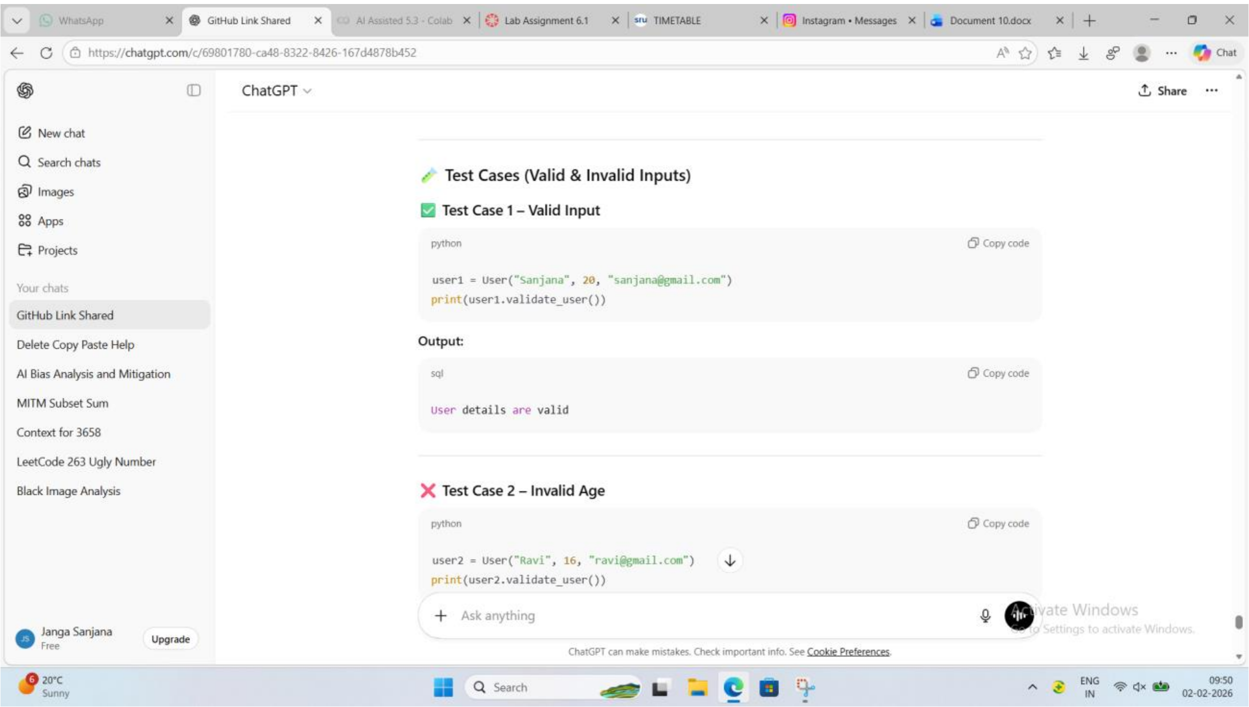
- AI-generated class with validation logic.
- Verification of condition handling.
- Test cases for valid and invalid inputs.

Code



```python
class User:
    def __init__(self, name, age, email):
        self.name = name
        self.age = age
        self.email = email

    def validate_age(self):
        if self.age >= 18:
            return True
        else:
            return False

    def validate_email(self):
        if "@" in self.email and "." in self.email:
            return True
        else:
            return False

    def validate_user(self):
        if self.validate_age() and self.validate_email():
            return "User details are valid"
        else:
            return "User details are invalid"
```

Output



🧪 Test Cases (Valid & Invalid Inputs)

✅ Test Case 1 – Valid Input

```python
user1 = User("Sanjana", 20, "sanjana@gmail.com")
print(user1.validate_user())
```

Output:

```sql
User details are valid
```

❌ Test Case 2 – Invalid Age

```python
user2 = User("Ravi", 16, "ravi@gmail.com")
print(user2.validate_user())
```

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

Explanation

Verification of Condition Handling

✔ Age Validation

- Uses an **if–else condition**
- Accepts users aged **18 or above**
- Returns `True` for valid age, otherwise `False`

✔ Email Validation

- Checks for **`@` and `.`** in the email string
- Ensures basic email format validity
- Returns `True` for valid email, otherwise `False`

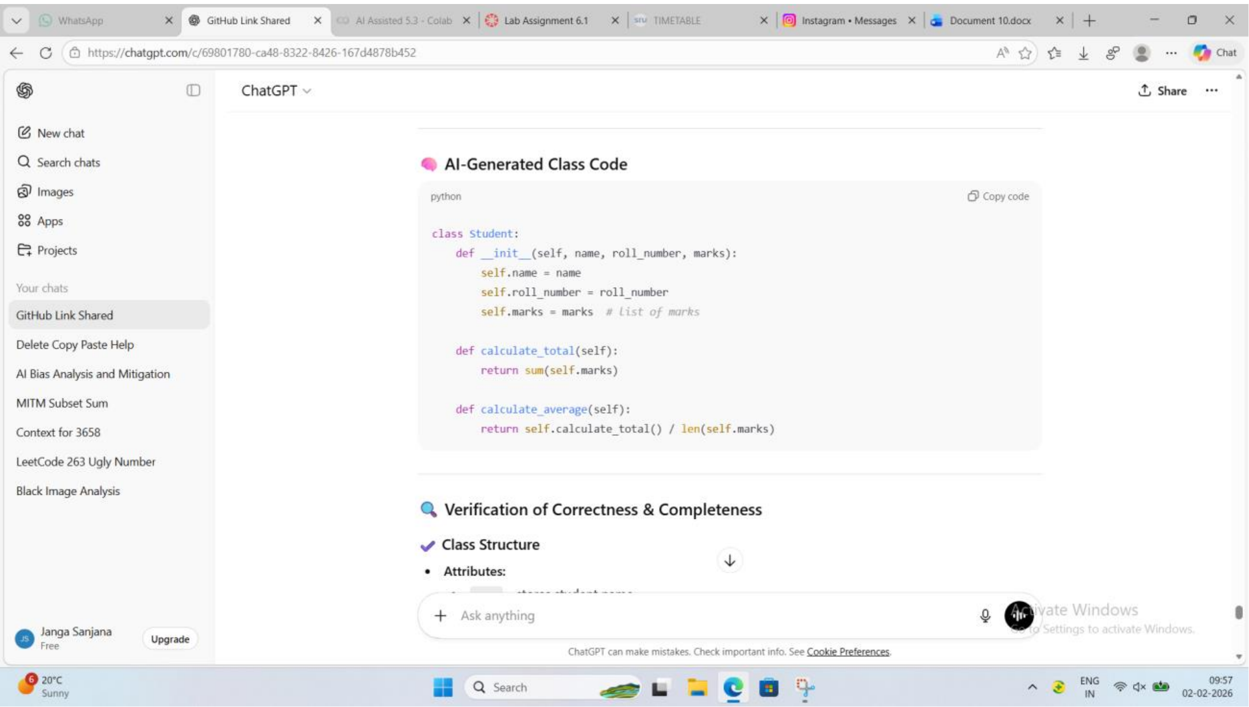
✔ Combined Validation

- Both conditions must be satisfied for the user to be valid

Conclusion

- The AI-generated `User` class correctly uses **conditional statements**
- Age and email validations work as expected
- Test cases confirm proper handling of valid and invalid inputs

Task Description #4 (AI-Based Code Completion for Classes)
Task: Use an AI code completion tool to generate a Python class for managing student details.
Prompt
“Generate a Python class Student with attributes (name, roll number, marks) and methods to calculate total and average marks.”
Expected Output:
• AI-generated class code.
• Verification of correctness and completeness of class structure.
• Minor manual improvements (if needed) with justification.
Code



AI-Generated Class Code

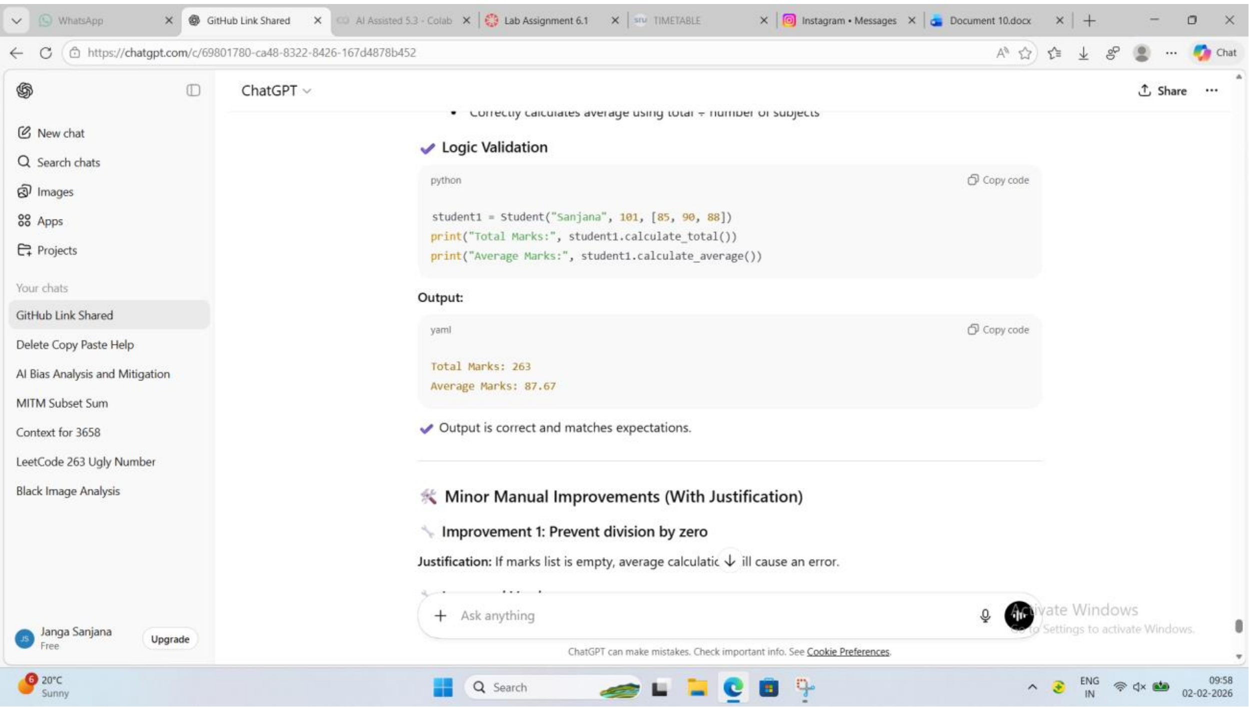
```python
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks  # list of marks

    def calculate_total(self):
        return sum(self.marks)

    def calculate_average(self):
        return self.calculate_total() / len(self.marks)
```

Verification of Correctness & Completeness

✓ Class Structure

- Attributes:

ChatGPT can make mistakes. Check important info. See Cookie Preferences.



- Correctly calculates average using total ÷ number of subjects

✓ Logic Validation

```python
student1 = Student("Sanjana", 101, [85, 90, 88])
print("Total Marks:", student1.calculate_total())
print("Average Marks:", student1.calculate_average())
```

Output:

```yaml
Total Marks: 263
Average Marks: 87.67
```

✓ Output is correct and matches expectations.

Minor Manual Improvements (With Justification)

Improvement 1: Prevent division by zero

Justification: If marks list is empty, average calculatic ill cause an error.

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

Verification of Correctness & Completeness

✓ Class Structure

- **Attributes:**

    - `name` – stores student name
    - `roll_number` – stores roll number
    - `marks` – list of subject marks

✓ Methods

- `calculate_total()`
    - Correctly calculates total marks using `sum()`
- `calculate_average()`
    - Correctly calculates average using total ÷ number of subjects

Conclusion

- AI successfully generated a complete `Student` class
- Total and average calculations are correct
- Minor improvement improves reliability and real-world usability

  Task Description 5 (AI-Assisted Code Completion Review)
  Task: Use an AI tool to generate a complete Python program using classes, loops, and conditionals together.
  Prompt
  “Generate a Python program for a simple bank account system using class, loops, and conditional statements.”
  Expected Output:
  • Complete AI-generated program.
  • Identification of strengths and limitations of AI suggestions.
  • Reflection on how AI assisted coding productivity.

  Code

class BankAccount: def **init**(self, owner, balance=0): self.owner = owner self.balance = 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 amount <= 0:
        print("Withdrawal amount must be positive.")
    elif amount > self.balance:
        print("Insufficient balance!")
    else:
        self.balance -= amount
        print(f"Withdrew ${amount}. New balance: ${self.balance}")

def check_balance(self):
    print(f"{self.owner}, your current balance is: ${self.balance}")
```

Main program loop

def main(): print("Welcome to the Python Bank System!") name = input("Enter your name: ") account = BankAccount(name)

```
while True:
    print("\nSelect an option:")
    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 amount to deposit: "))
        account.deposit(amount)
    elif choice == "2":
        amount = float(input("Enter amount to withdraw: "))
        account.withdraw(amount)
    elif choice == "3":
        account.check_balance()
    elif choice == "4":
        print("Thank you for using Python Bank. Goodbye!")
        break
    else:
        print("Invalid choice. Please select 1-4.")
```

if **name** == "**main**": main()

Output

```
Welcome to the Python Bank System!
Enter your name: Alice

Select an option:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter choice (1-4): 3
Alice, your current balance is: $0

Select an option:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter choice (1-4): 1
Enter amount to deposit: 500
Deposited $500. New balance: $500

Select an option:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter choice (1-4): 2
Enter amount to withdraw: 200
Withdrew $200. New balance: $300
```

```
Select an option:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter choice (1-4): 2
Enter amount to withdraw: 500
Insufficient balance!

Select an option:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter choice (1-4): 3
Alice, your current balance is: $300

Select an option:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter choice (1-4): 4
Thank you for using Python Bank. Goodbye!
```

Explanation of Output

1. Balance starts at 0.
2. Deposit of $500 updates balance to $500.
3. Withdrawal of $200 reduces balance to $300.
4. Withdrawal attempt of $500 fails due to insufficient funds.
5. Checking balance shows $300.
6. Exit ends the program.

Limitations / Areas for Improvement

1. **No persistence**: Account balance resets each time the program runs; real banking systems would store data.

2. **Single-user limitation**: Only supports one account at a time; multi-account management is not implemented.
3. **Minimal input validation**: Non-numeric inputs for amounts may crash the program.