Name:balaji

Rollno:2403A51102

Batch:06

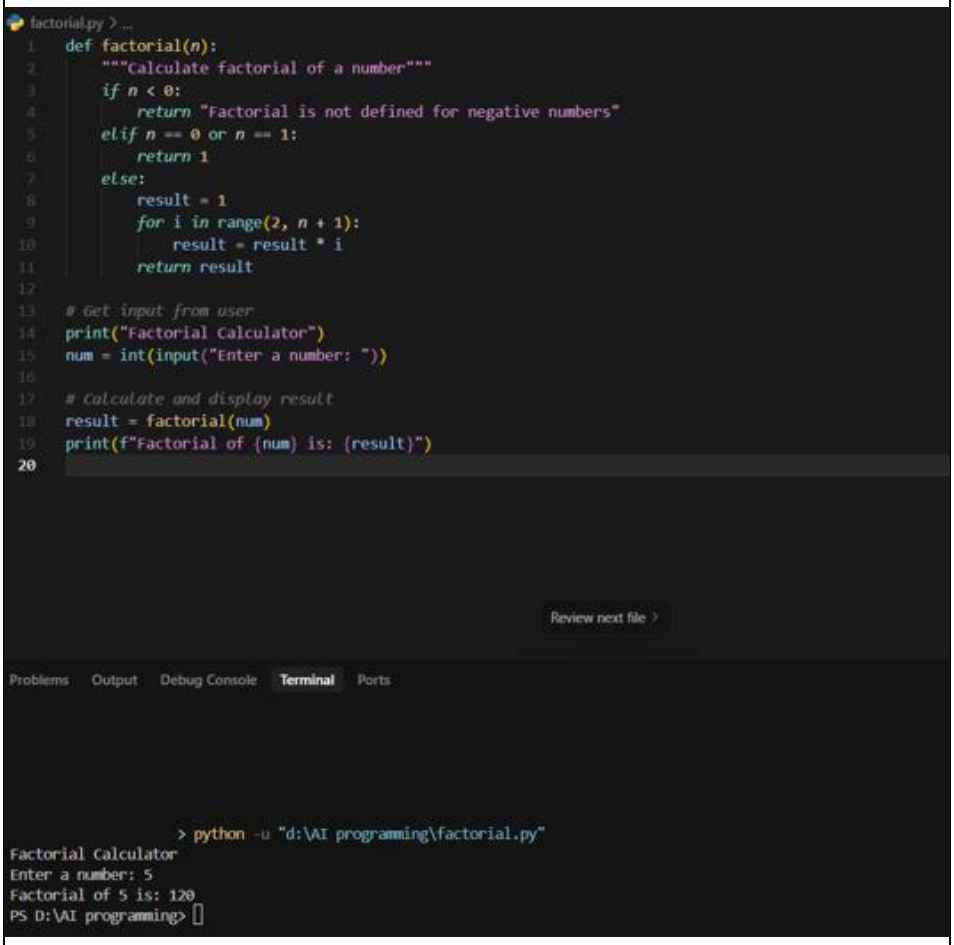
lab Assingment 3.3

| Q.No. | Question | *Expected Time to complete* |
|---|---|---|
| 1 | Lab 3: Prompt Engineering – Improving Prompts and Context Management<br><br>**Lab Objectives:**<br><br>• To understand how prompt structure and wording influence AI-generated code. | 03.08.2025 EOD |

| | | |
|---|---|---|
| | • To explore how context (like comments and function names) helps AI generate relevant output.<br>• To evaluate the quality and accuracy of code based on prompt clarity.<br>• To develop effective prompting strategies for AI-assisted programming.<br><br>**Lab Outcomes (LOs):**<br>After completing this lab, students will be able to:<br><br>• Generate Python code using Google Gemini in Google Colab.<br>• Analyze the effectiveness of code explanations and suggestions by Gemini. | |

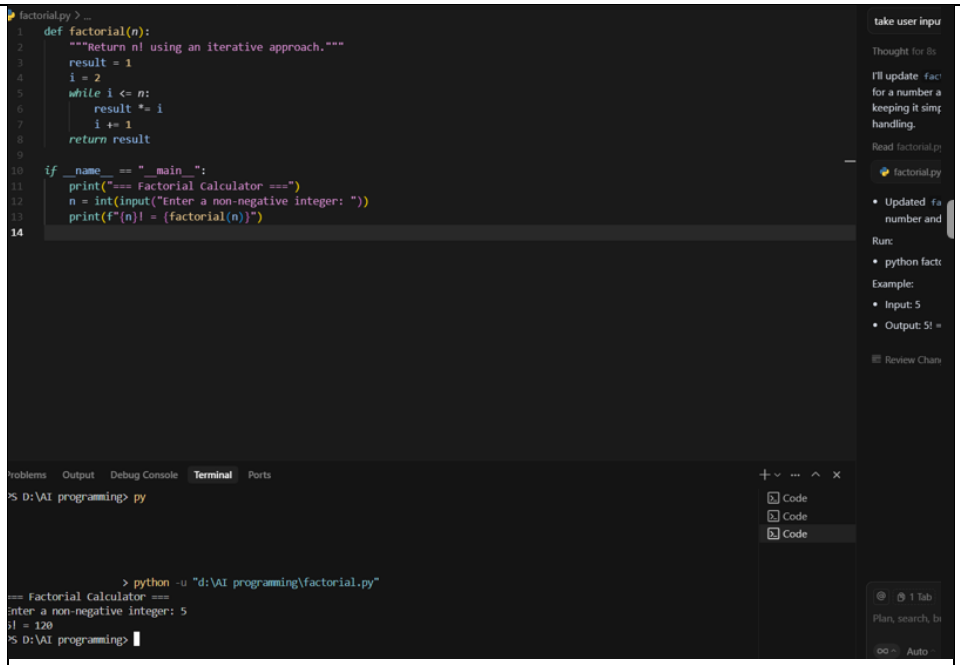
| | | |
|---|---|---|
| | • Set up and use Cursor AI for AI-powered coding assistance.<br>• Evaluate and refactor code using Cursor AI features.<br>• Compare AI tool behavior and code quality across different platforms.<br><br>**Task Description#1**<br>• Try 3 different prompts to generate a factorial function.<br>• Prompts<br>**1.program to calculate the factorial of the given number by user input**<br>**2. Write a Python function factorial(n) that returns n! using an iterative approach** | |

## 3.Create a Python function factorial(n: int) int using recursion.

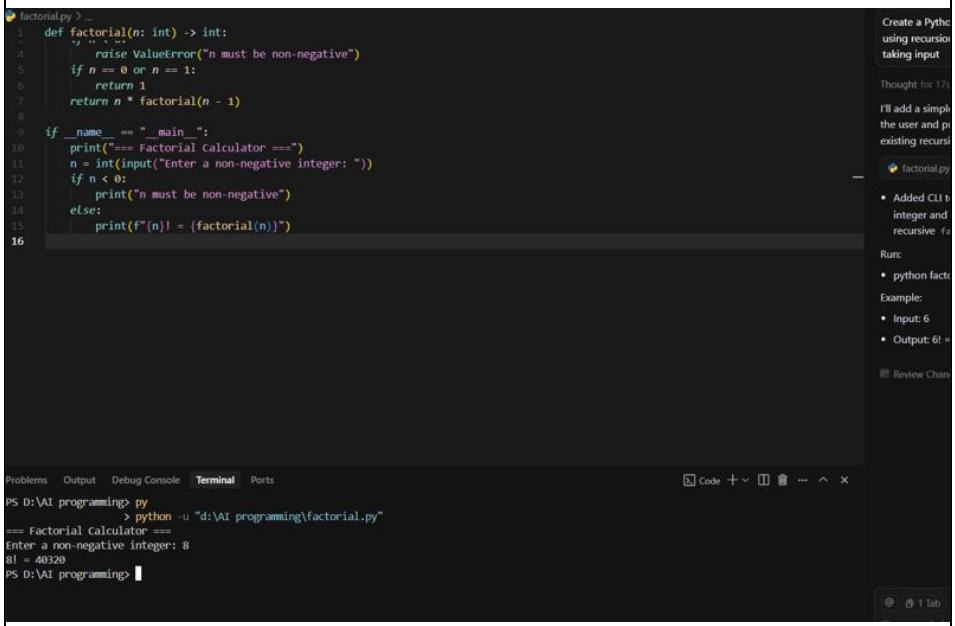



## Prompt1 is very basy and simple

## prompt 2 is best for learning basics — minimal features, focuses on concept clarity.




# Prompt 3 balances learning recursion and safety — more professional but still educational

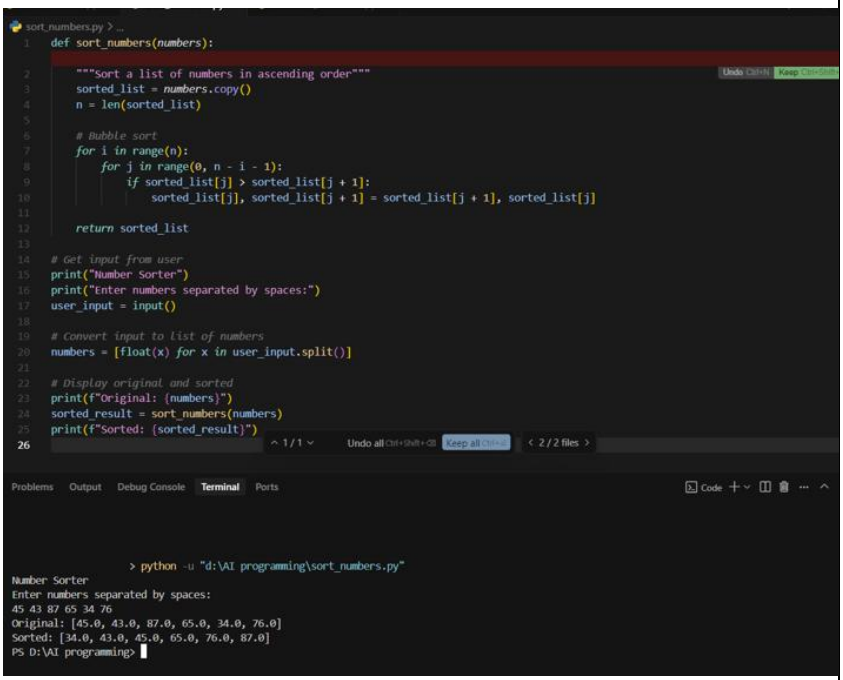


## Expected Output#1

- Comparison of AI-generated code styles

## Task Description#2

- Provide a clear example input-output prompt to generate a sorting function.




## Output#2

```
> python -u "d:\AI programming\sort_numbers.py"
Number Sorter
Enter numbers separated by spaces:
45 43 87 65 34 76
Original: [45.0, 43.0, 87.0, 65.0, 34.0, 76.0]
Sorted: [34.0, 43.0, 45.0, 65.0, 76.0, 87.0]
PS D:\AI programming>
```

- Functional sorting code from AI

## Task Description#3

- Start with the vague prompt "Generate python code to calculate power bill" and improve it step-by-step

power_bill_calculator.py > calculate_power_bill_slab

```python
def calculate_power_bill(units_consumed, rate_per_unit):
    total_bill = units_consumed * rate_per_unit
    return total_bill

def main():
    print("=== Power Bill Calculator ===")
    print("1. Fixed Rate Billing")
    print("2. Slab-based Billing")

    choice = input("\nSelect billing method (1 or 2): ")

    units = float(input("Enter units consumed (kWh): "))

    if choice == "1":
        rate = float(input("Enter rate per unit (Rs/kWh): "))
        bill = calculate_power_bill(units, rate)
        print(f"\nFixed Rate Billing:")
        print(f"Rate per Unit: Rs. {rate}")
    else:
        bill = calculate_power_bill_slab(units)
        print(f"\nSlab-based Billing:")
        print("Rates: 0-100: Rs.3.50, 101-200: Rs.4.50, 201-300: Rs.5.50, 300+: Rs.6.50")

    # Display result
    print(f"Units Consumed: {units} kWh")
    print(f"Total Bill: Rs. {bill:.2f}")

if __name__ == "__main__":
```

Cancel

Problems Output Debug Console **Terminal** Ports

```
1. Fixed Rate Billing
2. Slab-based Billing

Select billing method (1 or 2): 2
Enter units consumed (kWh): 200

Slab-based Billing:
Rates: 0-100: Rs.3.50, 101-200: Rs.4.50, 201-300: Rs.5.50, 300+: Rs.6.50
Units Consumed: 200.0 kWh
Total Bill: Rs. 900.00
PS D:\AI programming>
```

-

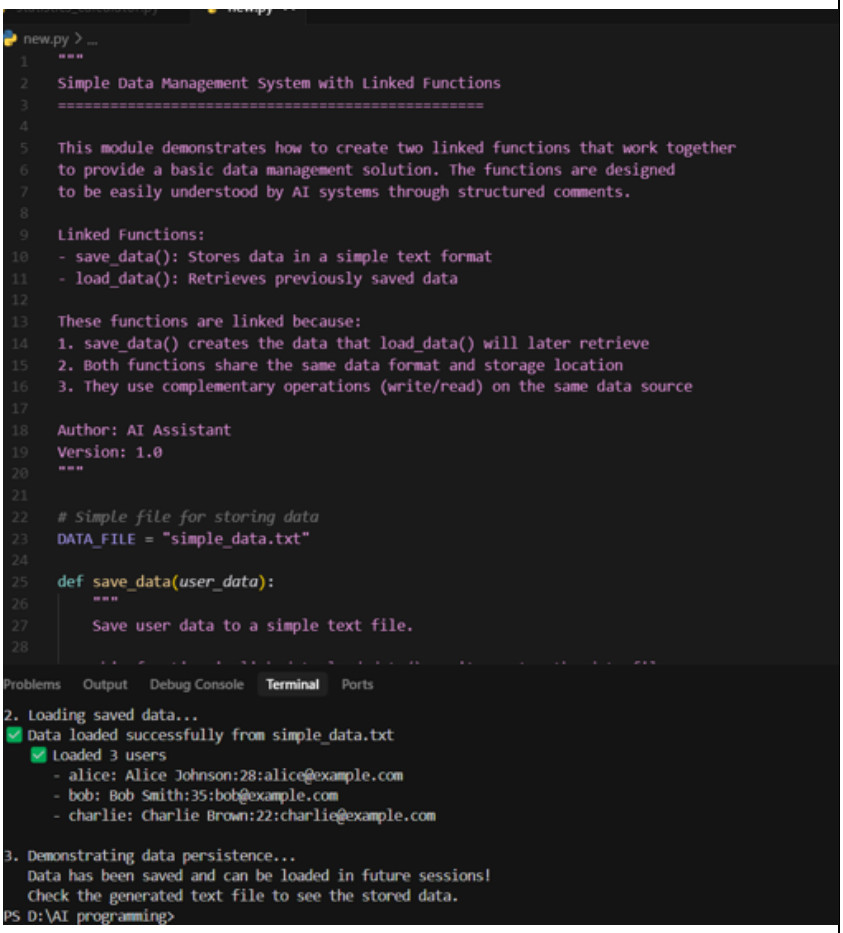
## Expected Output#3

- Enhanced AI output with clearer prompts
-


## Task Description#4

- Write structured comments to help AI generate two linked functions (e.g., login_user() and

register_user()).



```
def save_data(user_data):
    This function is linked to load_data() as it creates the data file
    that load_data() will later read from. The relationship is:
    - save_data() writes data → load_data() reads the same data
    - Both functions use the same file format
    - Data is stored in a simple text format for easy reading

    Args:
        user_data (dict): Dictionary containing user information to save

    Returns:
        bool: True if save operation successful, False otherwise

    Linked Functions:
        - load_data(): Reads the data that this function saves

    Data Format:
        The function saves data in simple text format:
        username:name:age:email
        Each user on a new line

    Example Usage:
        user_info = {"john": "John Doe:25:john@example.com"}
        if save_data(user_info):
            print("Data saved successfully!")
        else:
            print("Failed to save data")
    """
    try:
        # Open file for writing
        with open(DATA_FILE, 'w') as file:
            # Write each user's data on a separate line
            for username, user_info in user_data.items():
                file.write(f"{username}:{user_info}\n")

        print(f"✅ Data saved successfully to {DATA_FILE}")
        return True

    except:
        print("❌ Error saving data")
        return False
```

Snipping Too

| | | |
|---|---|---|
| | | |

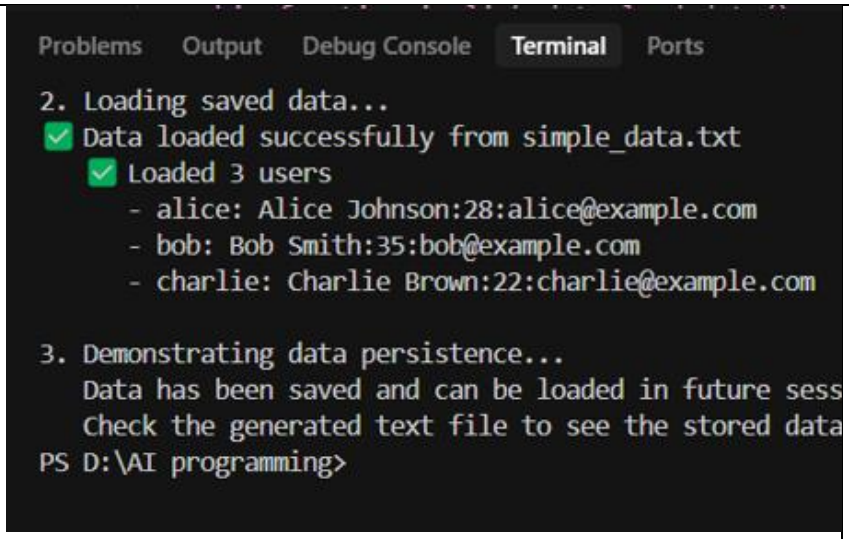
```
def load_data():
                    user_data[username] = user_info

        file.close()
        print(f"✅ Data loaded successfully from {DATA_FILE}")

    except:
        print("📁 No data file found or error reading file")

    return user_data

def main():
    """
    Main function demonstrating the linked functions in action.

    This function shows how save_data() and load_data() work together
    to provide a complete data management solution.

    The demonstration flow:
    1. Create sample user data
    2. Save data using save_data()
    3. Load data using load_data()
    4. Verify data integrity between operations
    """
    print("=== Simple Data Management System Demo ===\n")

    # Sample user data in simple format
    sample_users = {
        "alice": "Alice Johnson:28:alice@example.com",
        "bob": "Bob Smith:35:bob@example.com",
        "charlie": "Charlie Brown:22:charlie@example.com"
    }

    print("1. Saving sample user data...")
    if save_data(sample_users):
        print("   ✅ Data saved successfully")

    print("\n2. Loading saved data...")
    loaded_users = load_data()
    if loaded_users:
        print(f"   ✅ Loaded {len(loaded_users)} users")
        for username, user_info in loaded_users.items():
```

```
122  def main():
141          "charlie": "Charlie Brown:22:charlie@example.com"
142      }
143
144      print("1. Saving sample user data...")
145      if save_data(sample_users):
146          print("   ✅ Data saved successfully")
147
148      print("\n2. Loading saved data...")
149      loaded_users = load_data()
150      if loaded_users:
151          print(f"   ✅ Loaded {len(loaded_users)} users")
152          for username, user_info in loaded_users.items():
153              print(f"      - {username}: {user_info}")
154      else:
155          print("   ❌ No data loaded")
156
157      print("\n3. Demonstrating data persistence...")
158      print("   Data has been saved and can be loaded in future sessions!")
159      print("   Check the generated text file to see the stored data.")
160
161  if __name__ == "__main__":
162      main()
163  Ctrl+L to chat, Ctrl+K to generate
```

```python
def load_data():
    """
    Load user data from the text file.

    This function is linked to save_data() as it reads the data file
    that save_data() previously created. The relationship is:
    - load_data() reads data ← save_data() wrote the data
    - Both functions maintain data format consistency
    - Data is read from the same file that was written to

    Returns:
        dict: Loaded user data dictionary, or empty dict if no data exists

    Linked Functions:
        - save_data(): Provides the data that this function loads

    Data Format:
        The function reads data in the same format that save_data() writes:
        username:name:age:email
        Each user on a new line

    Example Usage:
        user_data = load_data()
        if user_data:
            print(f"Loaded {len(user_data)} users")
        else:
            print("No user data found")
    """
    user_data = {}

    try:
        # Check if file exists
        file = open(DATA_FILE, 'r')

        # Read each line and parse user data
        for line in file:
            line = line.strip()  # Remove extra spaces
            if line:  # Skip empty lines
                parts = line.split(':')
                if len(parts) >= 2:
                    username = parts[0]
                    user_info = ':'.join(parts[1:])  # Join remaining parts
```

## Expected Output#4

- Consistent functions with shared logic

- 


**Task Description#5**

- Analyzing Prompt Specificity: Improving Temperature Conversion Function with Clear

Instructions

```python
def display_menu():
    print("1. Celsius to Fahrenheit")
    print("2. Celsius to Kelvin")
    print("3. Fahrenheit to Celsius")
    print("4. Fahrenheit to Kelvin")
    print("5. Kelvin to Celsius")
    print("6. Kelvin to Fahrenheit")
    print("7. Exit")

def main():
    """Main temperature conversion program"""
    print("Welcome to Temperature Converter!")

    while True:
        display_menu()
        choice = input("\nEnter your choice (1-7): ")

        if choice == '7':
            print("Thank you for using Temperature Converter!")
            break

        if choice in ['1', '2', '3', '4', '5', '6']:
            try:
                # Get temperature input
                temp = float(input("Enter temperature: "))

                # Perform conversion based on choice
                if choice == '1':
```

## Output#5

```
2. Loading saved data...
✅ Data loaded successfully from simple_data.txt
   ✅ Loaded 3 users
      - alice: Alice Johnson:28:alice@example.com
      - bob: Bob Smith:35:bob@example.com
      - charlie: Charlie Brown:22:charlie@example.com

3. Demonstrating data persistence...
   Data has been saved and can be loaded in future sessions!
   Check the generated text file to see the stored data.
PS D:\AI programming>
```

**Note: Report should be submitted a word document for all tasks in a single document with prompts, comments &**

**code explanation, and output and if required, screenshots**

**Evaluation Criteria:**

| **Criteria** | **Max Marks** |
|---|---|
| Factorial Function (Task#1) | 0.5 |
| Sorting Function (Task#2) | 0.5 |
| Vogue Vs. Specific Prompting (Task #3) | 0.5 |
| Linked Functions (Task #4) | 0.5 |
| Temperature Conversion Function (Task #5) | 0.5 |

| | | | | |
|---|---|---|---|---|
| | **Total** | **2.5 Marks** | | |