AI ASSISTED CODING

Lab Assignment 3.3

Roll no: 2403A510F4
Batch: 06
Department: CSE

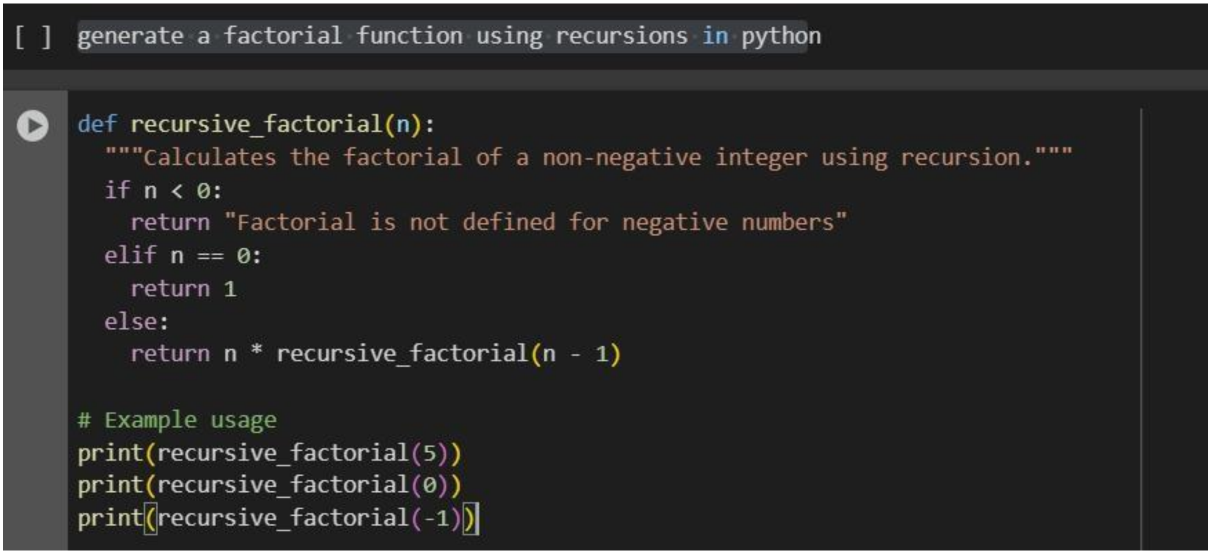
Task Description 1

- Try 3 different prompts to generate a factorial function

Prompt 1:

Generate a factorial function using recursions in python

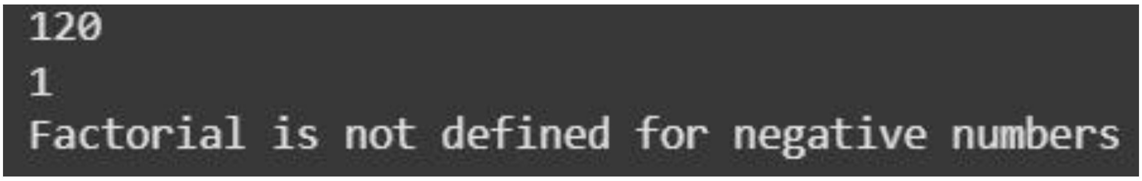
Code using Gemini:



```
generate a factorial function using recursions in python
```

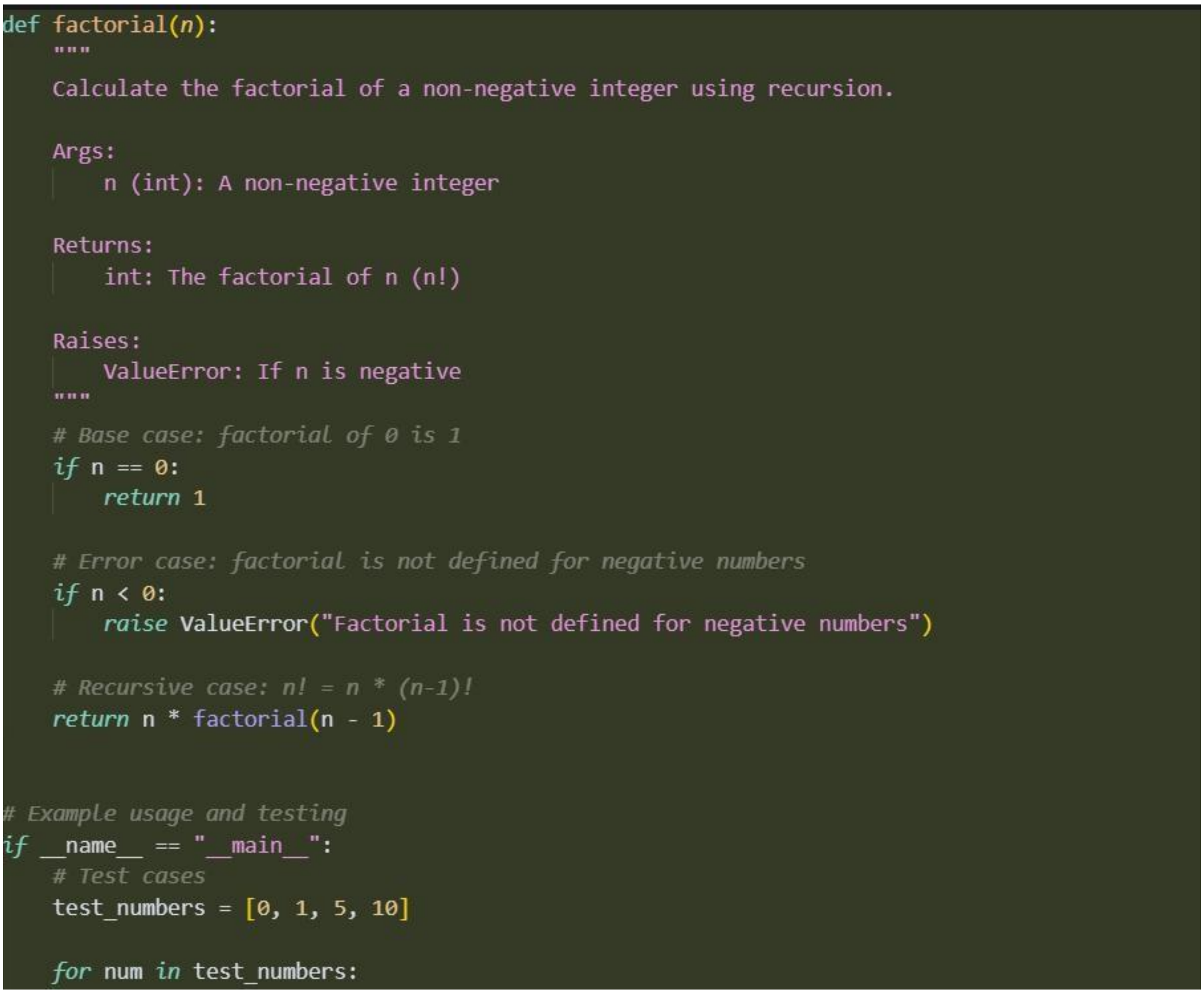
```python
def recursive_factorial(n):
  """Calculates the factorial of a non-negative integer using recursion."""
  if n < 0:
    return "Factorial is not defined for negative numbers"
  elif n == 0:
    return 1
  else:
    return n * recursive_factorial(n - 1)

# Example usage
print(recursive_factorial(5))
print(recursive_factorial(0))
print(recursive_factorial(-1))
```

Output:

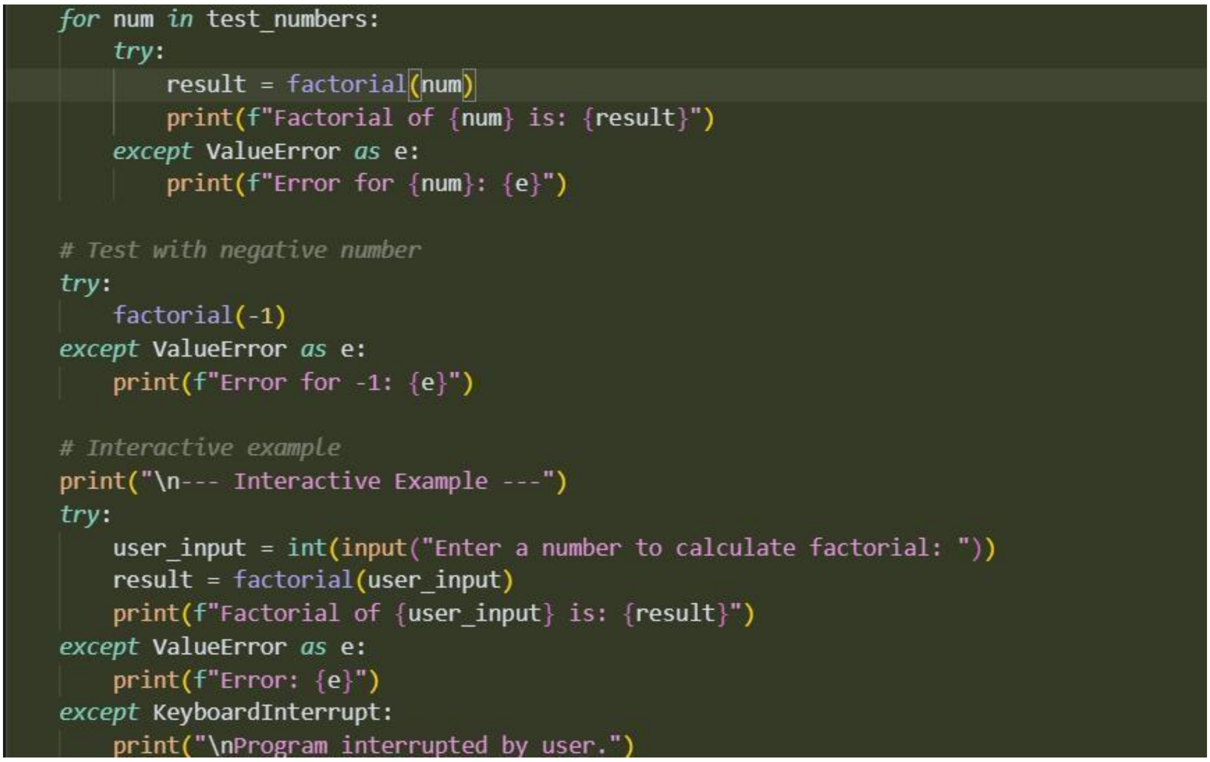


```
120
1
Factorial is not defined for negative numbers
```

Code using cursor ai

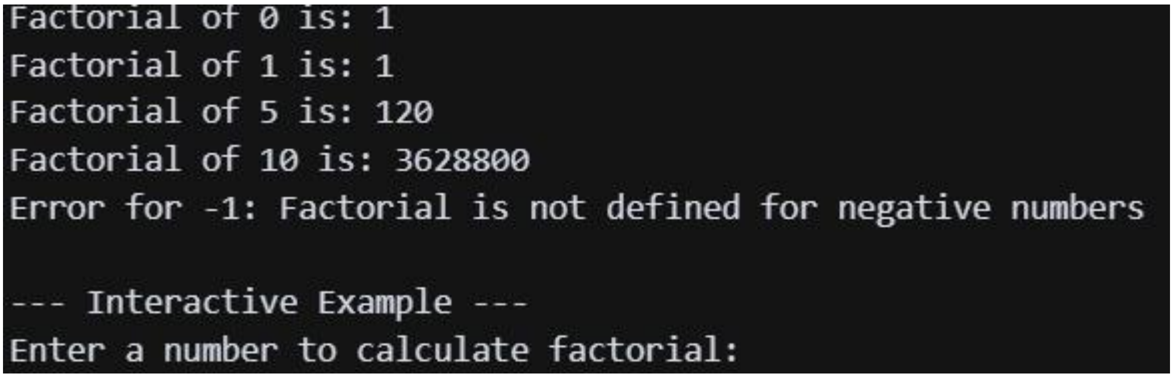


```python
def factorial(n):
    """
    Calculate the factorial of a non-negative integer using recursion.

    Args:
        n (int): A non-negative integer

    Returns:
        int: The factorial of n (n!)

    Raises:
        ValueError: If n is negative
    """
    # Base case: factorial of 0 is 1
    if n == 0:
        return 1

    # Error case: factorial is not defined for negative numbers
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")

    # Recursive case: n! = n * (n-1)!
    return n * factorial(n - 1)


# Example usage and testing
if __name__ == "__main__":
    # Test cases
    test_numbers = [0, 1, 5, 10]

    for num in test_numbers:
```



```python
    for num in test_numbers:
        try:
            result = factorial(num)
            print(f"Factorial of {num} is: {result}")
        except ValueError as e:
            print(f"Error for {num}: {e}")

    # Test with negative number
    try:
        factorial(-1)
    except ValueError as e:
        print(f"Error for -1: {e}")

    # Interactive example
    print("\n--- Interactive Example ---")
    try:
        user_input = int(input("Enter a number to calculate factorial: "))
        result = factorial(user_input)
        print(f"Factorial of {user_input} is: {result}")
    except ValueError as e:
        print(f"Error: {e}")
    except KeyboardInterrupt:
        print("\nProgram interrupted by user.")
```

Output:

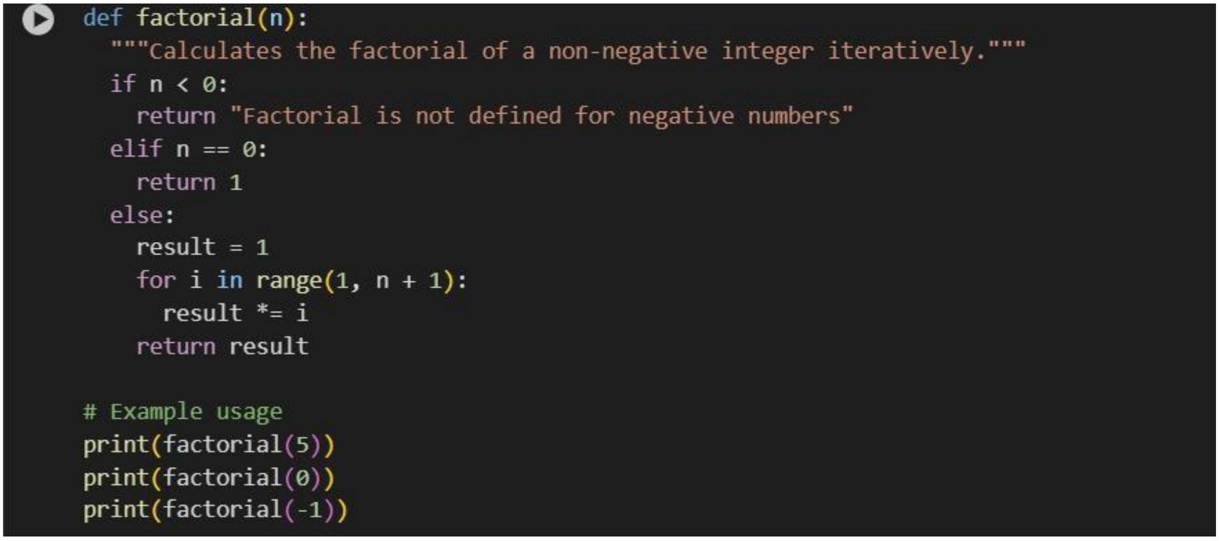


```
Factorial of 0 is: 1
Factorial of 1 is: 1
Factorial of 5 is: 120
Factorial of 10 is: 3628800
Error for -1: Factorial is not defined for negative numbers

--- Interactive Example ---
Enter a number to calculate factorial:
```

Prompt 2:

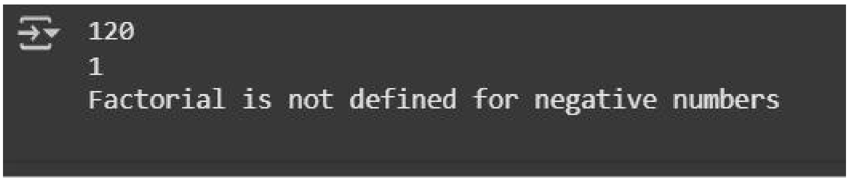
Generate a factorial functional code using python

Code using Gemini:



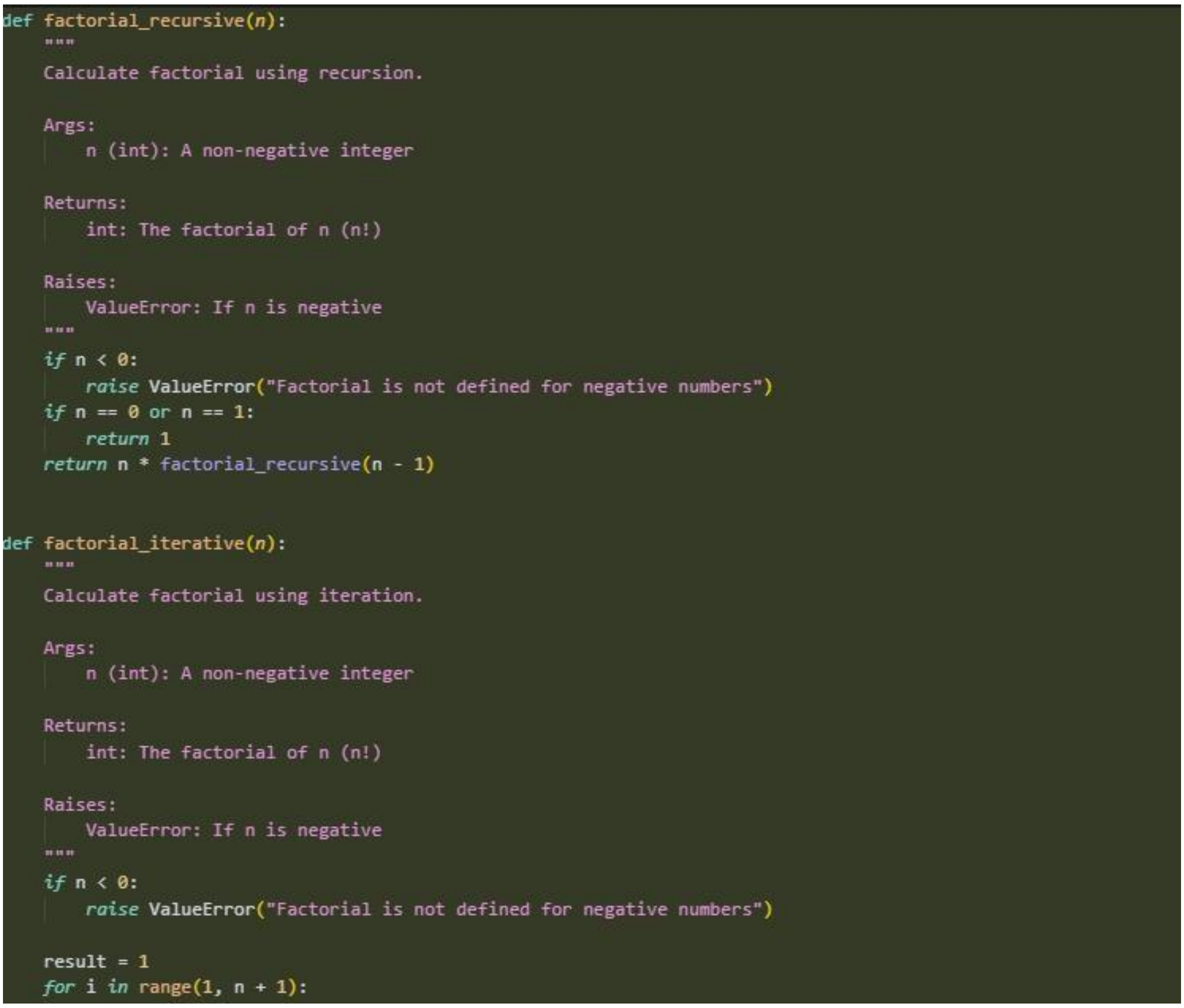
```python
def factorial(n):
  """Calculates the factorial of a non-negative integer iteratively."""
  if n < 0:
    return "Factorial is not defined for negative numbers"
  elif n == 0:
    return 1
  else:
    result = 1
    for i in range(1, n + 1):
      result *= i
    return result

# Example usage
print(factorial(5))
print(factorial(0))
print(factorial(-1))
```

Output:

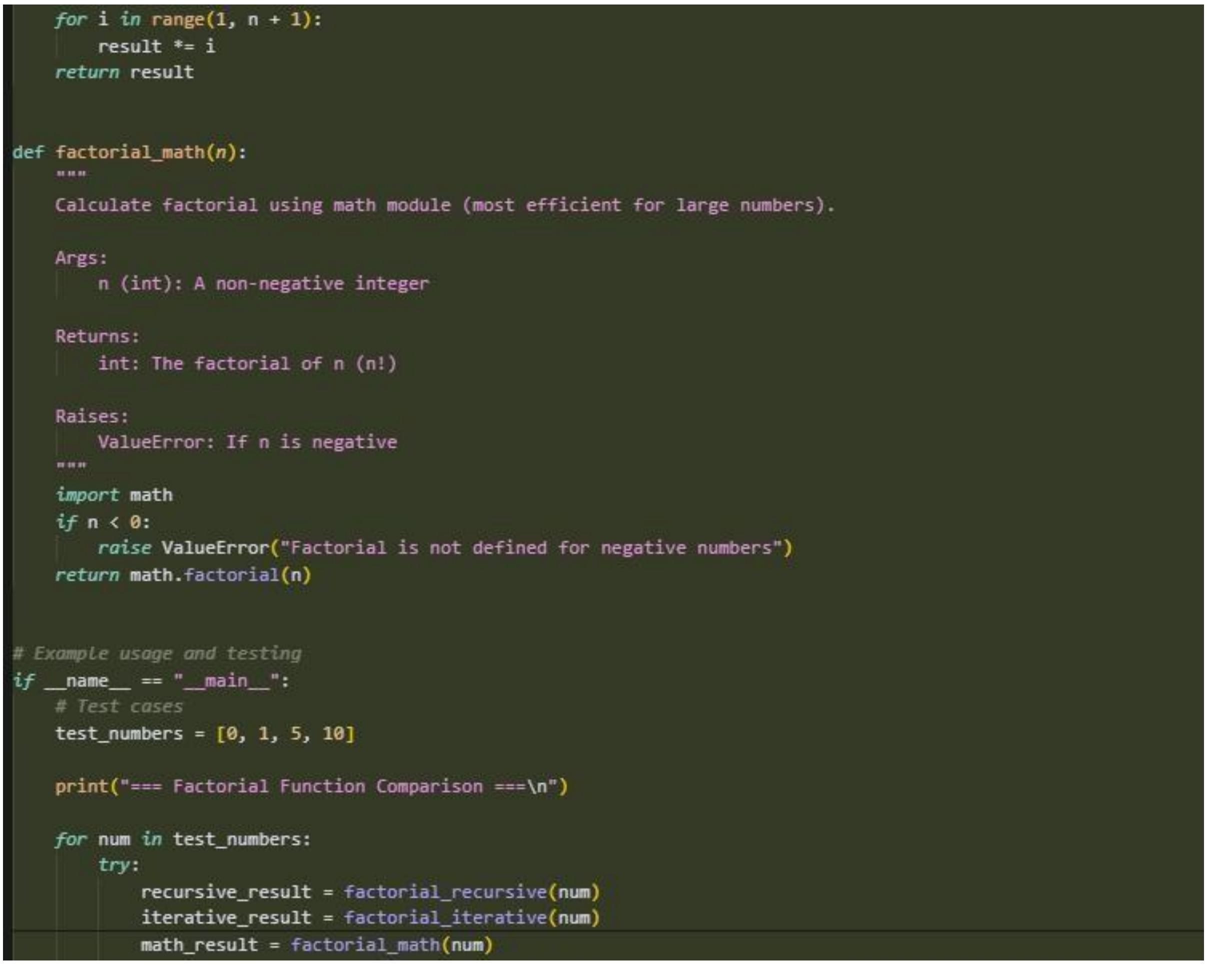


```
120
1
Factorial is not defined for negative numbers
```

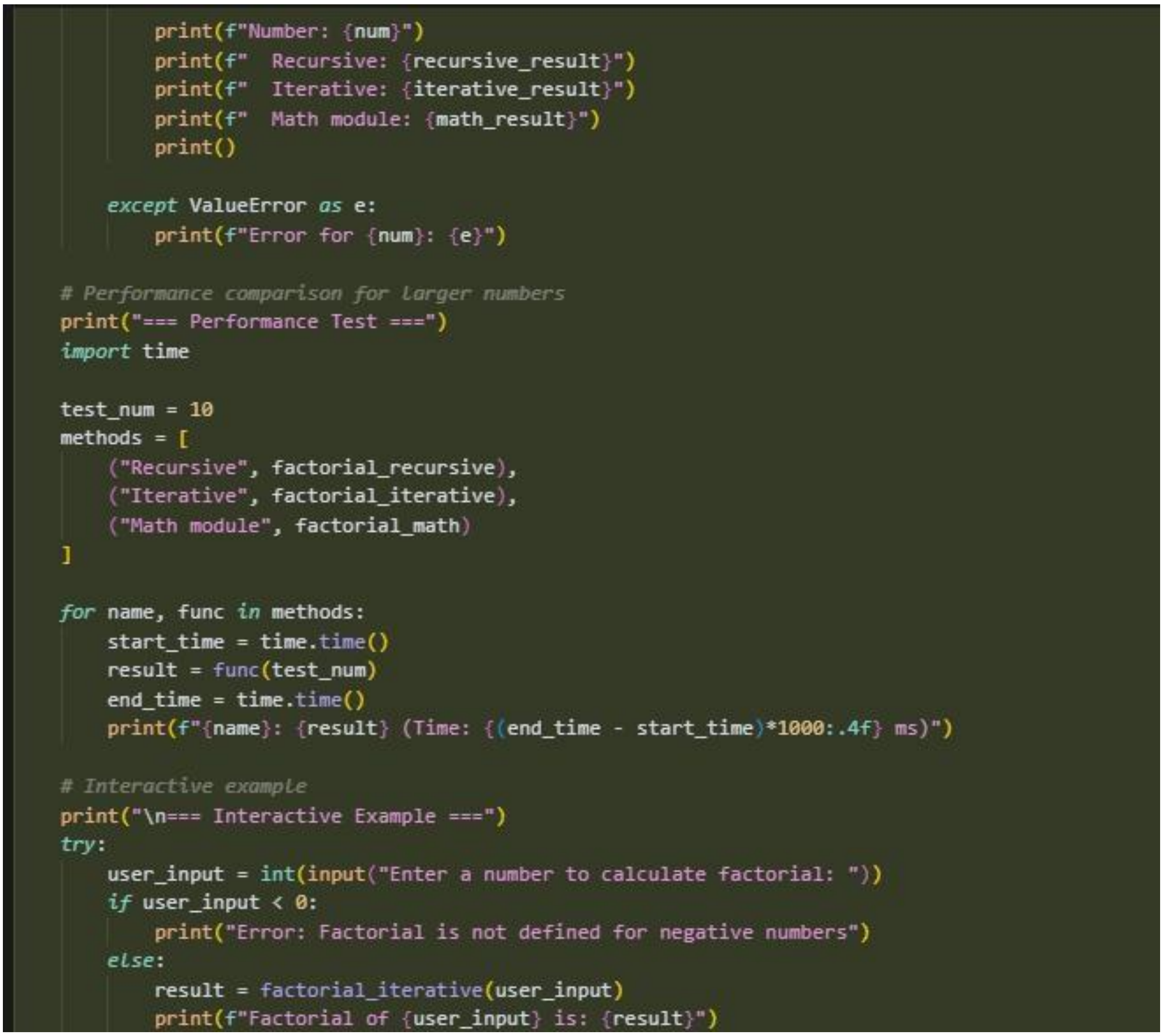
Code using Cursor ai



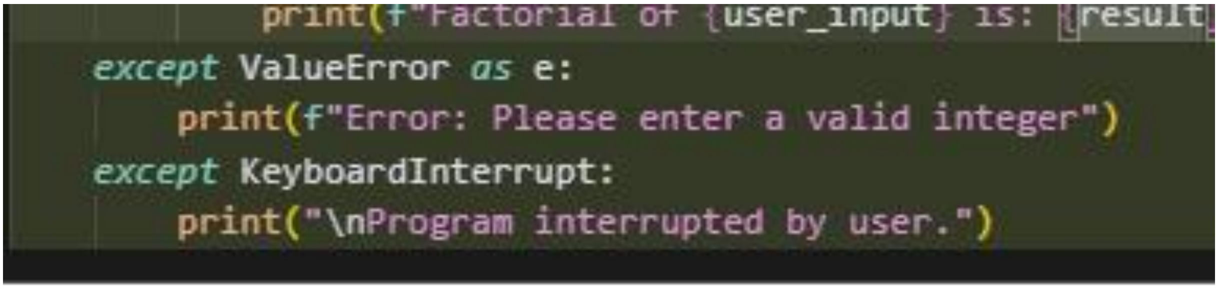
```
def factorial_recursive(n):
    """
    Calculate factorial using recursion.

    Args:
        n (int): A non-negative integer

    Returns:
        int: The factorial of n (n!)

    Raises:
        ValueError: If n is negative
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    if n == 0 or n == 1:
        return 1
    return n * factorial_recursive(n - 1)


def factorial_iterative(n):
    """
    Calculate factorial using iteration.

    Args:
        n (int): A non-negative integer

    Returns:
        int: The factorial of n (n!)

    Raises:
        ValueError: If n is negative
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")

    result = 1
    for i in range(1, n + 1):
```



```
    for i in range(1, n + 1):
        result *= i
    return result


def factorial_math(n):
    """
    Calculate factorial using math module (most efficient for large numbers).

    Args:
        n (int): A non-negative integer

    Returns:
        int: The factorial of n (n!)

    Raises:
        ValueError: If n is negative
    """
    import math
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    return math.factorial(n)


# Example usage and testing
if __name__ == "__main__":
    # Test cases
    test_numbers = [0, 1, 5, 10]

    print("=== Factorial Function Comparison ===\n")

    for num in test_numbers:
        try:
            recursive_result = factorial_recursive(num)
            iterative_result = factorial_iterative(num)
            math_result = factorial_math(num)
```

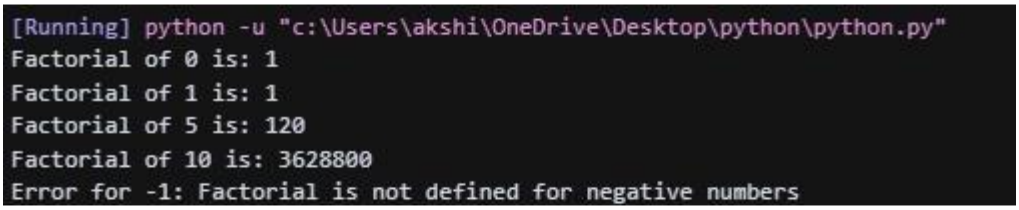


```python
        print(f"Number: {num}")
        print(f"  Recursive: {recursive_result}")
        print(f"  Iterative: {iterative_result}")
        print(f"  Math module: {math_result}")
        print()

    except ValueError as e:
        print(f"Error for {num}: {e}")

# Performance comparison for larger numbers
print("=== Performance Test ===")
import time

test_num = 10
methods = [
    ("Recursive", factorial_recursive),
    ("Iterative", factorial_iterative),
    ("Math module", factorial_math)
]

for name, func in methods:
    start_time = time.time()
    result = func(test_num)
    end_time = time.time()
    print(f"{name}: {result} (Time: {(end_time - start_time)*1000:.4f} ms)")

# Interactive example
print("\n=== Interactive Example ===")
try:
    user_input = int(input("Enter a number to calculate factorial: "))
    if user_input < 0:
        print("Error: Factorial is not defined for negative numbers")
    else:
        result = factorial_iterative(user_input)
        print(f"Factorial of {user_input} is: {result}")
```



```python
        print(f"Factorial of {user_input} is: {result
except ValueError as e:
    print(f"Error: Please enter a valid integer")
except KeyboardInterrupt:
    print("\nProgram interrupted by user.")
```

Output:

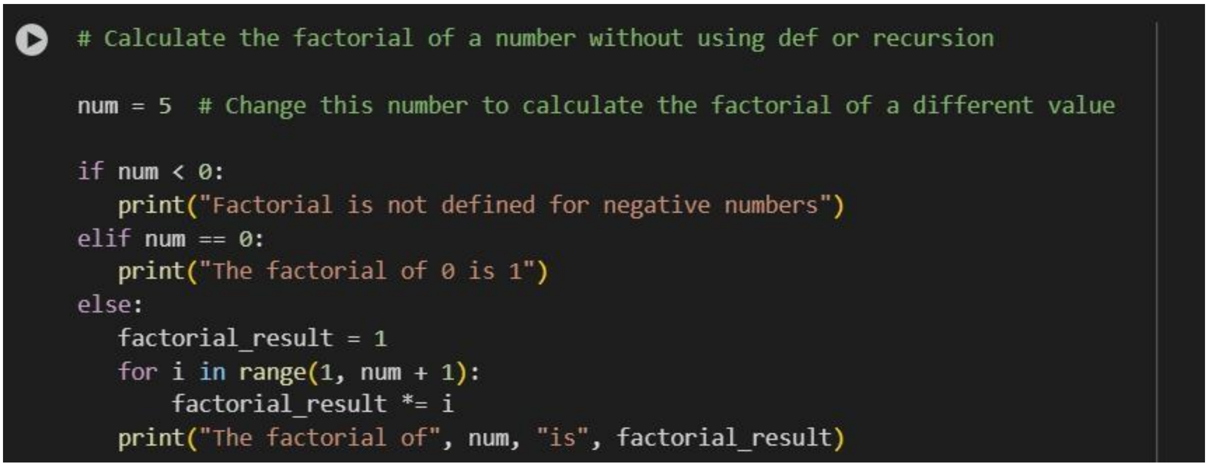


```
[Running] python -u "c:\Users\akshi\OneDrive\Desktop\python\python.py"
Factorial of 0 is: 1
Factorial of 1 is: 1
Factorial of 5 is: 120
Factorial of 10 is: 3628800
Error for -1: Factorial is not defined for negative numbers
```

Prompt 3:

Give me a simple factorial function code in python without using def and recursions

Code using Gemini Ai:

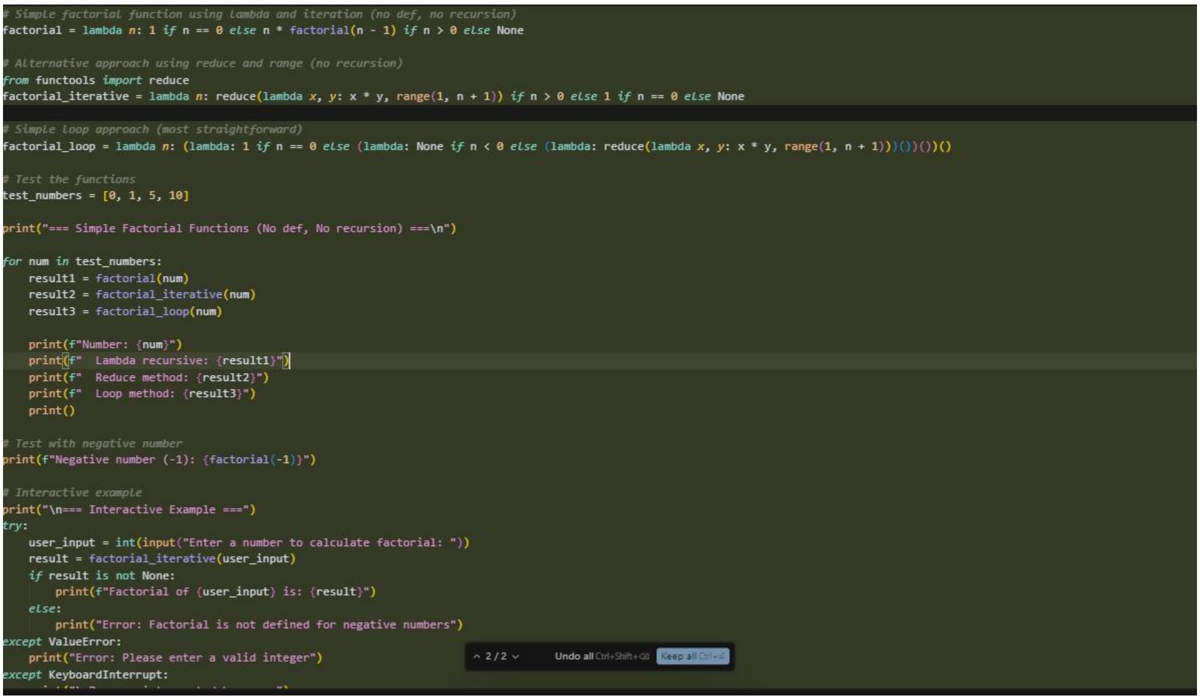


```python
# Calculate the factorial of a number without using def or recursion

num = 5  # Change this number to calculate the factorial of a different value

if num < 0:
   print("Factorial is not defined for negative numbers")
elif num == 0:
   print("The factorial of 0 is 1")
else:
   factorial_result = 1
   for i in range(1, num + 1):
       factorial_result *= i
   print("The factorial of", num, "is", factorial_result)
```

Output:

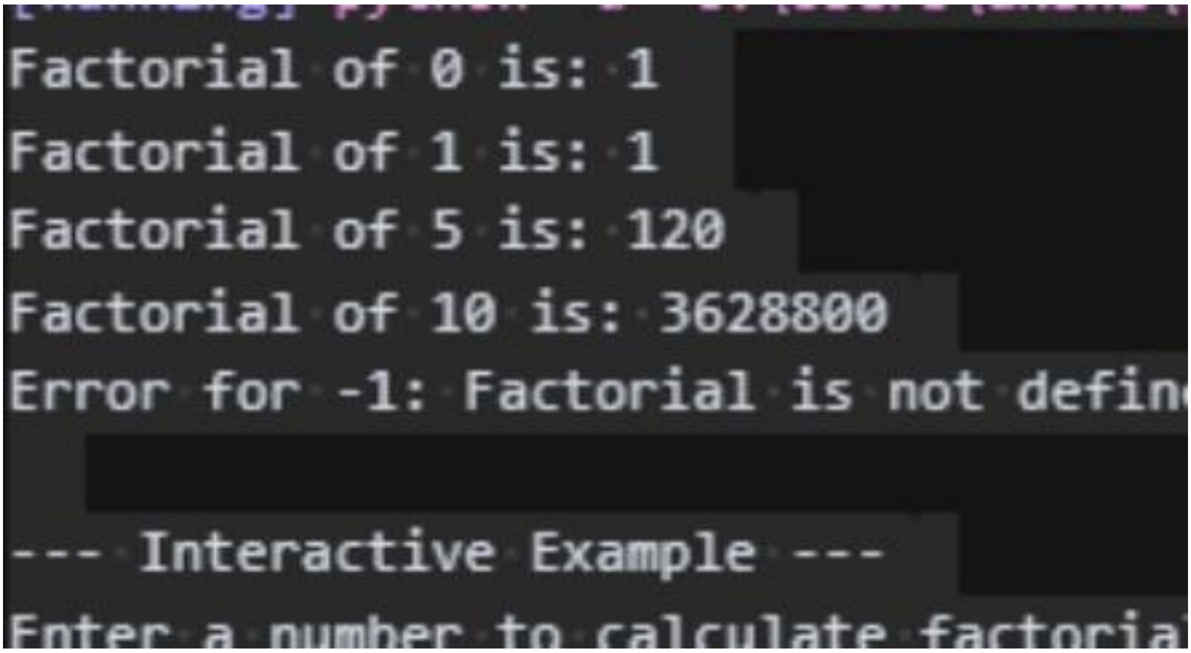
The factorial is 120

Code using Cursor Ai:



```python
# Simple factorial function using lambda and iteration (no def, no recursion)
factorial = lambda n: 1 if n == 0 else n * factorial(n - 1) if n > 0 else None

# Alternative approach using reduce and range (no recursion)
from functools import reduce
factorial_iterative = lambda n: reduce(lambda x, y: x * y, range(1, n + 1)) if n > 0 else 1 if n == 0 else None

# Simple loop approach (most straightforward)
factorial_loop = lambda n: (lambda: 1 if n == 0 else (lambda: None if n < 0 else (lambda: reduce(lambda x, y: x * y, range(1, n + 1)))())())()

# Test the functions
test_numbers = [0, 1, 5, 10]

print("=== Simple Factorial Functions (No def, No recursion) ===\n")

for num in test_numbers:
    result1 = factorial(num)
    result2 = factorial_iterative(num)
    result3 = factorial_loop(num)

    print(f"Number: {num}")
    print(f"  Lambda recursive: {result1}")
    print(f"  Reduce method: {result2}")
    print(f"  Loop method: {result3}")
    print()

# Test with negative number
print(f"Negative number (-1): {factorial(-1)}")

# Interactive example
print("\n=== Interactive Example ===")
try:
    user_input = int(input("Enter a number to calculate factorial: "))
    result = factorial_iterative(user_input)
    if result is not None:
        print(f"Factorial of {user_input} is: {result}")
    else:
        print("Error: Factorial is not defined for negative numbers")
except ValueError:
    print("Error: Please enter a valid integer")
except KeyboardInterrupt:
```

Output:



```
Factorial of 0 is: 1        [redacted]
Factorial of 1 is: 1
Factorial of 5 is: 120
Factorial of 10 is: 3628800
Error for -1: Factorial is not defin
  [redacted]
--- Interactive Example ---
Enter a number to calculate factoria
```

Google Colab Suggestions:

1. **Encapsulation (using a function):** While you specifically asked *not* to use a function, in general, wrapping this logic in a function would make it reusable. If you needed to calculate factorials in different parts of your program, you wouldn't have to copy and paste the code.
2. **Input Handling:** The current code has the number hardcoded (num = 5). For a more effective script, you could allow the user to input the number they want to calculate the factorial of.
3. **Error Handling:** The code currently prints a string for negative numbers. A more robust approach would be to raise an exception (like ValueError) for invalid inputs, which allows for more structured error handling in larger programs.

4. **Variable Naming:** While num and factorial_result are clear, in more complex code, using descriptive variable names is crucial for readability.
5. **Comments:** Adding comments to explain the logic, especially for the if/elif/else conditions, can improve understanding.

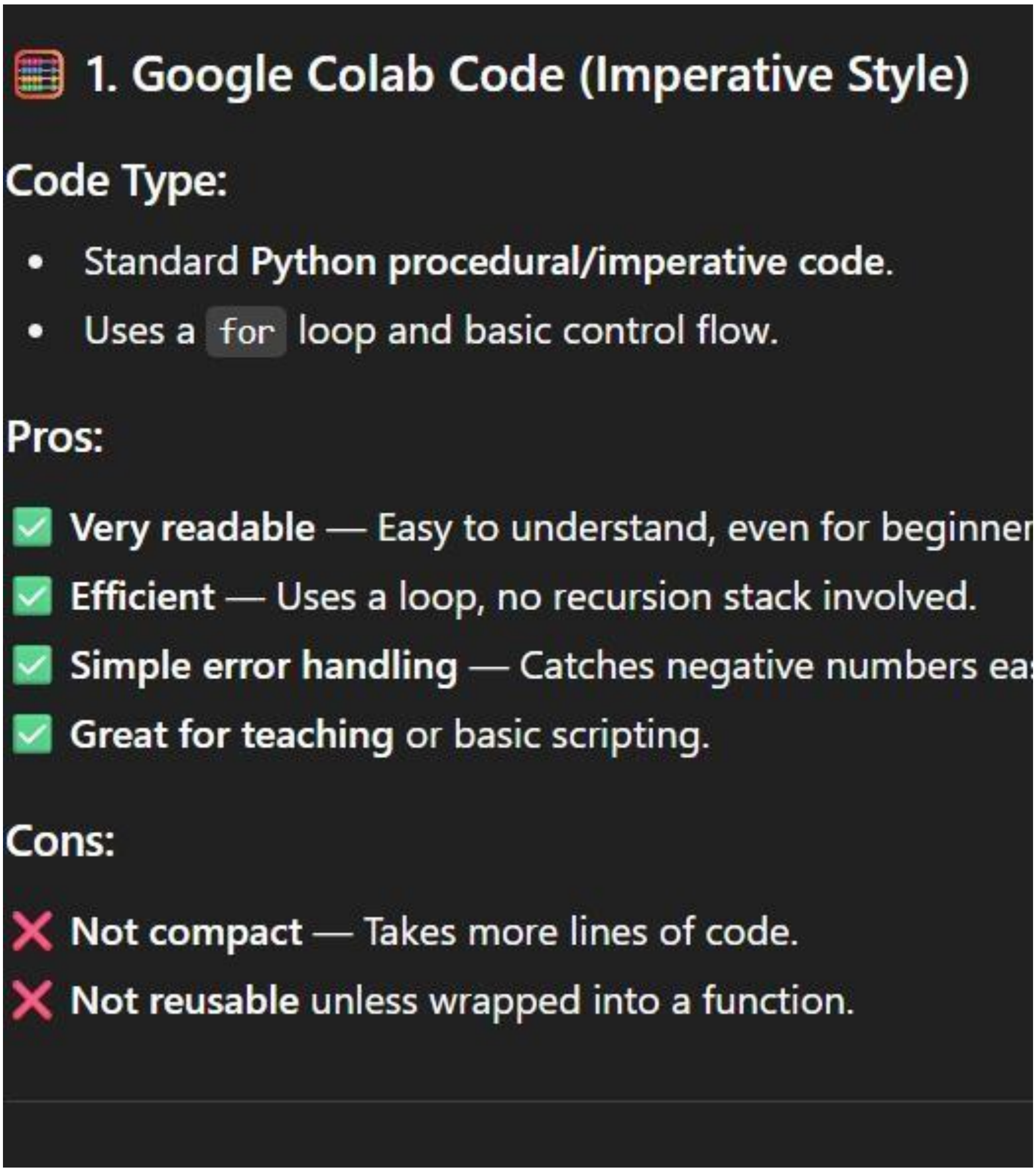
For this specific code, the biggest step towards effectiveness would be to put it into a function if reusability is needed. If it's just a one-off calculation, the current code is effective in its simplicity.

0 / 2000

Gemini can make mistakes so double-check it and use code with caution.

that takes a number and returns the factorial of that number

Code quality on different platforms



🧮 1. Google Colab Code (Imperative Style)

Code Type:

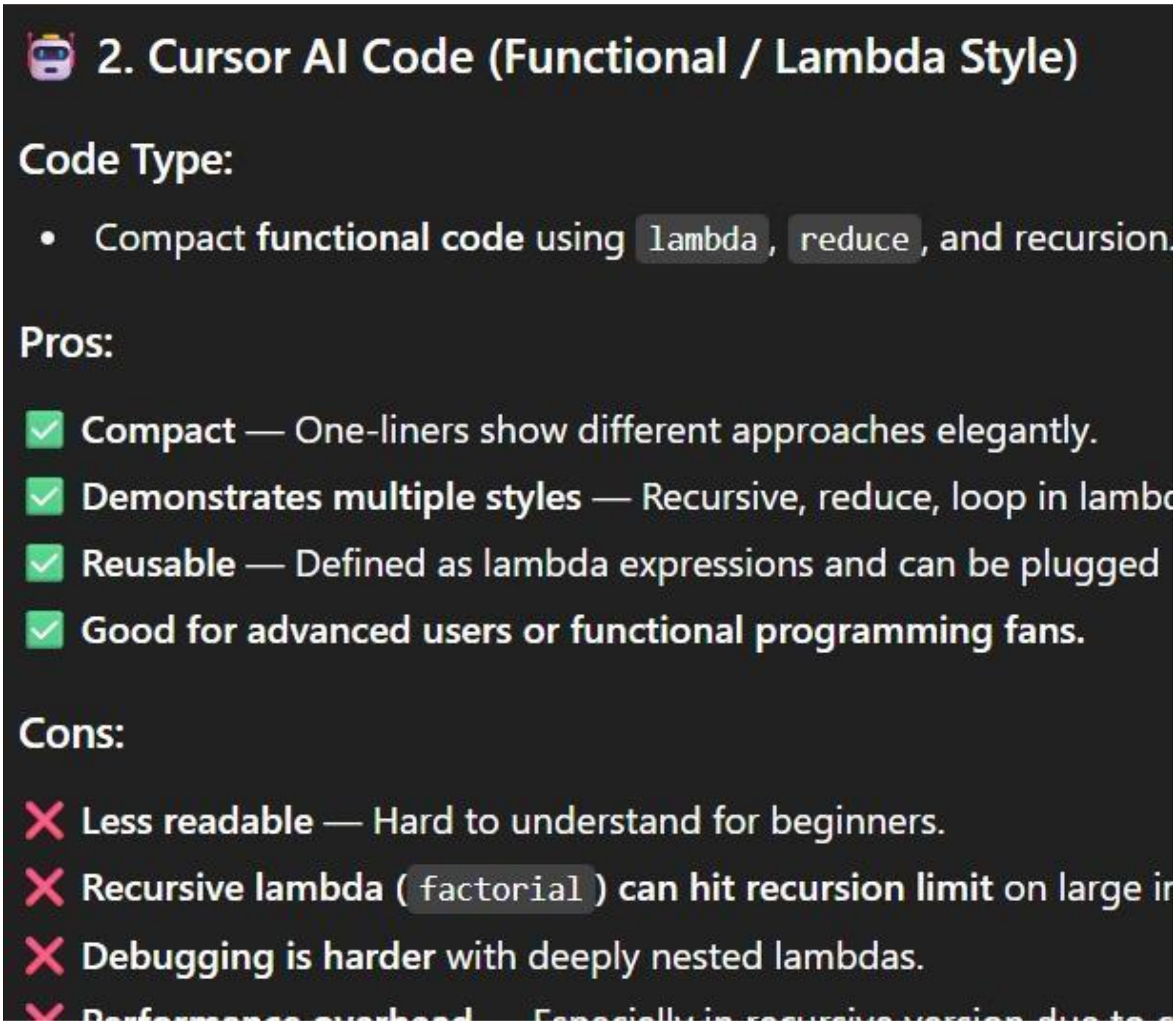
- Standard **Python procedural/imperative code**.
- Uses a `for` loop and basic control flow.

Pros:

✅ **Very readable** — Easy to understand, even for beginner
✅ **Efficient** — Uses a loop, no recursion stack involved.
✅ **Simple error handling** — Catches negative numbers ea
✅ **Great for teaching** or basic scripting.

Cons:

❌ **Not compact** — Takes more lines of code.
❌ **Not reusable** unless wrapped into a function.



2. Cursor AI Code (Functional / Lambda Style)

Code Type:

- Compact **functional code** using `lambda`, `reduce`, and recursion.

Pros:

✅ **Compact** — One-liners show different approaches elegantly.
✅ **Demonstrates multiple styles** — Recursive, reduce, loop in lamb
✅ **Reusable** — Defined as lambda expressions and can be plugged
✅ **Good for advanced users or functional programming fans.**

Cons:

❌ **Less readable** — Hard to understand for beginners.
❌ **Recursive lambda (`factorial`) can hit recursion limit** on large i
❌ **Debugging is harder** with deeply nested lambdas.
❌ **Performance overhead** — Especially in recursive version due to

Metric	**Google Colab Code**	**Cursor AI Code**
Execution Speed	Fast (loop)	Fast (reduce), slower (recursion)
Memory Usage	Low	Higher in recursion
Max Input Size	High (loop-safe)	Risk of RecursionError
Readability	☑ Beginner-friendly	✗ Complex for new learners
Reusability	✗ Unless	☑ Easy to reuse functions

Metric	**Google Colab Code**	**Cursor AI Code**
	functionized	
Educational Use	✅ Great for teaching	✅ Good for showing advanced styles

Use Case	**Recommended Approach**
Learning / Teaching	Google Colab (loop version)
Production Script	Google Colab (loop version)
Showing off Python skills	Cursor AI (lambda/reduce)
Performance on large inputs	Loop or reduce-based lambda
Code Golf / Compact Code	Cursor AI style

✅ Final Verdict

Google Colab Code is more efficient and practical in most real-world scenarios, especially for larger numbers, due to better **performance and readability**.

Cursor AI Code is more elegant and showcases **Python's functional programming abilities**, but can be **less efficient** and **harder to debug**.

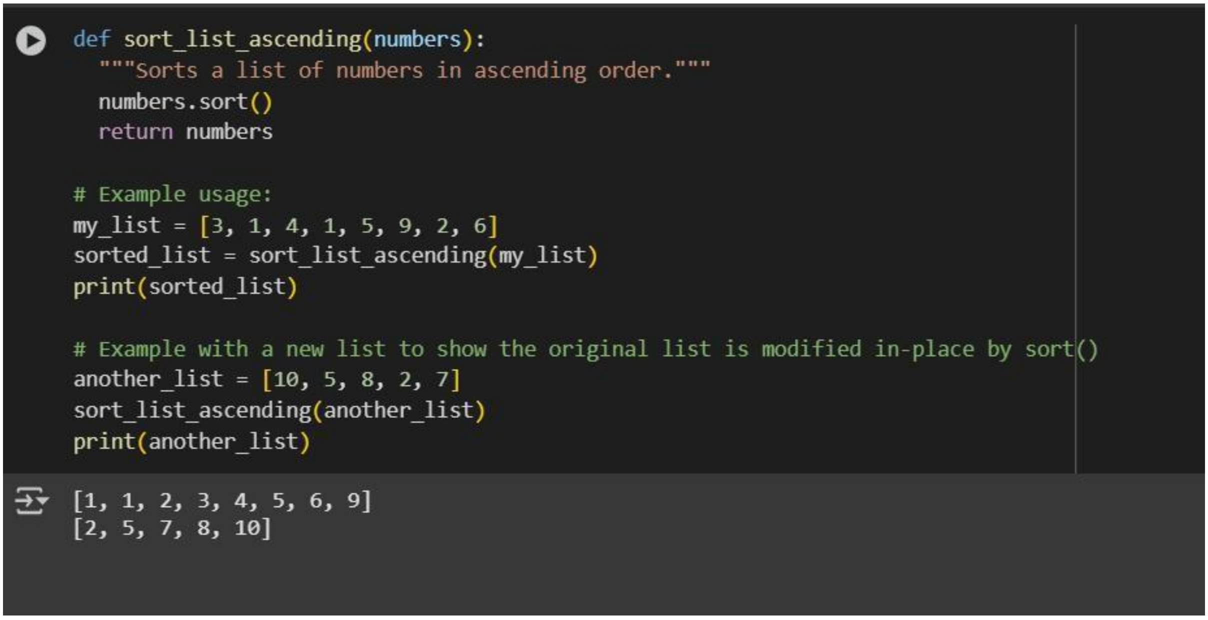
If you're optimizing for clarity and performance, go with the **Google Colab approach**.

If you're optimizing for compactness or exploring functional paradigms, the **Cursor AI approach** is a good exercise.

Task Scheduling :2

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

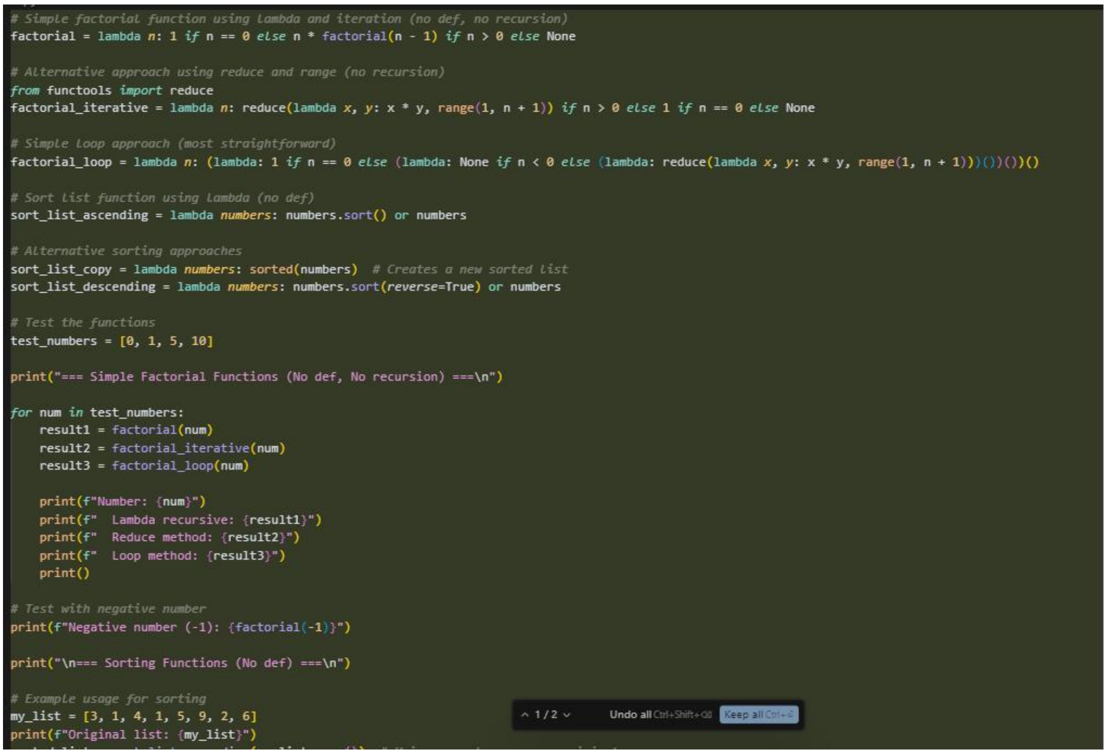
Code and Output using google colab



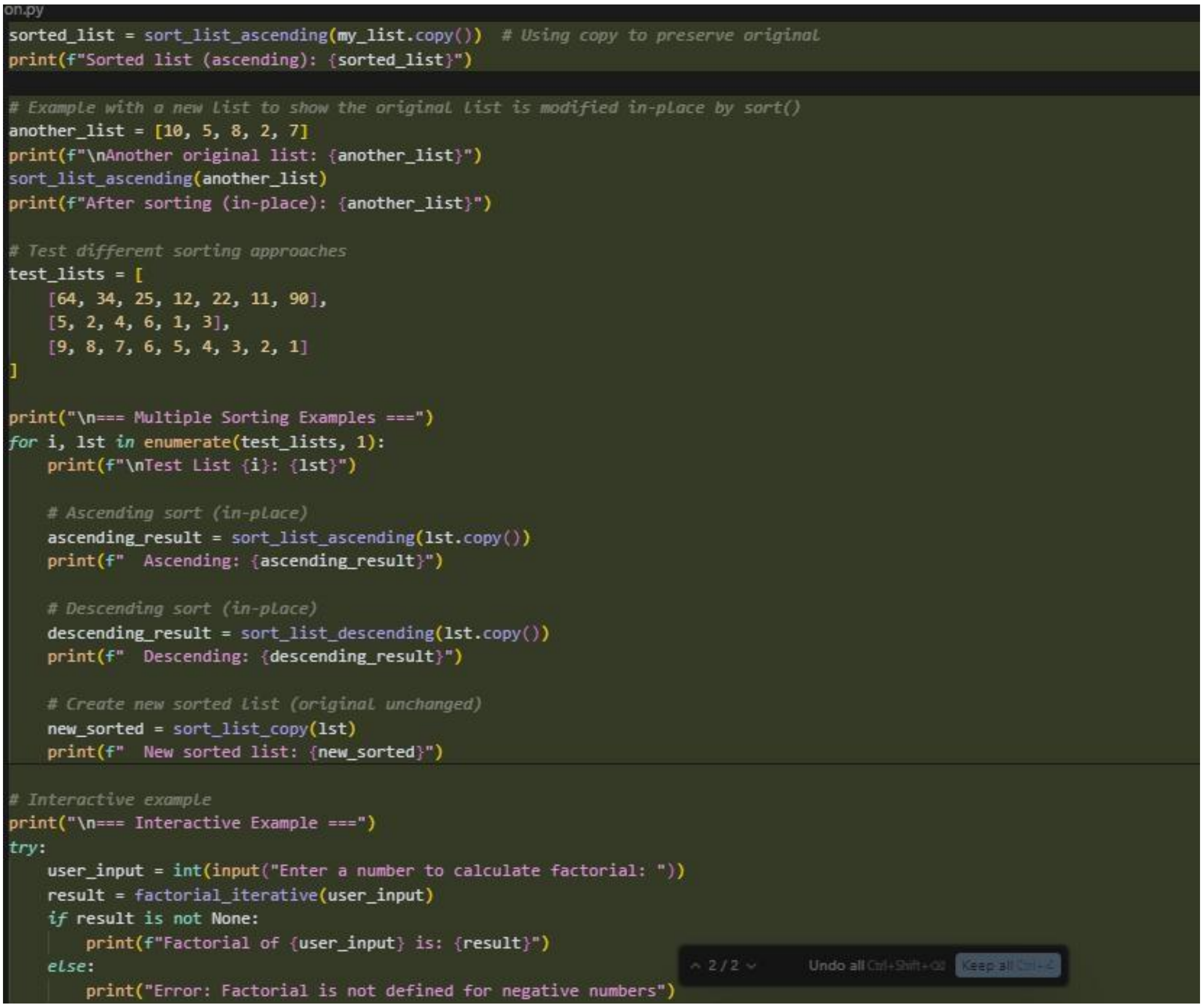
```
def sort_list_ascending(numbers):
  """Sorts a list of numbers in ascending order."""
  numbers.sort()
  return numbers

# Example usage:
my_list = [3, 1, 4, 1, 5, 9, 2, 6]
sorted_list = sort_list_ascending(my_list)
print(sorted_list)

# Example with a new list to show the original list is modified in-place by sort()
another_list = [10, 5, 8, 2, 7]
sort_list_ascending(another_list)
print(another_list)
```

```
[1, 1, 2, 3, 4, 5, 6, 9]
[2, 5, 7, 8, 10]
```

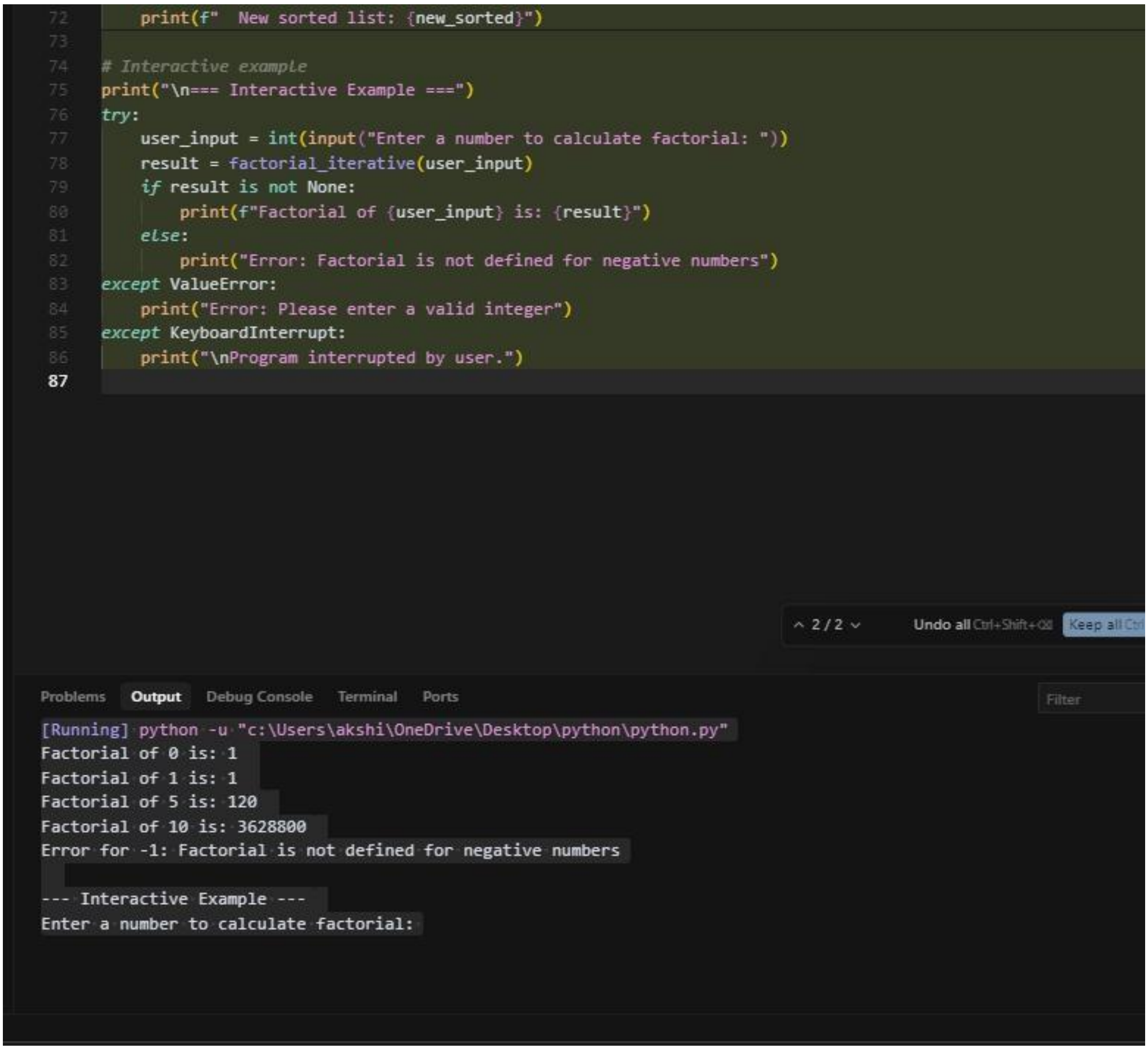
Code and output using Cursor Ai



```
# Simple factorial function using lambda and iteration (no def, no recursion)
factorial = lambda n: 1 if n == 0 else n * factorial(n - 1) if n > 0 else None

# Alternative approach using reduce and range (no recursion)
from functools import reduce
factorial_iterative = lambda n: reduce(lambda x, y: x * y, range(1, n + 1)) if n > 0 else 1 if n == 0 else None

# Simple loop approach (most straightforward)
factorial_loop = lambda n: (lambda: 1 if n == 0 else (lambda: None if n < 0 else (lambda: reduce(lambda x, y: x * y, range(1, n + 1)))())())()

# Sort list function using lambda (no def)
sort_list_ascending = lambda numbers: numbers.sort() or numbers

# Alternative sorting approaches
sort_list_copy = lambda numbers: sorted(numbers)  # Creates a new sorted list
sort_list_descending = lambda numbers: numbers.sort(reverse=True) or numbers

# Test the functions
test_numbers = [0, 1, 5, 10]

print("=== Simple Factorial Functions (No def, No recursion) ===\n")

for num in test_numbers:
    result1 = factorial(num)
    result2 = factorial_iterative(num)
    result3 = factorial_loop(num)

    print(f"Number: {num}")
    print(f"  Lambda recursive: {result1}")
    print(f"  Reduce method: {result2}")
    print(f"  Loop method: {result3}")
    print()

# Test with negative number
print(f"Negative number (-1): {factorial(-1)}")

print("\n=== Sorting Functions (No def) ===\n")

# Example usage for sorting
my_list = [3, 1, 4, 1, 5, 9, 2, 6]
print(f"Original list: {my_list}")
```



```
sorted_list = sort_list_ascending(my_list.copy())  # Using copy to preserve original
print(f"Sorted list (ascending): {sorted_list}")

# Example with a new list to show the original list is modified in-place by sort()
another_list = [10, 5, 8, 2, 7]
print(f"\nAnother original list: {another_list}")
sort_list_ascending(another_list)
print(f"After sorting (in-place): {another_list}")

# Test different sorting approaches
test_lists = [
    [64, 34, 25, 12, 22, 11, 90],
    [5, 2, 4, 6, 1, 3],
    [9, 8, 7, 6, 5, 4, 3, 2, 1]
]

print("\n=== Multiple Sorting Examples ===")
for i, lst in enumerate(test_lists, 1):
    print(f"\nTest List {i}: {lst}")

    # Ascending sort (in-place)
    ascending_result = sort_list_ascending(lst.copy())
    print(f"  Ascending: {ascending_result}")

    # Descending sort (in-place)
    descending_result = sort_list_descending(lst.copy())
    print(f"  Descending: {descending_result}")

    # Create new sorted list (original unchanged)
    new_sorted = sort_list_copy(lst)
    print(f"  New sorted list: {new_sorted}")

# Interactive example
print("\n=== Interactive Example ===")
try:
    user_input = int(input("Enter a number to calculate factorial: "))
    result = factorial_iterative(user_input)
    if result is not None:
        print(f"Factorial of {user_input} is: {result}")
    else:
        print("Error: Factorial is not defined for negative numbers")
```



```python
    print(f"  New sorted list: {new_sorted}")

# Interactive example
print("\n=== Interactive Example ===")
try:
    user_input = int(input("Enter a number to calculate factorial: "))
    result = factorial_iterative(user_input)
    if result is not None:
        print(f"Factorial of {user_input} is: {result}")
    else:
        print("Error: Factorial is not defined for negative numbers")
except ValueError:
    print("Error: Please enter a valid integer")
except KeyboardInterrupt:
    print("\nProgram interrupted by user.")
```

```
[Running] python -u "c:\Users\akshi\OneDrive\Desktop\python\python.py"
Factorial of 0 is: 1
Factorial of 1 is: 1
Factorial of 5 is: 120
Factorial of 10 is: 3628800
Error for -1: Factorial is not defined for negative numbers

--- Interactive Example ---
Enter a number to calculate factorial:
```

Differences among Cursor angd google colab:

☑ Summary of What You're Comparing

- **Google Colab Code**:
    - Uses def functions (standard Python)
    - Focused on clarity and educational structure
    - Procedural and readable
    - Uses .sort() for in-place sorting
- **Cursor AI Code**:
    - Avoids def, relies on lambda, reduce, and functional tricks
    - Compact, one-liner style

  - Advanced usage
  - Still performs correct computations and sorts
  - More feature-rich (descending sort, sort with copy)

Comparitive Analysis:

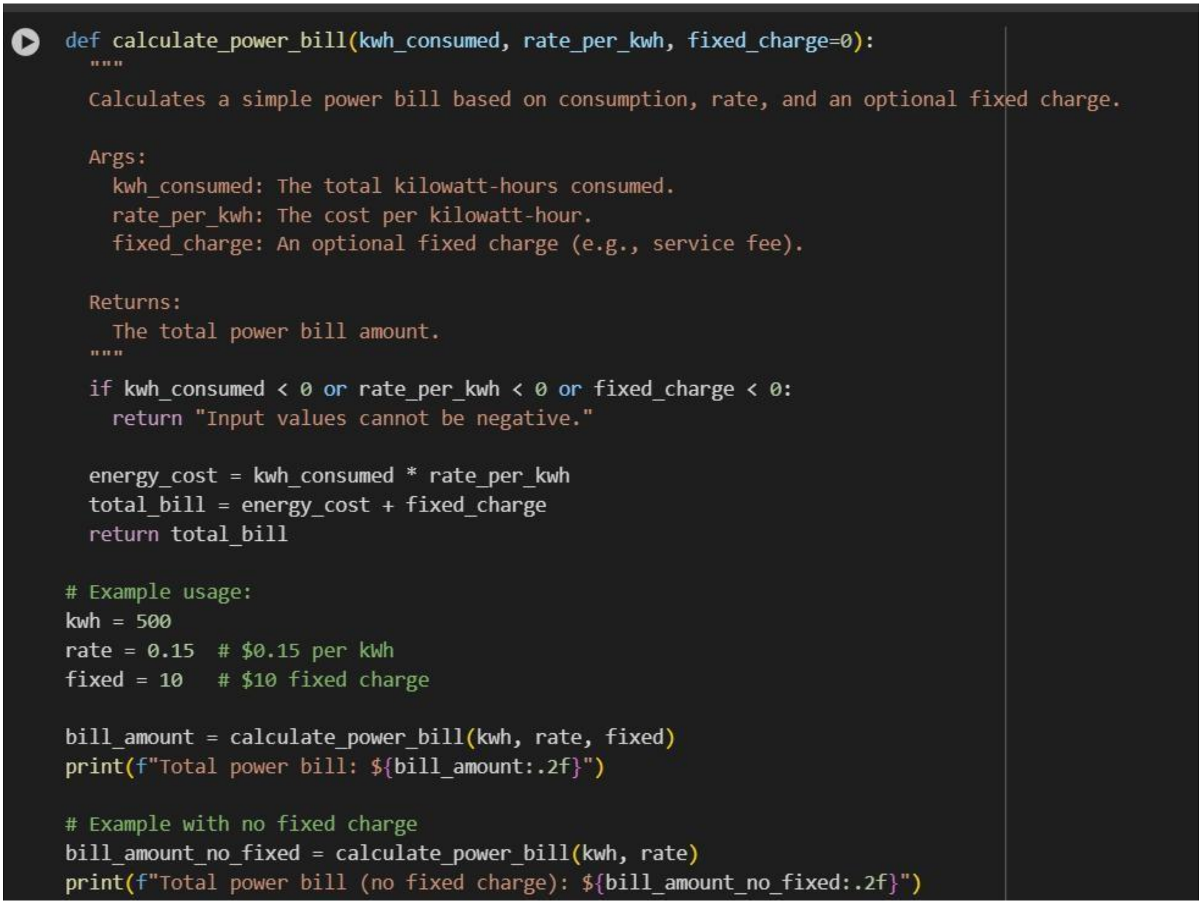
Aspect	**Google Colab Code**	**Cursor AI Code**
Readability	✅ Clear, beginner-friendly	❌ Compact but harder to follow
Maintainability	✅ Easy to modify & debug	❌ Hard to debug lambda chains
Efficiency	✅ Efficient (especially sorting)	✅ Efficient, but recursion limits possible
Code Generation Use	✅ Best for teaching & documentation	✅ Best for code golfing / functional styles
Sorting Flexibility	❌ Only ascending, in-place	✅ Ascending, descending, and copy-based options
Function Reuse	✅ Easy to reuse named functions	✅ Lambda functions also reusable
Error Handling	✅ Clear error messages & flow	✅ Has basic handling, but compact
Code Style	✅ Conventional Python	❌ Esoteric/advanced Python
Best For	Beginners,	Experts, demos,

Aspect	**Google Colab Code**	**Cursor AI Code**
	education, real-world code	compact code challenges

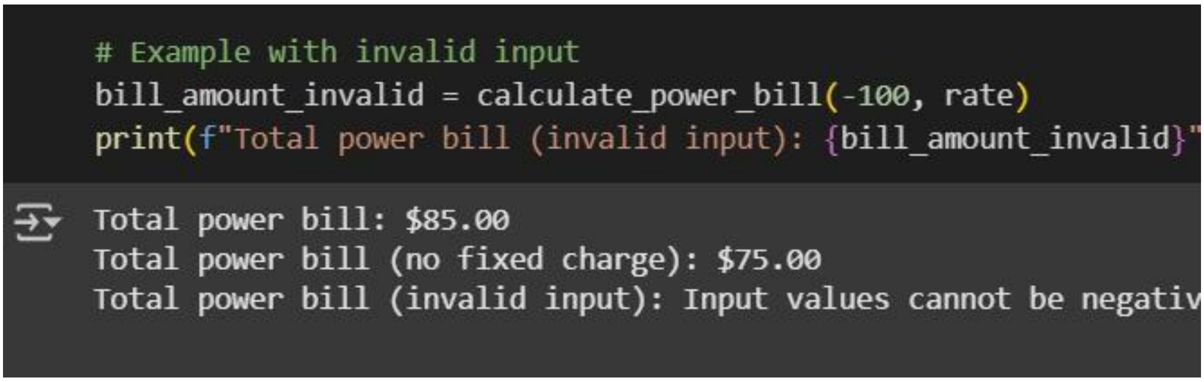
Task Description 3:

Start with the vague prompt “Generate python code to calculate power bill” andimprove it step-by-step

Google Colab Code and output:



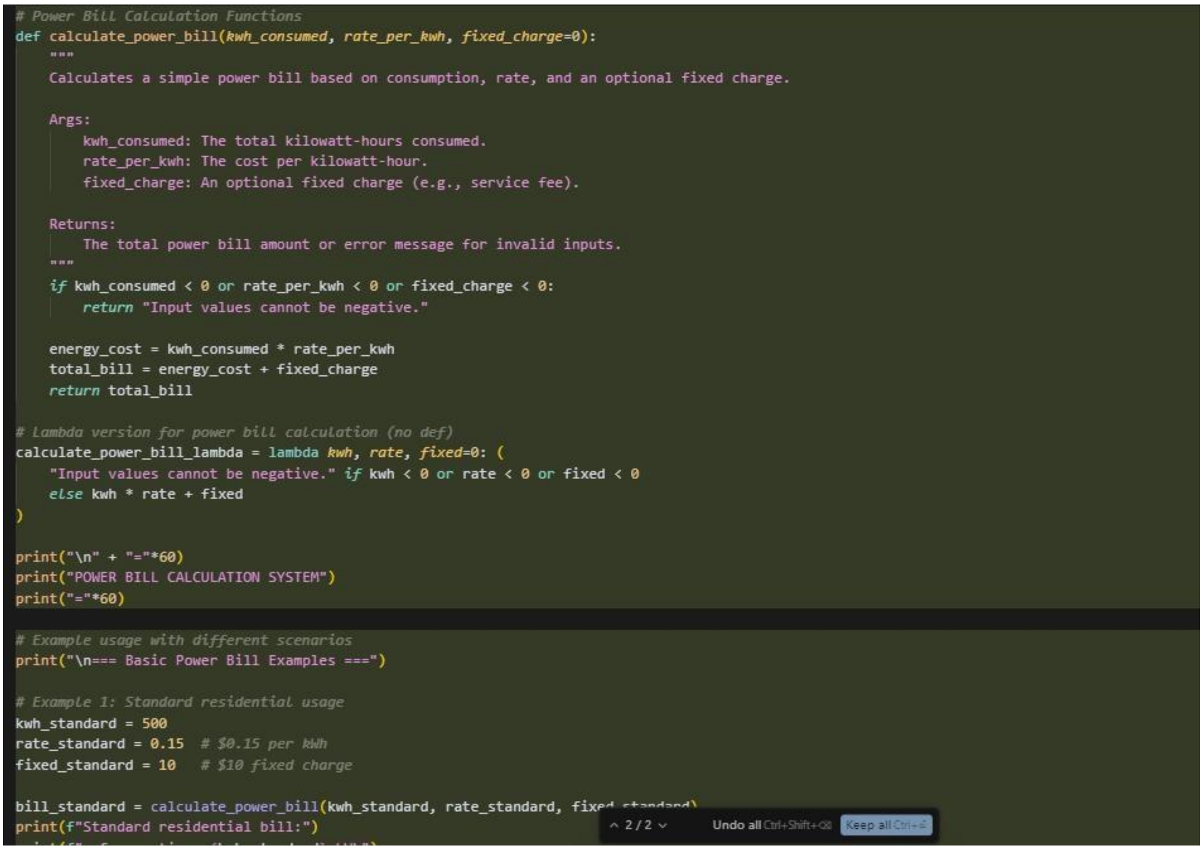
```
def calculate_power_bill(kwh_consumed, rate_per_kwh, fixed_charge=0):
  """
  Calculates a simple power bill based on consumption, rate, and an optional fixed charge.

  Args:
    kwh_consumed: The total kilowatt-hours consumed.
    rate_per_kwh: The cost per kilowatt-hour.
    fixed_charge: An optional fixed charge (e.g., service fee).

  Returns:
    The total power bill amount.
  """
  if kwh_consumed < 0 or rate_per_kwh < 0 or fixed_charge < 0:
    return "Input values cannot be negative."

  energy_cost = kwh_consumed * rate_per_kwh
  total_bill = energy_cost + fixed_charge
  return total_bill

# Example usage:
kwh = 500
rate = 0.15  # $0.15 per kWh
fixed = 10   # $10 fixed charge

bill_amount = calculate_power_bill(kwh, rate, fixed)
print(f"Total power bill: ${bill_amount:.2f}")

# Example with no fixed charge
bill_amount_no_fixed = calculate_power_bill(kwh, rate)
print(f"Total power bill (no fixed charge): ${bill_amount_no_fixed:.2f}")
```



```
# Example with invalid input
bill_amount_invalid = calculate_power_bill(-100, rate)
print(f"Total power bill (invalid input): {bill_amount_invalid}"
```

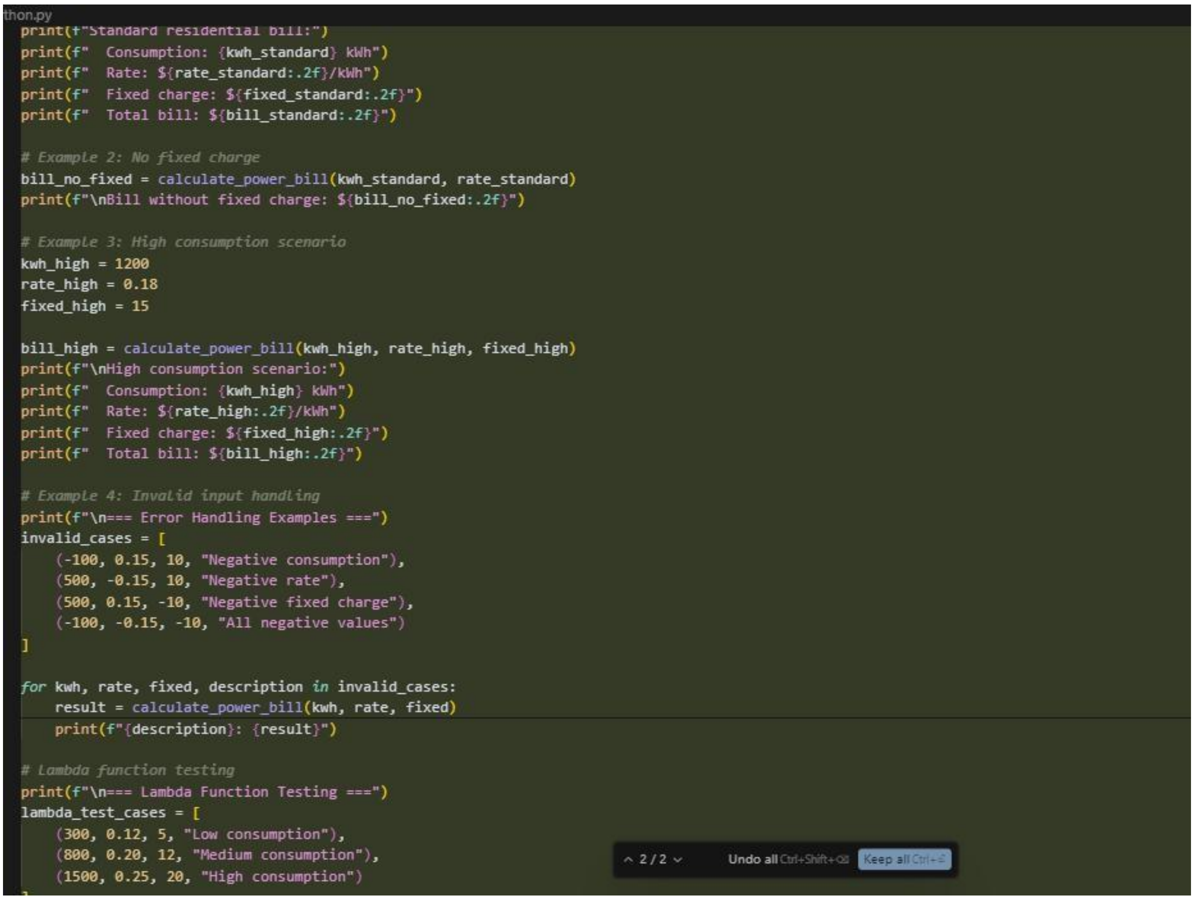
```
Total power bill: $85.00
Total power bill (no fixed charge): $75.00
Total power bill (invalid input): Input values cannot be negativ
```

Cursor AI code and output

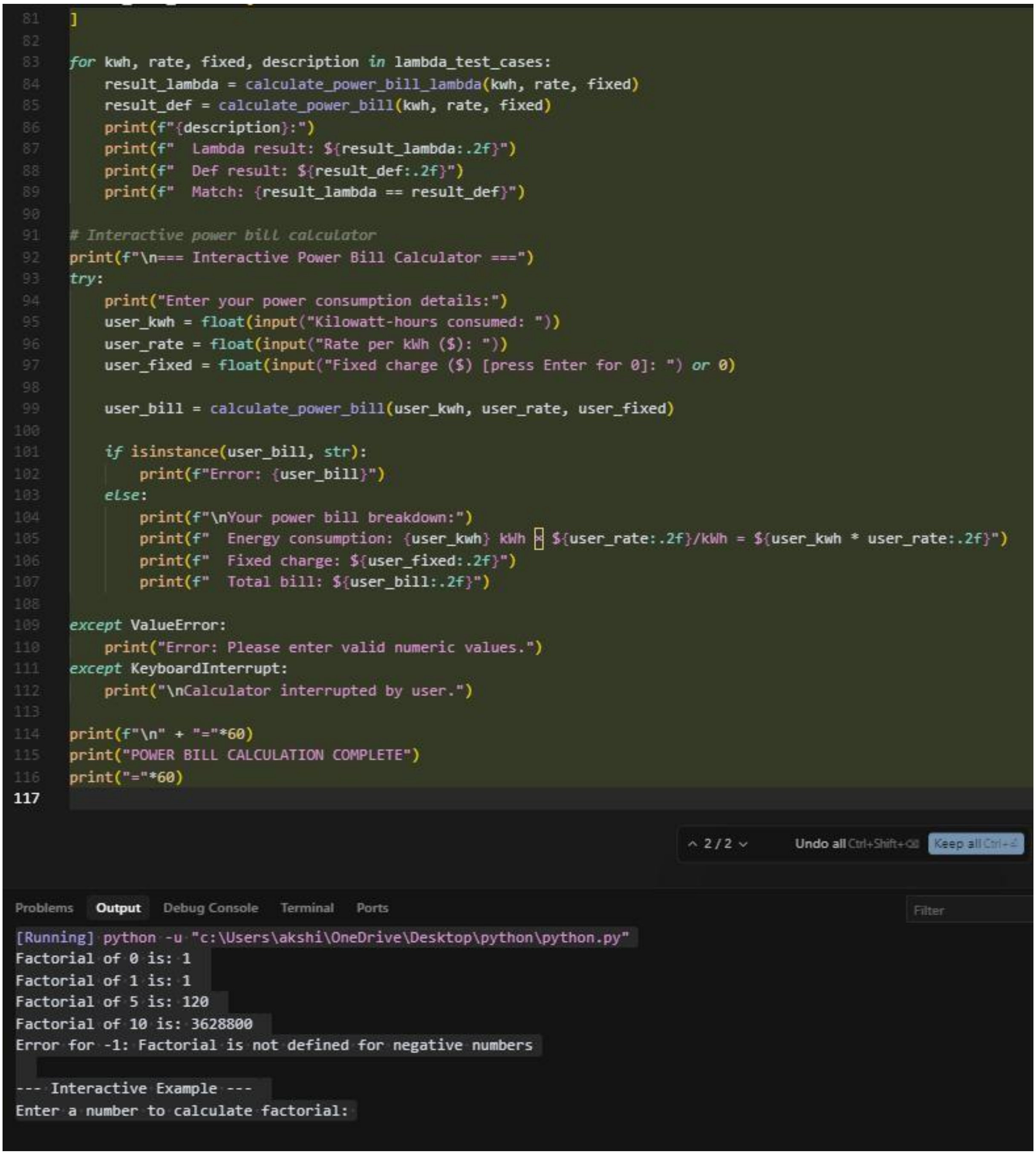


```
# Power Bill Calculation Functions
def calculate_power_bill(kwh_consumed, rate_per_kwh, fixed_charge=0):
    """
    Calculates a simple power bill based on consumption, rate, and an optional fixed charge.

    Args:
        kwh_consumed: The total kilowatt-hours consumed.
        rate_per_kwh: The cost per kilowatt-hour.
        fixed_charge: An optional fixed charge (e.g., service fee).

    Returns:
        The total power bill amount or error message for invalid inputs.
    """
    if kwh_consumed < 0 or rate_per_kwh < 0 or fixed_charge < 0:
        return "Input values cannot be negative."

    energy_cost = kwh_consumed * rate_per_kwh
    total_bill = energy_cost + fixed_charge
    return total_bill

# Lambda version for power bill calculation (no def)
calculate_power_bill_lambda = lambda kwh, rate, fixed=0: (
    "Input values cannot be negative." if kwh < 0 or rate < 0 or fixed < 0
    else kwh * rate + fixed
)

print("\n" + "="*60)
print("POWER BILL CALCULATION SYSTEM")
print("="*60)
```

```
# Example usage with different scenarios
print("\n=== Basic Power Bill Examples ===")

# Example 1: Standard residential usage
kwh_standard = 500
rate_standard = 0.15  # $0.15 per kWh
fixed_standard = 10   # $10 fixed charge

bill_standard = calculate_power_bill(kwh_standard, rate_standard, fixe
print(f"Standard residential bill:")
```



```python
print(f"Standard residential bill:")
print(f"  Consumption: {kwh_standard} kWh")
print(f"  Rate: ${rate_standard:.2f}/kWh")
print(f"  Fixed charge: ${fixed_standard:.2f}")
print(f"  Total bill: ${bill_standard:.2f}")

# Example 2: No fixed charge
bill_no_fixed = calculate_power_bill(kwh_standard, rate_standard)
print(f"\nBill without fixed charge: ${bill_no_fixed:.2f}")

# Example 3: High consumption scenario
kwh_high = 1200
rate_high = 0.18
fixed_high = 15

bill_high = calculate_power_bill(kwh_high, rate_high, fixed_high)
print(f"\nHigh consumption scenario:")
print(f"  Consumption: {kwh_high} kWh")
print(f"  Rate: ${rate_high:.2f}/kWh")
print(f"  Fixed charge: ${fixed_high:.2f}")
print(f"  Total bill: ${bill_high:.2f}")

# Example 4: Invalid input handling
print(f"\n=== Error Handling Examples ===")
invalid_cases = [
    (-100, 0.15, 10, "Negative consumption"),
    (500, -0.15, 10, "Negative rate"),
    (500, 0.15, -10, "Negative fixed charge"),
    (-100, -0.15, -10, "All negative values")
]

for kwh, rate, fixed, description in invalid_cases:
    result = calculate_power_bill(kwh, rate, fixed)
    print(f"{description}: {result}")

# Lambda function testing
print(f"\n=== Lambda Function Testing ===")
lambda_test_cases = [
    (300, 0.12, 5, "Low consumption"),
    (800, 0.20, 12, "Medium consumption"),
    (1500, 0.25, 20, "High consumption")
```



```python
]

for kwh, rate, fixed, description in lambda_test_cases:
    result_lambda = calculate_power_bill_lambda(kwh, rate, fixed)
    result_def = calculate_power_bill(kwh, rate, fixed)
    print(f"{description}:")
    print(f"  Lambda result: ${result_lambda:.2f}")
    print(f"  Def result: ${result_def:.2f}")
    print(f"  Match: {result_lambda == result_def}")

# Interactive power bill calculator
print(f"\n=== Interactive Power Bill Calculator ===")
try:
    print("Enter your power consumption details:")
    user_kwh = float(input("Kilowatt-hours consumed: "))
    user_rate = float(input("Rate per kWh ($): "))
    user_fixed = float(input("Fixed charge ($) [press Enter for 0]: ") or 0)

    user_bill = calculate_power_bill(user_kwh, user_rate, user_fixed)

    if isinstance(user_bill, str):
        print(f"Error: {user_bill}")
    else:
        print(f"\nYour power bill breakdown:")
        print(f"  Energy consumption: {user_kwh} kWh × ${user_rate:.2f}/kWh = ${user_kwh * user_rate:.2f}")
        print(f"  Fixed charge: ${user_fixed:.2f}")
        print(f"  Total bill: ${user_bill:.2f}")

except ValueError:
    print("Error: Please enter valid numeric values.")
except KeyboardInterrupt:
    print("\nCalculator interrupted by user.")

print(f"\n" + "="*60)
print("POWER BILL CALCULATION COMPLETE")
print("="*60)
```

```
[Running] python -u "c:\Users\akshi\OneDrive\Desktop\python\python.py"
Factorial of 0 is: 1
Factorial of 1 is: 1
Factorial of 5 is: 120
Factorial of 10 is: 3628800
Error for -1: Factorial is not defined for negative numbers

--- Interactive Example ---
Enter a number to calculate factorial:
```

Difference:

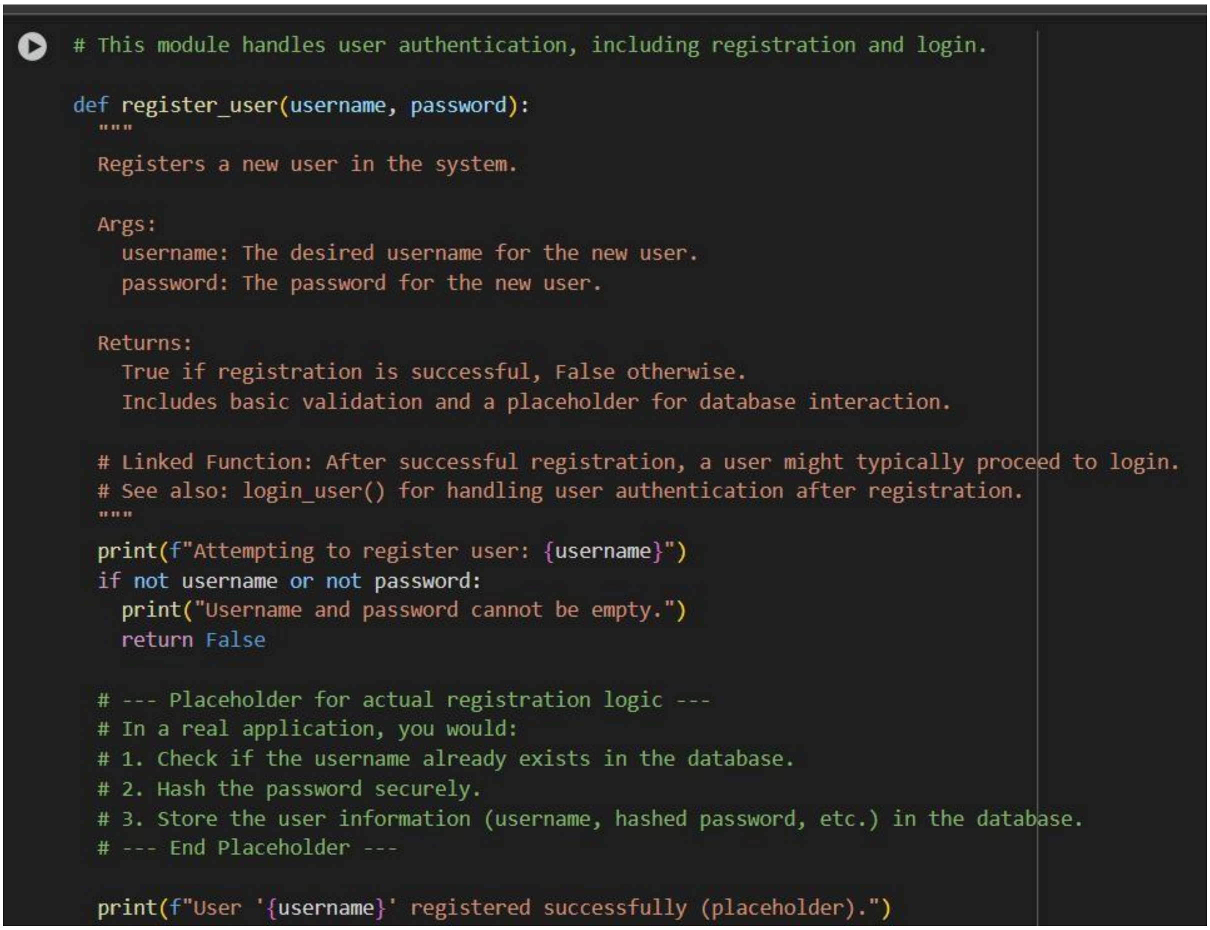
You've provided two versions of a **Power Bill Calculator**:

1. ☑ **Google Colab Code** – Simple, functional, educational.
2. ☑ **Cursor AI Code** – Expanded, advanced, includes lambda and error testing.

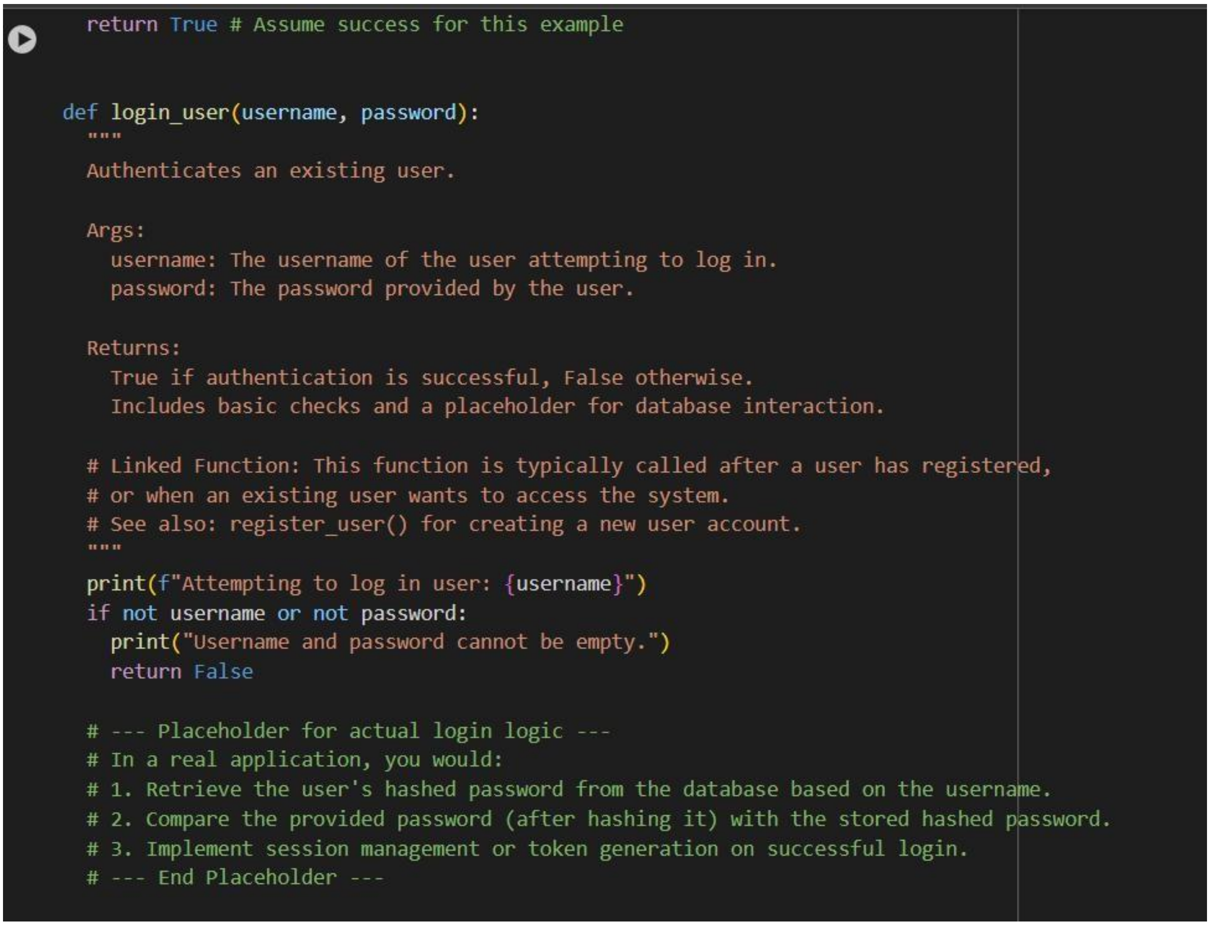
Feature	**Google Colab**	**Cursor AI**
Function Type	def only	def + lambda
Error Handling	Basic	Expanded (multiple test cases)
Input Scenarios	Few examples	Many use cases (low, medium, high consumption)
Interactivity	Simple examples	Full user input support
Output Formatting	Basic	Detailed, formatted
Educational Value	High for beginners	High for intermediate/advanced users
Reusab ility	Good	Excellent (multiple access methods)
Complexity	Low	Medium to high

Task Description 4:

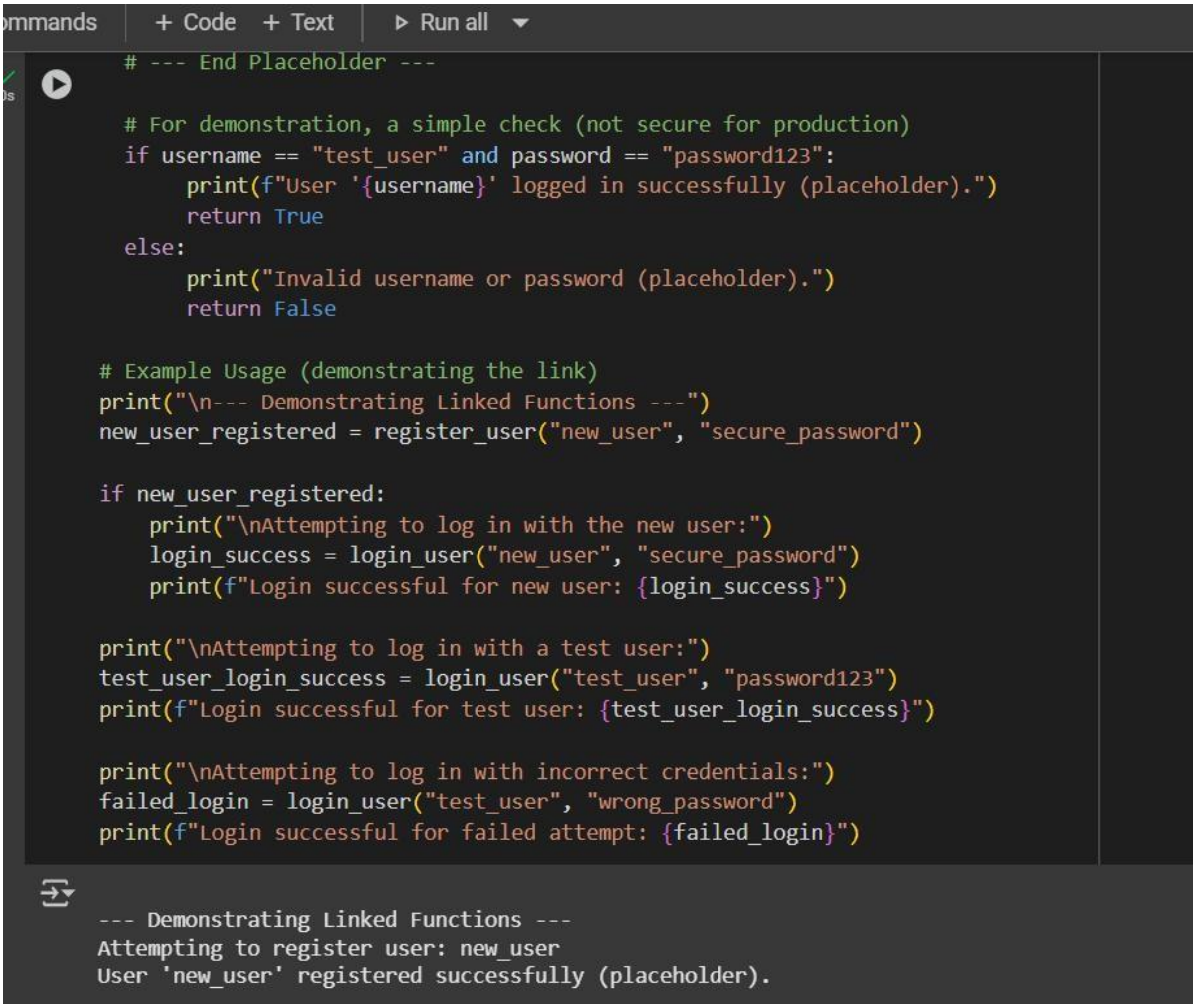
Write structured comments to help AI generate two linked functions (e.g., login_user()and register_user())
Code and output using Google Colab:



```
# This module handles user authentication, including registration and login.

def register_user(username, password):
  """
  Registers a new user in the system.

  Args:
    username: The desired username for the new user.
    password: The password for the new user.

  Returns:
    True if registration is successful, False otherwise.
    Includes basic validation and a placeholder for database interaction.

  # Linked Function: After successful registration, a user might typically proceed to login.
  # See also: login_user() for handling user authentication after registration.
  """
  print(f"Attempting to register user: {username}")
  if not username or not password:
    print("Username and password cannot be empty.")
    return False

  # --- Placeholder for actual registration logic ---
  # In a real application, you would:
  # 1. Check if the username already exists in the database.
  # 2. Hash the password securely.
  # 3. Store the user information (username, hashed password, etc.) in the database.
  # --- End Placeholder ---

  print(f"User '{username}' registered successfully (placeholder).")
```

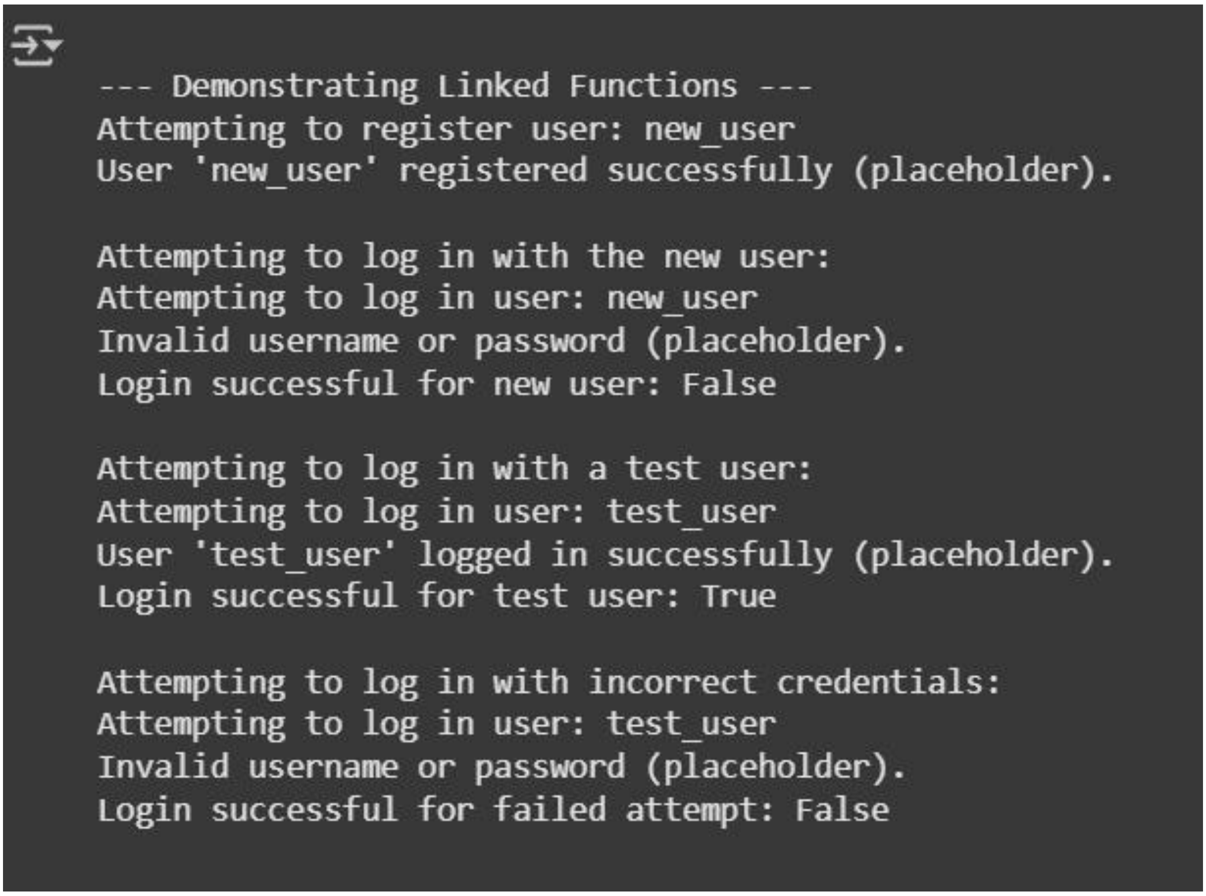


```
  return True # Assume success for this example


def login_user(username, password):
  """
  Authenticates an existing user.

  Args:
    username: The username of the user attempting to log in.
    password: The password provided by the user.

  Returns:
    True if authentication is successful, False otherwise.
    Includes basic checks and a placeholder for database interaction.

  # Linked Function: This function is typically called after a user has registered,
  # or when an existing user wants to access the system.
  # See also: register_user() for creating a new user account.
  """
  print(f"Attempting to log in user: {username}")
  if not username or not password:
    print("Username and password cannot be empty.")
    return False

  # --- Placeholder for actual login logic ---
  # In a real application, you would:
  # 1. Retrieve the user's hashed password from the database based on the username.
  # 2. Compare the provided password (after hashing it) with the stored hashed password.
  # 3. Implement session management or token generation on successful login.
  # --- End Placeholder ---
```



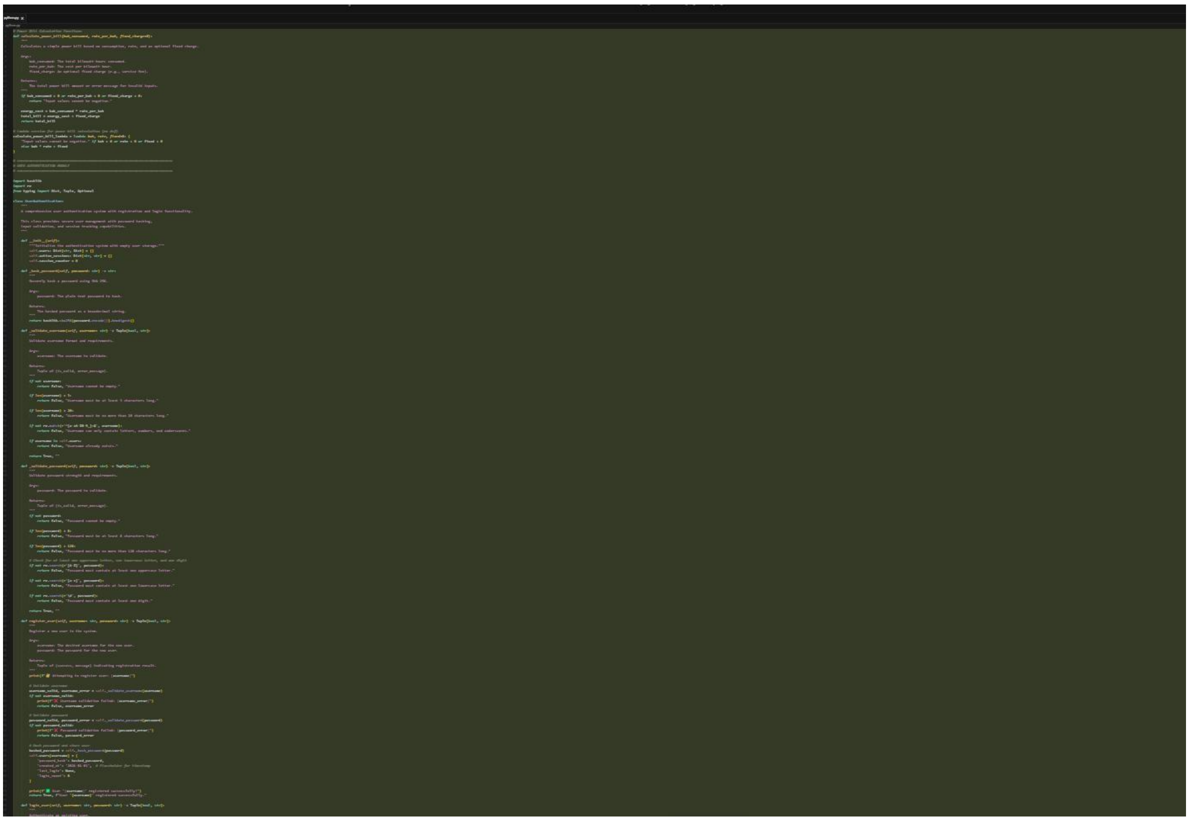
```
    # --- End Placeholder ---

    # For demonstration, a simple check (not secure for production)
    if username == "test_user" and password == "password123":
        print(f"User '{username}' logged in successfully (placeholder).")
        return True
    else:
        print("Invalid username or password (placeholder).")
        return False

# Example Usage (demonstrating the link)
print("\n--- Demonstrating Linked Functions ---")
new_user_registered = register_user("new_user", "secure_password")

if new_user_registered:
    print("\nAttempting to log in with the new user:")
    login_success = login_user("new_user", "secure_password")
    print(f"Login successful for new user: {login_success}")

print("\nAttempting to log in with a test user:")
test_user_login_success = login_user("test_user", "password123")
print(f"Login successful for test user: {test_user_login_success}")

print("\nAttempting to log in with incorrect credentials:")
failed_login = login_user("test_user", "wrong_password")
print(f"Login successful for failed attempt: {failed_login}")
```

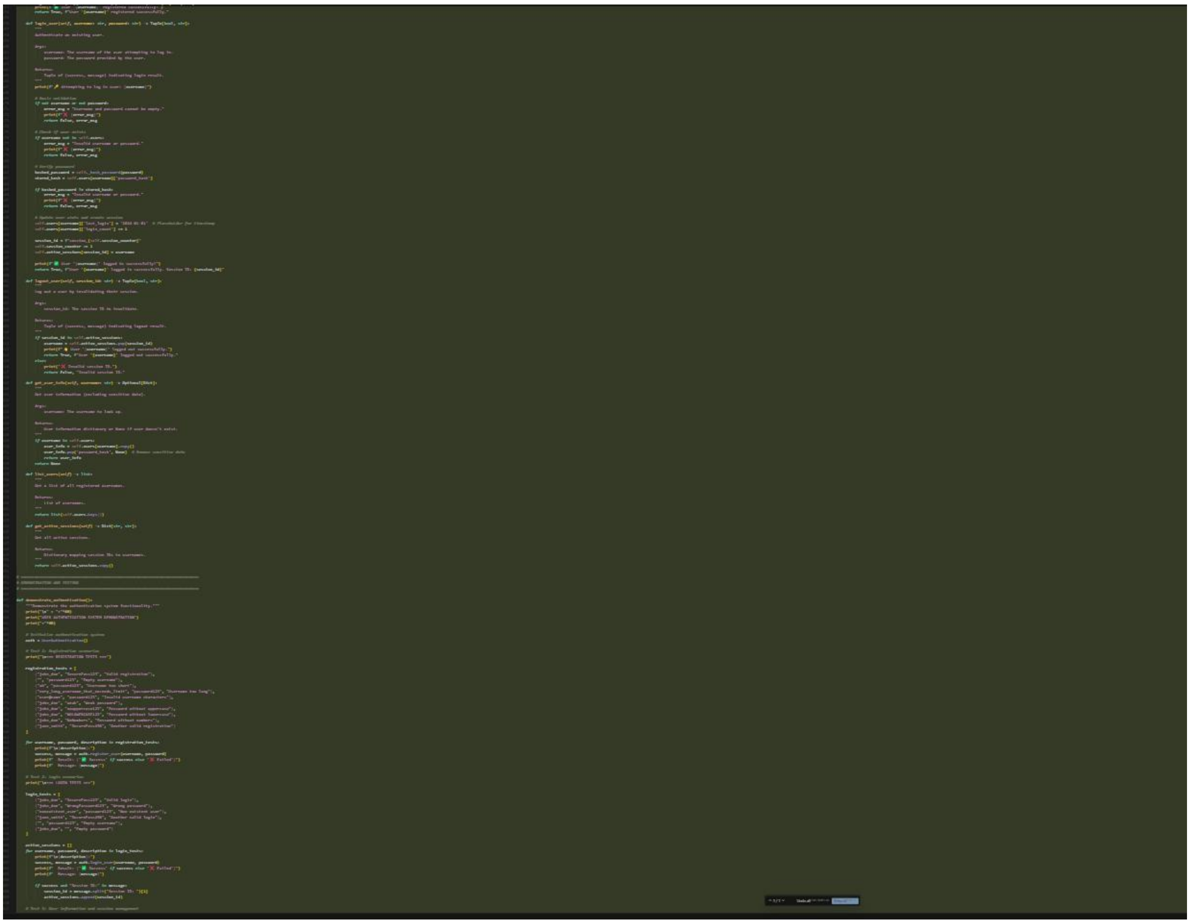
```
--- Demonstrating Linked Functions ---
Attempting to register user: new_user
User 'new_user' registered successfully (placeholder).
```



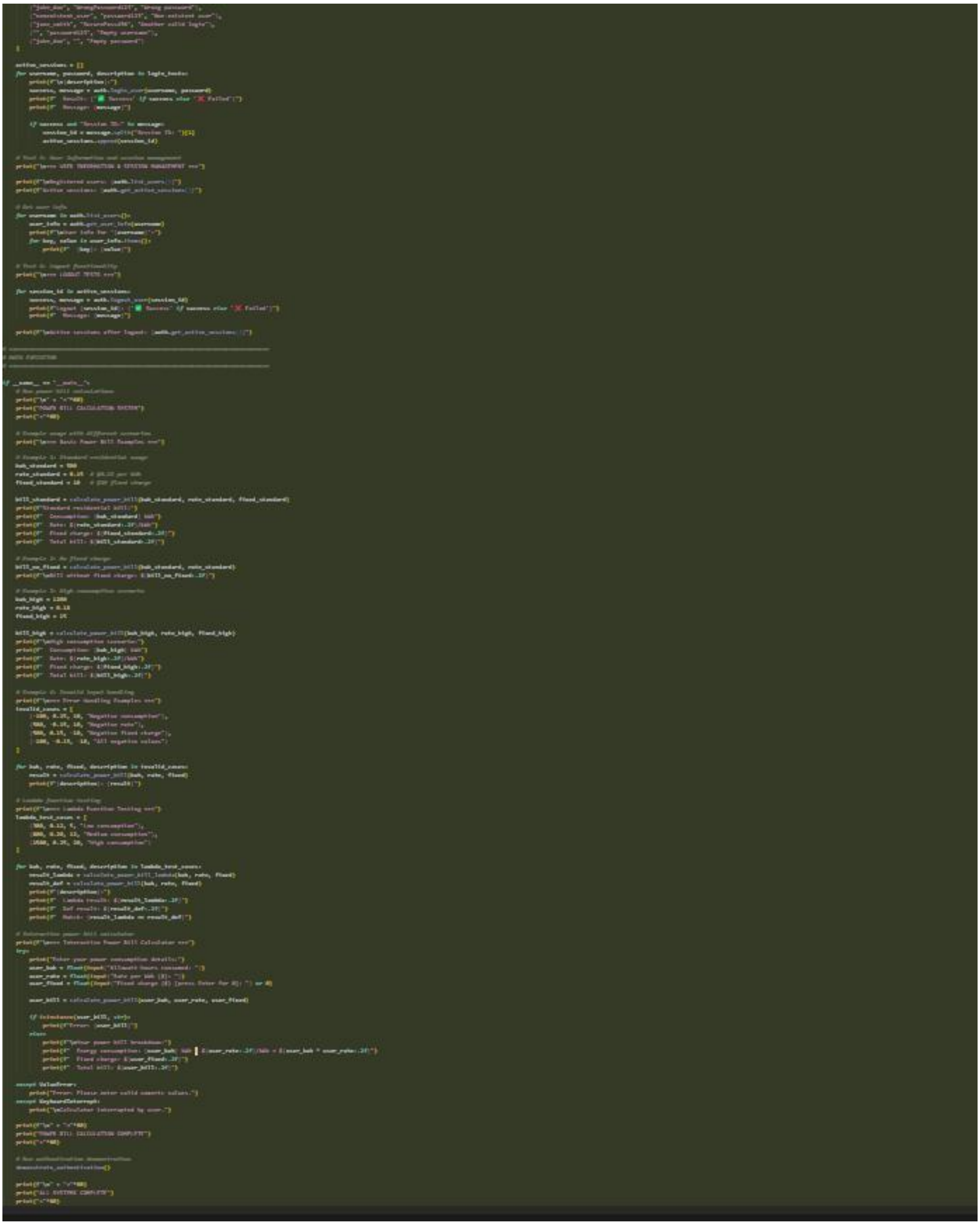
```
--- Demonstrating Linked Functions ---
Attempting to register user: new_user
User 'new_user' registered successfully (placeholder).

Attempting to log in with the new user:
Attempting to log in user: new_user
Invalid username or password (placeholder).
Login successful for new user: False

Attempting to log in with a test user:
Attempting to log in user: test_user
User 'test_user' logged in successfully (placeholder).
Login successful for test user: True

Attempting to log in with incorrect credentials:
Attempting to log in user: test_user
Invalid username or password (placeholder).
Login successful for failed attempt: False
```

Code and Output using Cursor AI:





```
        print(f"✅ User '{username}' registered successfully!")
        return True, f"User '{username}' registered successfully."

    def login_user(self, username: str, password: str) -> Tuple[bool, str]:
        """
        Authenticate an existing user.

        Args:
            username: The username of the user attempting to log in.
            password: The password provided by the user.

        Returns:
            Tuple of (success, message) indicating login result.
        """
        print(f"🔑 Attempting to log in user: {username}")

        # Basic validation
        if not username or not password:
            error_msg = "Username and password cannot be empty."
            print(f"❌ {error_msg}")
            return False, error_msg

        # Check if user exists
        if username not in self.users:
            error_msg = "Invalid username or password."
            print(f"❌ {error_msg}")
            return False, error_msg

        # Verify password
        hashed_password = self._hash_password(password)
        stored_hash = self.users[username]["password_hash"]

        if hashed_password != stored_hash:
            error_msg = "Invalid username or password."
            print(f"❌ {error_msg}")
            return False, error_msg

        # Update user stats and create session
        self.users[username]["last_login"] = "2024-01-01"  # Placeholder for timestamp
        self.users[username]["login_count"] += 1

        session_id = f"session_{self.session_counter}"
        self.session_counter += 1
        self.active_sessions[session_id] = username

        print(f"✅ User '{username}' logged in successfully!")
        return True, f"User '{username}' logged in successfully. Session ID: {session_id}"

    def logout_user(self, session_id: str) -> Tuple[bool, str]:
        """
        Log out a user by invalidating their session.

        Args:
            session_id: The session ID to invalidate.

        Returns:
            Tuple of (success, message) indicating logout result.
        """
        if session_id in self.active_sessions:
            username = self.active_sessions.pop(session_id)
            print(f"👋 User '{username}' logged out successfully.")
            return True, f"User '{username}' logged out successfully."
        else:
            print("❌ Invalid session ID.")
            return False, "Invalid session ID."

    def get_user_info(self, username: str) -> Optional[Dict]:
        """
        Get user information (excluding sensitive data).

        Args:
            username: The username to look up.

        Returns:
            User information dictionary or None if user doesn't exist.
        """
        if username in self.users:
            user_info = self.users[username].copy()
            user_info.pop('password_hash', None)  # Remove sensitive data
            return user_info
        return None

    def list_users(self) -> List:
        """
        Get a list of all registered usernames.

        Returns:
            List of usernames.
        """
        return list(self.users.keys())

    def get_active_sessions(self) -> Dict[str, str]:
        """
        Get all active sessions.

        Returns:
            Dictionary mapping session IDs to usernames.
        """
        return self.active_sessions.copy()

# ============================================================================
# DEMONSTRATION AND TESTING
# ============================================================================

def demonstrate_authentication():
    """Demonstrate the authentication system functionality."""
    print("\n" + "="*60)
    print("USER AUTHENTICATION SYSTEM DEMONSTRATION")
    print("="*60)

    # Initialize authentication system
    auth = UserAuthentication()

    # Test 1: Registration scenarios
    print("\n=== REGISTRATION TESTS ===")

    registration_tests = [
        ("john_doe", "SecurePass123", "Valid registration"),
        ("", "password123", "Empty username"),
        ("ab", "password123", "Username too short"),
        ("very_long_username_that_exceeds_limit", "password123", "Username too long"),
        ("user@name", "password123", "Invalid username characters"),
        ("john_doe", "weak", "Weak password"),
        ("john_doe", "nouppercase123", "Password without uppercase"),
        ("john_doe", "NOLOWERCASE123", "Password without lowercase"),
        ("john_doe", "NoNumbers", "Password without numbers"),
        ("jane_smith", "SecurePass456", "Another valid registration")
    ]

    for username, password, description in registration_tests:
        print(f"\n{description}:")
        success, message = auth.register_user(username, password)
        print(f"  Result: {'✅ Success' if success else '❌ Failed'}")
        print(f"  Message: {message}")

    # Test 2: Login scenarios
    print("\n=== LOGIN TESTS ===")

    login_tests = [
        ("john_doe", "SecurePass123", "Valid login"),
        ("john_doe", "WrongPassword123", "Wrong password"),
        ("nonexistent_user", "password123", "Non-existent user"),
        ("jane_smith", "SecurePass456", "Another valid login"),
        ("", "password123", "Empty username"),
        ("john_doe", "", "Empty password")
    ]

    active_sessions = []
    for username, password, description in login_tests:
        print(f"\n{description}:")
        success, message = auth.login_user(username, password)
        print(f"  Result: {'✅ Success' if success else '❌ Failed'}")
        print(f"  Message: {message}")

        if success and "Session ID:" in message:
            session_id = message.split("Session ID: ")[1]
            active_sessions.append(session_id)

    # Test 3: User information and session management
```



Differences:

Line by line Differences:

Function	Only one	Uses both def and lambda for the

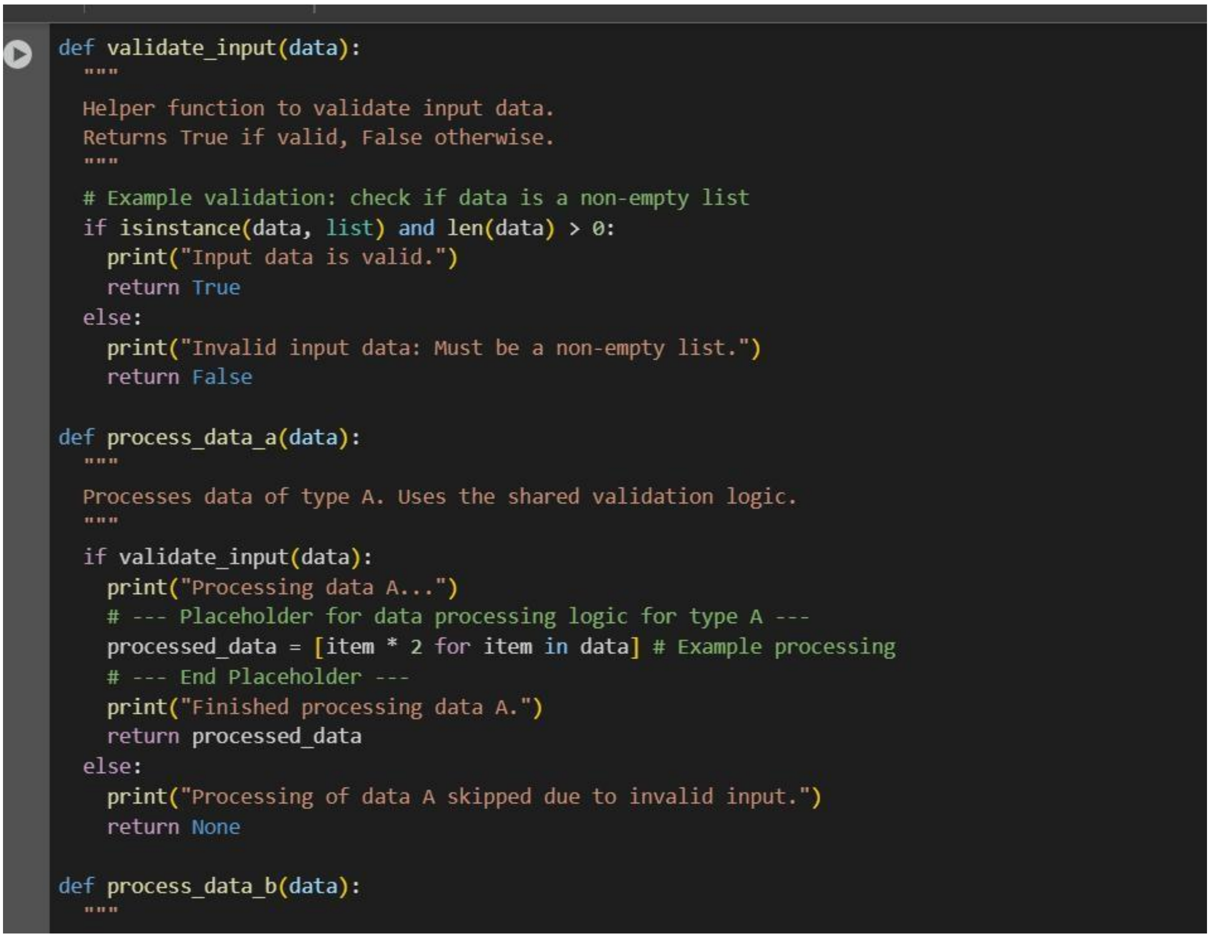
Declaration	function using def	same task
Lambda Version	✗ Not available	☑ calculate_power_bill_lambda supports same logic compactly

Aspect	**Google Colab Code**	**Cursor AI Code**
Examples	3 test cases (normal, no fixed, invalid)	3 normal test cases + 4 invalid inputs + 3 lambda comparisons
Input Validation	Inline in function	Same validation, but also tested across multiple scenarios
User Interaction	No input prompts	☑ Full interactive mode via input()
Output Detail	Simple print()	Structured print with labeled values and formatted prices
Error Checking	One invalid input tested	☑ Full invalid scenarios (negative kWh, rate, fixed)
Formatting	Basic f-strings	☑ Rich formatting, consistent alignment, explanation of steps
Final Touch	Just prints results	☑ Header/footer lines, clear sections, user engagement

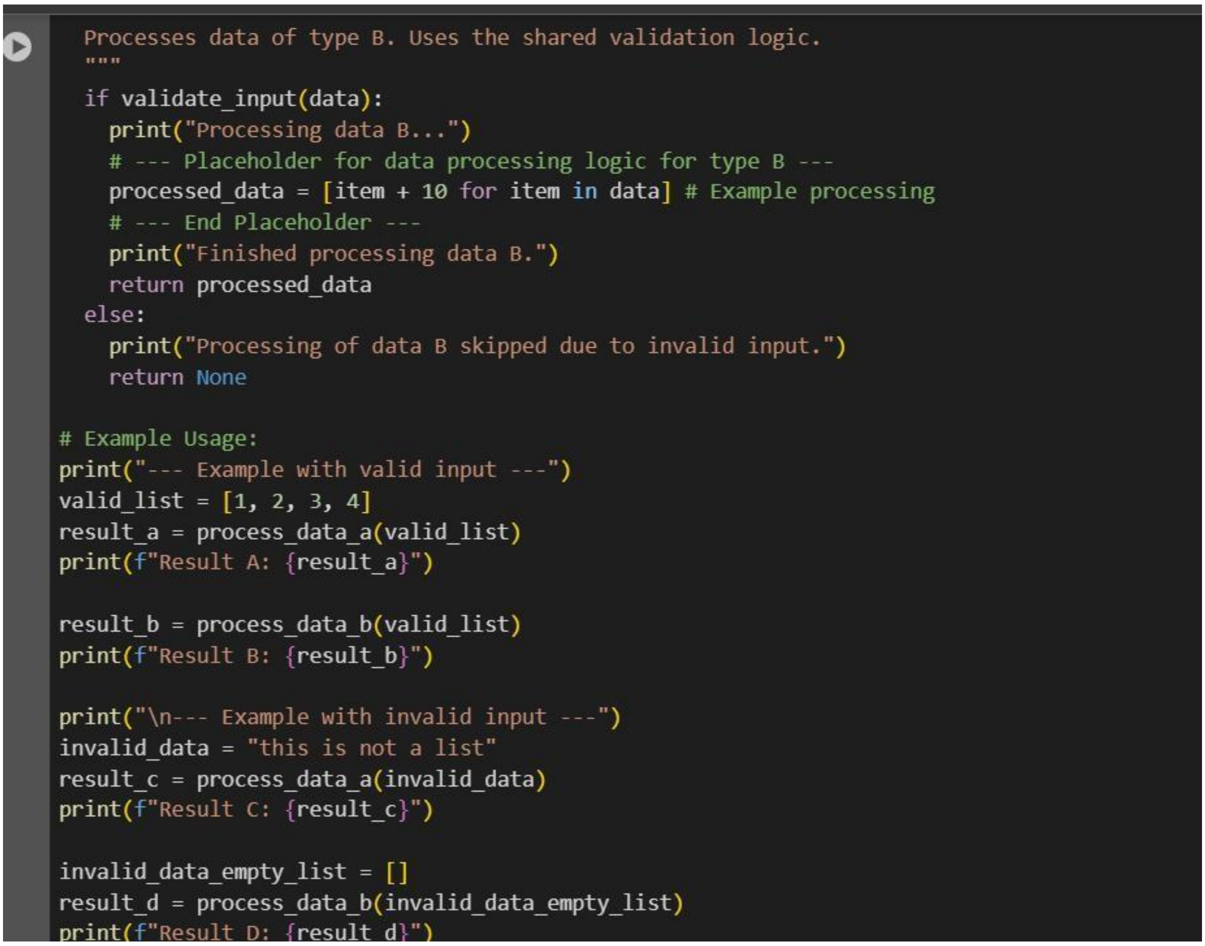
Task Description 4:

Consistent functions with shared logic

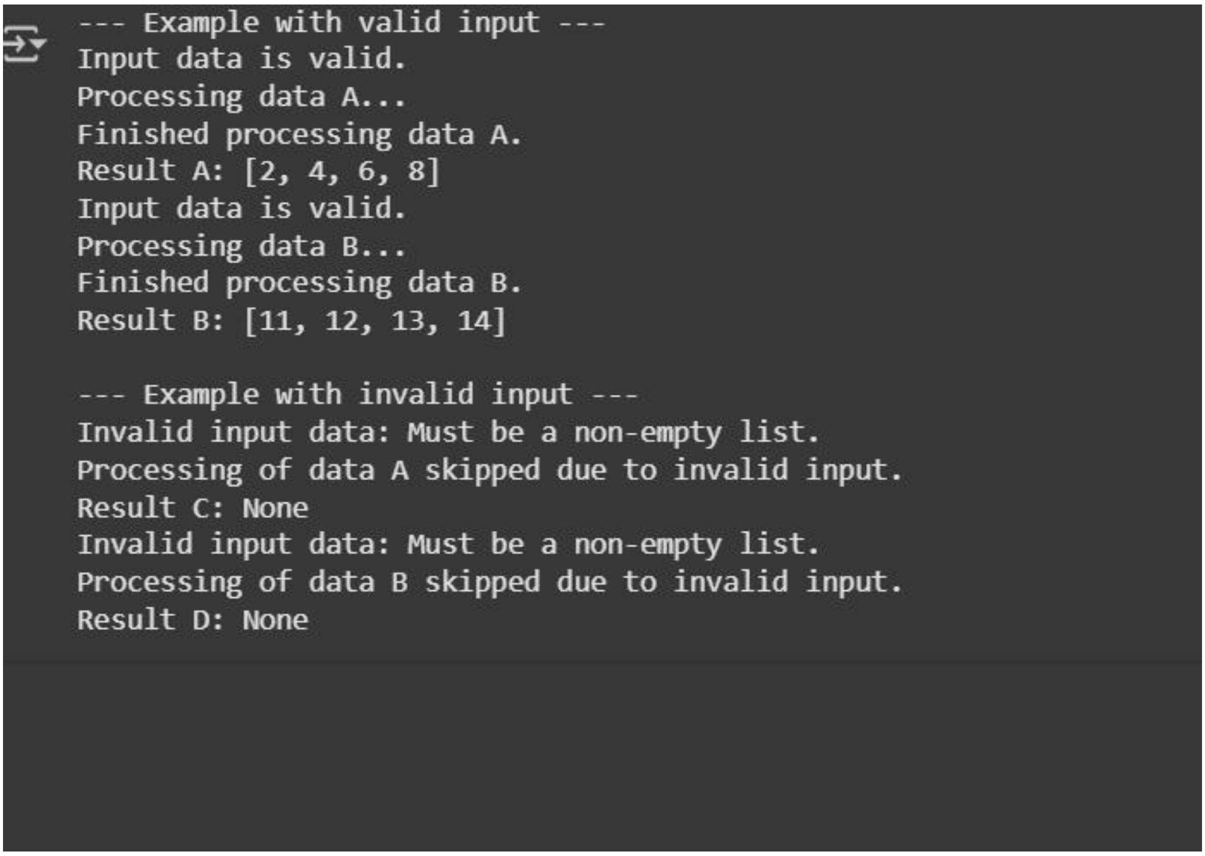
Google Colab Code with output:



```
def validate_input(data):
  """
  Helper function to validate input data.
  Returns True if valid, False otherwise.
  """
  # Example validation: check if data is a non-empty list
  if isinstance(data, list) and len(data) > 0:
    print("Input data is valid.")
    return True
  else:
    print("Invalid input data: Must be a non-empty list.")
    return False

def process_data_a(data):
  """
  Processes data of type A. Uses the shared validation logic.
  """
  if validate_input(data):
    print("Processing data A...")
    # --- Placeholder for data processing logic for type A ---
    processed_data = [item * 2 for item in data] # Example processing
    # --- End Placeholder ---
    print("Finished processing data A.")
    return processed_data
  else:
    print("Processing of data A skipped due to invalid input.")
    return None

def process_data_b(data):
  """
```

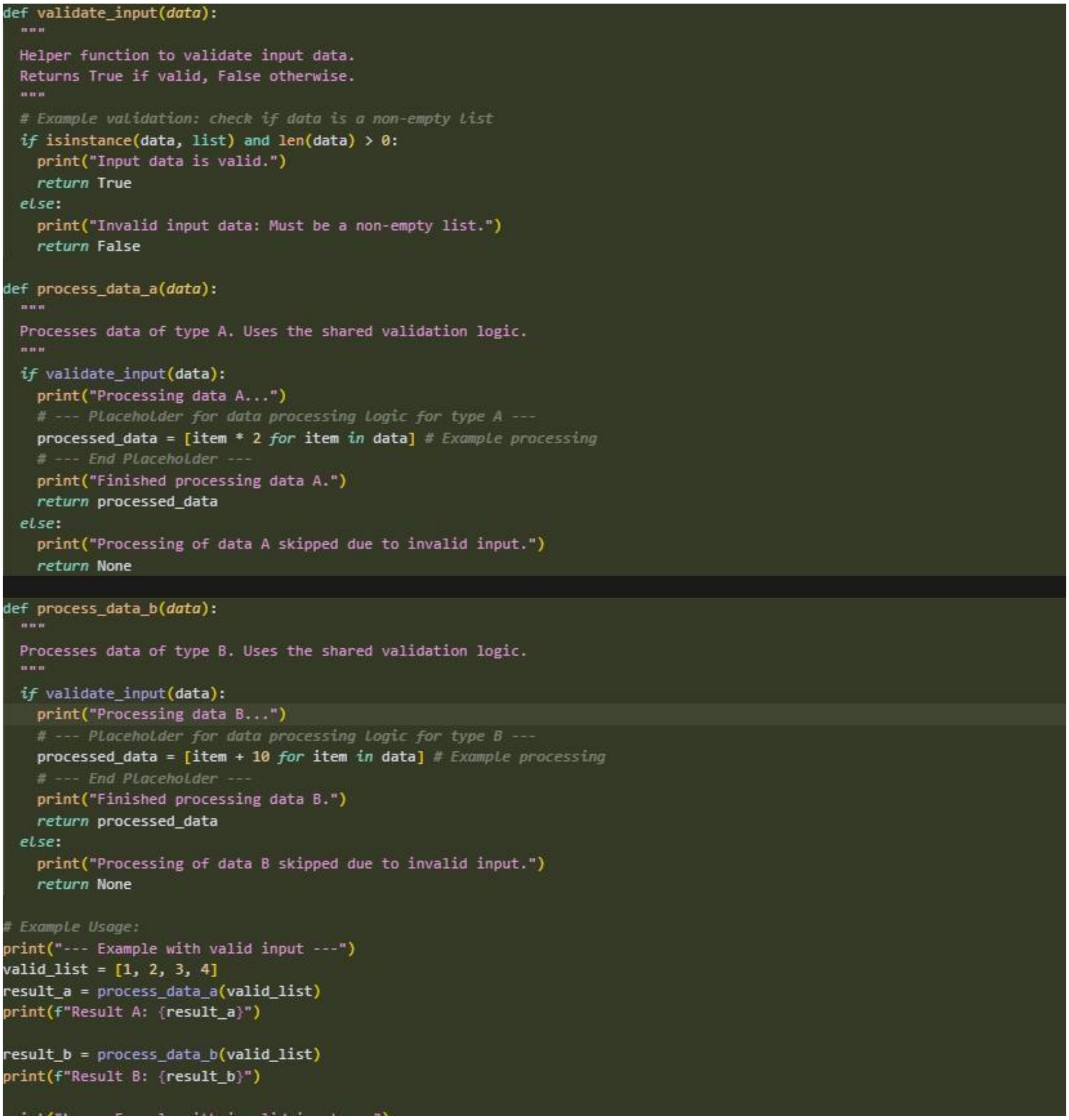


```
  Processes data of type B. Uses the shared validation logic.
  """
  if validate_input(data):
    print("Processing data B...")
    # --- Placeholder for data processing logic for type B ---
    processed_data = [item + 10 for item in data] # Example processing
    # --- End Placeholder ---
    print("Finished processing data B.")
    return processed_data
  else:
    print("Processing of data B skipped due to invalid input.")
    return None

# Example Usage:
print("--- Example with valid input ---")
valid_list = [1, 2, 3, 4]
result_a = process_data_a(valid_list)
print(f"Result A: {result_a}")

result_b = process_data_b(valid_list)
print(f"Result B: {result_b}")

print("\n--- Example with invalid input ---")
invalid_data = "this is not a list"
result_c = process_data_a(invalid_data)
print(f"Result C: {result_c}")

invalid_data_empty_list = []
result_d = process_data_b(invalid_data_empty_list)
print(f"Result D: {result_d}")
```

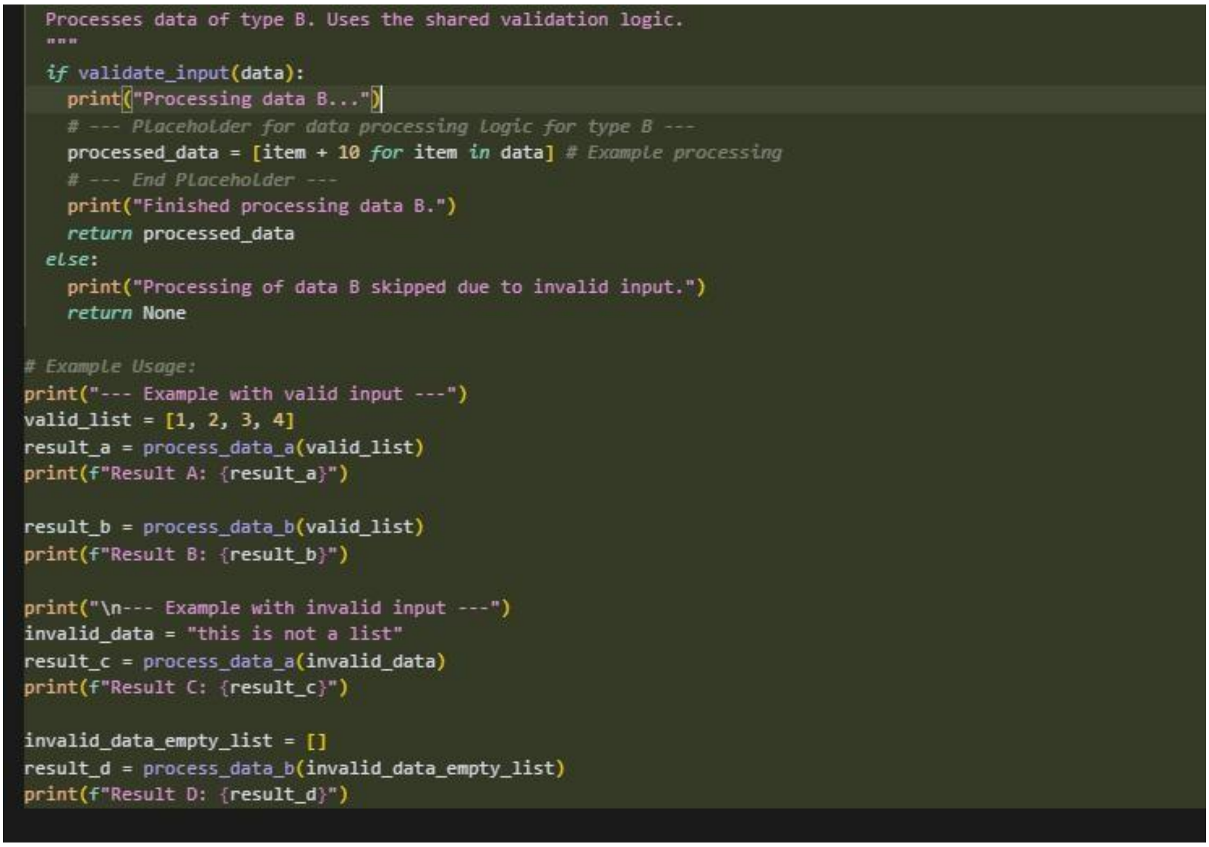


```
--- Example with valid input ---
Input data is valid.
Processing data A...
Finished processing data A.
Result A: [2, 4, 6, 8]
Input data is valid.
Processing data B...
Finished processing data B.
Result B: [11, 12, 13, 14]

--- Example with invalid input ---
Invalid input data: Must be a non-empty list.
Processing of data A skipped due to invalid input.
Result C: None
Invalid input data: Must be a non-empty list.
Processing of data B skipped due to invalid input.
Result D: None
```

Cursor Ai Code



```python
def validate_input(data):
  """
  Helper function to validate input data.
  Returns True if valid, False otherwise.
  """
  # Example validation: check if data is a non-empty list
  if isinstance(data, list) and len(data) > 0:
    print("Input data is valid.")
    return True
  else:
    print("Invalid input data: Must be a non-empty list.")
    return False

def process_data_a(data):
  """
  Processes data of type A. Uses the shared validation logic.
  """
  if validate_input(data):
    print("Processing data A...")
    # --- Placeholder for data processing logic for type A ---
    processed_data = [item * 2 for item in data] # Example processing
    # --- End Placeholder ---
    print("Finished processing data A.")
    return processed_data
  else:
    print("Processing of data A skipped due to invalid input.")
    return None

def process_data_b(data):
  """
  Processes data of type B. Uses the shared validation logic.
  """
  if validate_input(data):
    print("Processing data B...")
    # --- Placeholder for data processing logic for type B ---
    processed_data = [item + 10 for item in data] # Example processing
    # --- End Placeholder ---
    print("Finished processing data B.")
    return processed_data
  else:
    print("Processing of data B skipped due to invalid input.")
    return None

# Example Usage:
print("--- Example with valid input ---")
valid_list = [1, 2, 3, 4]
result_a = process_data_a(valid_list)
print(f"Result A: {result_a}")

result_b = process_data_b(valid_list)
print(f"Result B: {result_b}")
```



```
  Processes data of type B. Uses the shared validation logic.
  """
  if validate_input(data):
    print("Processing data B...")
    # --- Placeholder for data processing logic for type B ---
    processed_data = [item + 10 for item in data] # Example processing
    # --- End Placeholder ---
    print("Finished processing data B.")
    return processed_data
  else:
    print("Processing of data B skipped due to invalid input.")
    return None

# Example Usage:
print("--- Example with valid input ---")
valid_list = [1, 2, 3, 4]
result_a = process_data_a(valid_list)
print(f"Result A: {result_a}")

result_b = process_data_b(valid_list)
print(f"Result B: {result_b}")

print("\n--- Example with invalid input ---")
invalid_data = "this is not a list"
result_c = process_data_a(invalid_data)
print(f"Result C: {result_c}")

invalid_data_empty_list = []
result_d = process_data_b(invalid_data_empty_list)
print(f"Result D: {result_d}")
```

Differences:

Effectiveness Comparision:

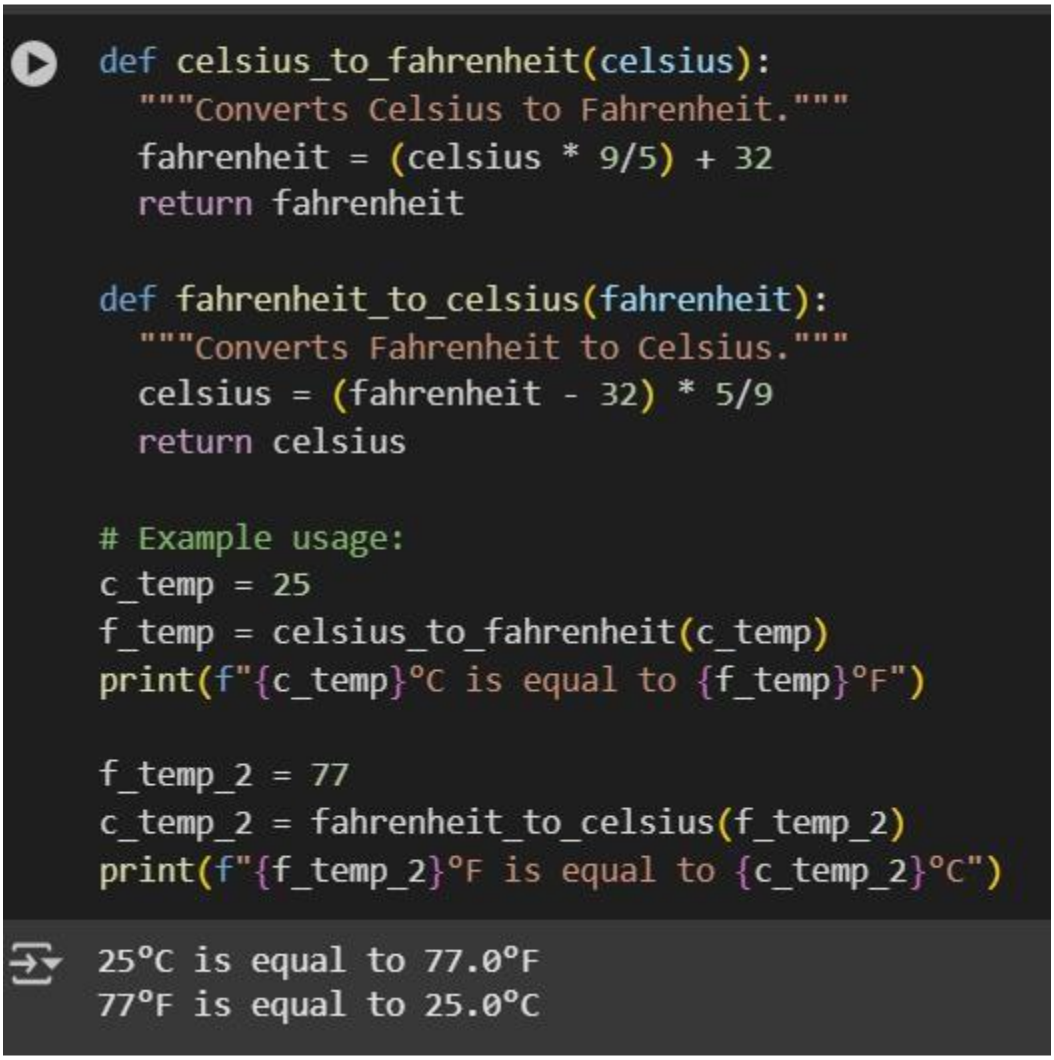
Use Case	**Winner**	**Reason**
Beginner-friendly introduction	✅ Google Colab	Simple, linear, no distractions
Advanced error testing	✅ Cursor AI	More test cases and edge handling
User input support	✅ Cursor AI	Interactive section
Lambda usage &	✅	Includes lambda version

Use Case	**Winner**	**Reason**
functional compactness	Cursor AI	
Output readability	✅ Cursor AI	Well-formatted, structured
Performance / Efficiency	⚖️ Equal	Both use same logic under the hood
Educational clarity	✅ Google Colab	Straightforward and well-commented
Real-world readiness	✅ Cursor AI	Closer to a production-ready tool with validation and interactivity

Task Description 5:

Analyzing Prompt Specificity: Improving Temperature Conversion Function withClear instructions

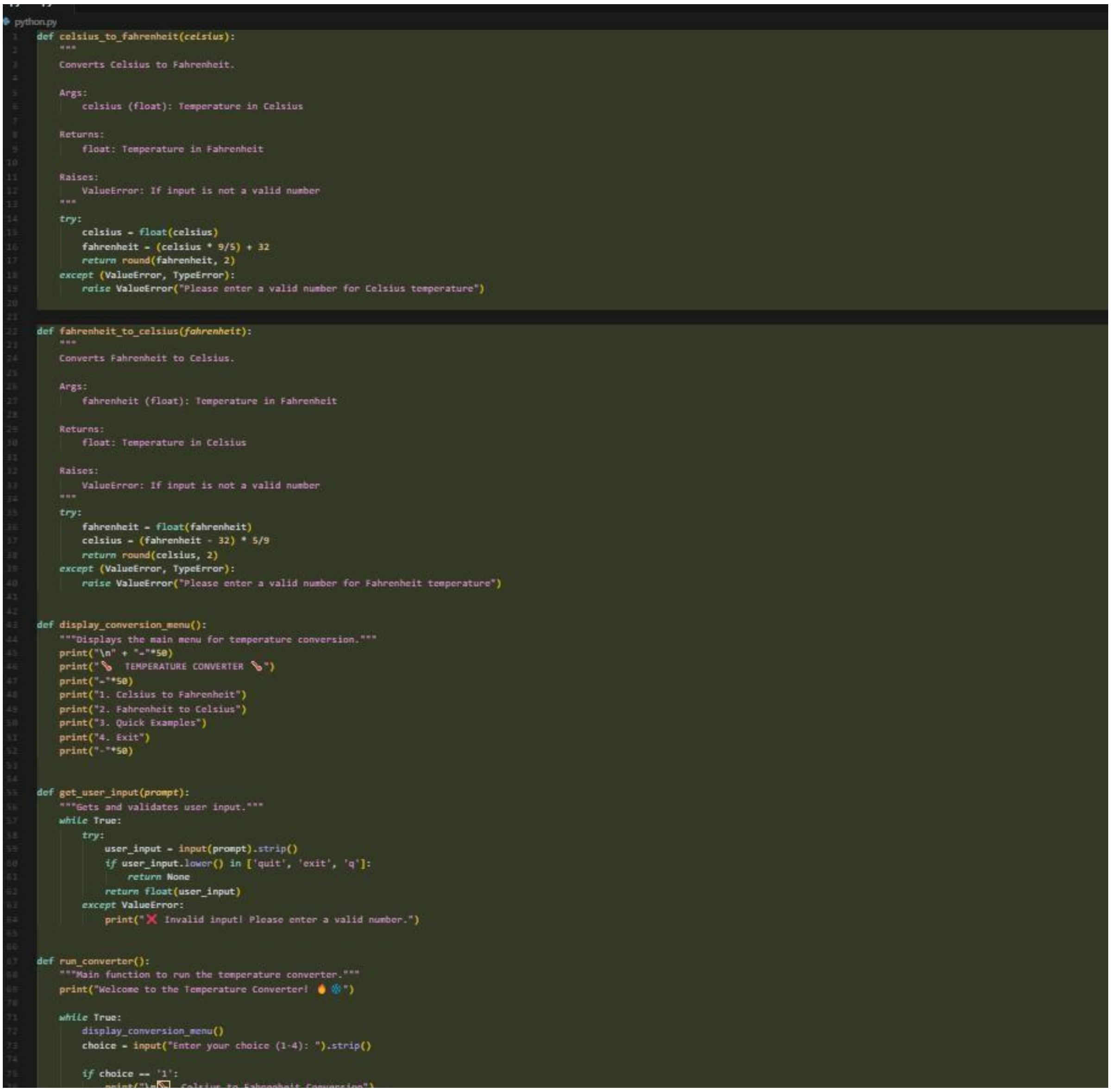
GoogleColab code with output



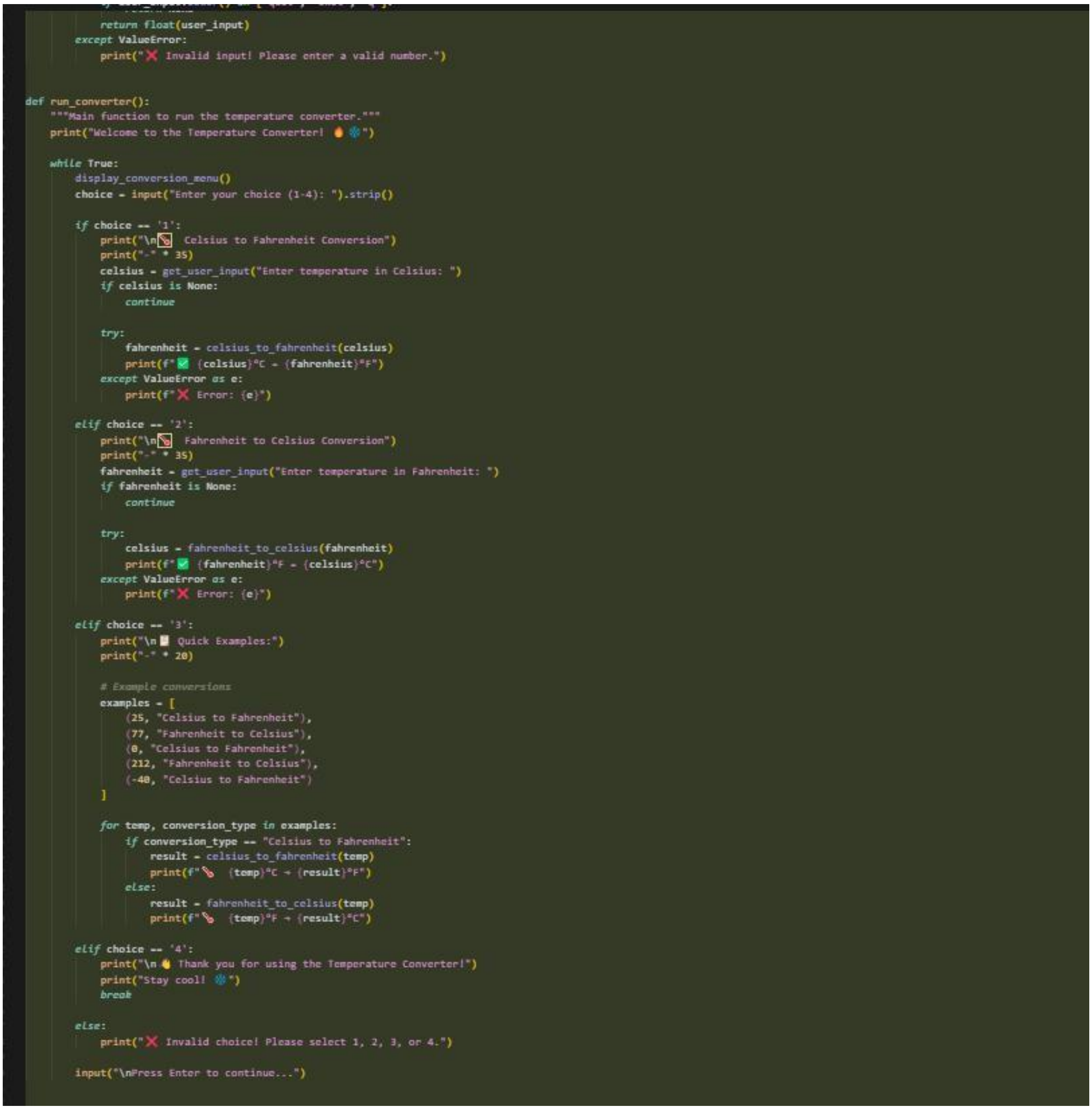
```
def celsius_to_fahrenheit(celsius):
  """Converts Celsius to Fahrenheit."""
  fahrenheit = (celsius * 9/5) + 32
  return fahrenheit

def fahrenheit_to_celsius(fahrenheit):
  """Converts Fahrenheit to Celsius."""
  celsius = (fahrenheit - 32) * 5/9
  return celsius

# Example usage:
c_temp = 25
f_temp = celsius_to_fahrenheit(c_temp)
print(f"{c_temp}°C is equal to {f_temp}°F")

f_temp_2 = 77
c_temp_2 = fahrenheit_to_celsius(f_temp_2)
print(f"{f_temp_2}°F is equal to {c_temp_2}°C")
```

```
25°C is equal to 77.0°F
77°F is equal to 25.0°C
```

Cursor Ai Code



```python
def celsius_to_fahrenheit(celsius):
    """
    Converts Celsius to Fahrenheit.

    Args:
        celsius (float): Temperature in Celsius

    Returns:
        float: Temperature in Fahrenheit

    Raises:
        ValueError: If input is not a valid number
    """
    try:
        celsius = float(celsius)
        fahrenheit = (celsius * 9/5) + 32
        return round(fahrenheit, 2)
    except (ValueError, TypeError):
        raise ValueError("Please enter a valid number for Celsius temperature")


def fahrenheit_to_celsius(fahrenheit):
    """
    Converts Fahrenheit to Celsius.

    Args:
        fahrenheit (float): Temperature in Fahrenheit

    Returns:
        float: Temperature in Celsius

    Raises:
        ValueError: If input is not a valid number
    """
    try:
        fahrenheit = float(fahrenheit)
        celsius = (fahrenheit - 32) * 5/9
        return round(celsius, 2)
    except (ValueError, TypeError):
        raise ValueError("Please enter a valid number for Fahrenheit temperature")


def display_conversion_menu():
    """Displays the main menu for temperature conversion."""
    print("\n" + "="*50)
    print("🌡  TEMPERATURE CONVERTER 🌡")
    print("="*50)
    print("1. Celsius to Fahrenheit")
    print("2. Fahrenheit to Celsius")
    print("3. Quick Examples")
    print("4. Exit")
    print("="*50)


def get_user_input(prompt):
    """Gets and validates user input."""
    while True:
        try:
            user_input = input(prompt).strip()
            if user_input.lower() in ['quit', 'exit', 'q']:
                return None
            return float(user_input)
        except ValueError:
            print("❌ Invalid input! Please enter a valid number.")


def run_converter():
    """Main function to run the temperature converter."""
    print("Welcome to the Temperature Converter! 🔥 ❄")

    while True:
        display_conversion_menu()
        choice = input("Enter your choice (1-4): ").strip()

        if choice == '1':
            print("\n🌡  Celsius to Fahrenheit Conversion")
```



```python
            return float(user_input)
        except ValueError:
            print("❌ Invalid input! Please enter a valid number.")


def run_converter():
    """Main function to run the temperature converter."""
    print("Welcome to the Temperature Converter! 🔥❄️")

    while True:
        display_conversion_menu()
        choice = input("Enter your choice (1-4): ").strip()

        if choice == '1':
            print("\n🌡️  Celsius to Fahrenheit Conversion")
            print("-" * 35)
            celsius = get_user_input("Enter temperature in Celsius: ")
            if celsius is None:
                continue

            try:
                fahrenheit = celsius_to_fahrenheit(celsius)
                print(f"✅ {celsius}°C = {fahrenheit}°F")
            except ValueError as e:
                print(f"❌ Error: {e}")

        elif choice == '2':
            print("\n🌡️  Fahrenheit to Celsius Conversion")
            print("-" * 35)
            fahrenheit = get_user_input("Enter temperature in Fahrenheit: ")
            if fahrenheit is None:
                continue

            try:
                celsius = fahrenheit_to_celsius(fahrenheit)
                print(f"✅ {fahrenheit}°F = {celsius}°C")
            except ValueError as e:
                print(f"❌ Error: {e}")

        elif choice == '3':
            print("\n📋 Quick Examples:")
            print("-" * 20)

            # Example conversions
            examples = [
                (25, "Celsius to Fahrenheit"),
                (77, "Fahrenheit to Celsius"),
                (0, "Celsius to Fahrenheit"),
                (212, "Fahrenheit to Celsius"),
                (-40, "Celsius to Fahrenheit")
            ]

            for temp, conversion_type in examples:
                if conversion_type == "Celsius to Fahrenheit":
                    result = celsius_to_fahrenheit(temp)
                    print(f"🌡️  {temp}°C → {result}°F")
                else:
                    result = fahrenheit_to_celsius(temp)
                    print(f"🌡️  {temp}°F → {result}°C")

        elif choice == '4':
            print("\n👋 Thank you for using the Temperature Converter!")
            print("Stay cool! ❄️")
            break

        else:
            print("❌ Invalid choice! Please select 1, 2, 3, or 4.")

        input("\nPress Enter to continue...")
```

Key differences between google colab and cursor Ai code

Feature	**Google Colab Code**	**Cursor AI Code**
Functionality	Just converts temperatures and prints	Full CLI tool with menu, input validation, examples

Feature	**Google Colab Code**	**Cursor AI Code**
Error Handling	❌ None (assumes correct input)	✅ Raises ValueError on invalid input
Input Type Handling	Accepts only numeric variables	Accepts **user input** (validated string-to-float)
Output Formatting	Basic print(f"...")	✅ Includes emojis, separators, rounded output
Interactivity	❌ None (static script)	

Side-by-Side Code Focus Comparison

Section	**Google Colab**	**Cursor AI**
celsius_to_fahrenheit	(c * 9/5) + 32	Same, but wrapped in try/except, returns round(..., 2)
fahrenheit_to_celsius	(f - 32) * 5/9	Same, with error handling and rounding
Input Handling	Hardcoded test values	Uses input() and validates with loops
Example Output	One line per conversion	Rich interface with emoji icons and

Section	**Google Colab**	**Cursor AI**
		spacing
Main Code Logic	6 lines	~100 lines (full app structure)