

AI ASSISTED CODING

Lab Assignment 3.3

Roll no: 2403A510C2

Batch: 05

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

▶ 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

```

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:
        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:

```
▶ 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:

```
Σ 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)
        except ValueError:
            print(f"ValueError for num={num}: {str(ValueError)}")
        else:
            print(f"Recursive Result: {recursive_result}, Iterative Result: {iterative_result}, Math Result: {math_result}")
```

```

        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}")
        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:

```
▶ # 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:

```
# 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:
    print("KeyboardInterrupt")
```

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

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 beginners.
- Efficient** — Uses a loop, no recursion stack involved.
- Simple error handling** — Catches negative numbers easily.
- 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 lambda
- 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 inputs.
- Debugging** is harder with deeply nested lambdas.
- Performance** is slower than iterative loops.

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	<input checked="" type="checkbox"/> Beginner-friendly	<input checked="" type="checkbox"/> Complex for new learners
Reusability	<input checked="" type="checkbox"/> Unless functionized	<input checked="" type="checkbox"/> Easy to reuse functions

Metric	Google Colab Code	Cursor AI Code
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_listAscending(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_listAscending(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_listAscending(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))(0))))()

# Sort list function using Lambda (no def)
sort_listAscending = 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"\nNegative 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"\nOriginal list: {my_list}")

sorted_list = sort_listAscending(my_list.copy()) # Using copy to preserve original
print(f"\nSorted 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_listAscending(another_list)
print(f"\nAfter 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_listAscending(lst.copy())
    print(f"\nAscending: {ascending_result}")

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

    # Create new sorted list (original unchanged)
    new_sorted = sort_list_copy(lst)
    print(f"\nNew 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"\nFactorial of {user_input} is: {result}")
    else:
        print("Error: Factorial is not defined for negative numbers")

```

```
72     print(f" New sorted list: {new_sorted}")
73
74 # Interactive example
75 print("\n==== Interactive Example ===")
76 try:
77     user_input = int(input("Enter a number to calculate factorial: "))
78     result = factorial_iterative(user_input)
79     if result is not None:
80         print(f"Factorial of {user_input} is: {result}")
81     else:
82         print("Error: Factorial is not defined for negative numbers")
83 except ValueError:
84     print("Error: Please enter a valid integer")
85 except KeyboardInterrupt:
86     print("\nProgram interrupted by user.")
87
```

^ 2 / 2 Undo all Ctrl+Shift+G Keep all Ctrl+F Filter

Problems Output Debug Console Terminal Ports

[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 and 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)

Comparative 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

Aspect	Google Colab Code	Cursor AI Code
Best For	Beginners, education, real-world code	Experts, demos, compact code challenges

Task Description 3:

Start with the vague prompt “Generate python code to calculate power bill” and improve 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 negative

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, fixed_standard)
print(f"Standard residential bill:")
```

▲ 2 / 2 ▼ Undo all Ctrl+Shift+Q Keep all Ctrl+D

```
thon.py
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("\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("\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")
]
```

^ 2 / 2 Undo all Ctrl+Shift+QD Keep all Ctrl+Q

```

81 ]
82
83 for kwh, rate, fixed, description in lambda_test_cases:
84     result_lambda = calculate_power_bill_lambda(kwh, rate, fixed)
85     result_def = calculate_power_bill(kwh, rate, fixed)
86     print(f"{description}:")
87     print(f" Lambda result: ${result_lambda:.2f}")
88     print(f" Def result: ${result_def:.2f}")
89     print(f" Match: {result_lambda == result_def}")
90
91 # Interactive power bill calculator
92 print(f"\n==== Interactive Power Bill Calculator ====")
93 try:
94     print("Enter your power consumption details:")
95     user_kwh = float(input("Kilowatt-hours consumed: "))
96     user_rate = float(input("Rate per kWh ($): "))
97     user_fixed = float(input("Fixed charge ($) [press Enter for 0]: ") or 0)
98
99     user_bill = calculate_power_bill(user_kwh, user_rate, user_fixed)
100
101    if isinstance(user_bill, str):
102        print(f"Error: {user_bill}")
103    else:
104        print("\nYour power bill breakdown:")
105        print(f" Energy consumption: {user_kwh} kWh | ${user_rate:.2f}/kWh = ${user_kwh * user_rate:.2f}")
106        print(f" Fixed charge: ${user_fixed:.2f}")
107        print(f" Total bill: ${user_bill:.2f}")
108
109 except ValueError:
110     print("Error: Please enter valid numeric values.")
111 except KeyboardInterrupt:
112     print("\nCalculator interrupted by user.")
113
114 print(f"\n" + "="*60)
115 print("POWER BILL CALCULATION COMPLETE")
116 print("=".*60)
117

```

^ 2 / 2 ▾ Undo all Ctrl+Shift+D Keep all Ctrl+I

Problems Output Debug Console Terminal Ports Filter

```
[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
Reusability	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 ---
```

```
Commands | + Code + Text | ▶ Run all ▾
# --- 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:

```
def register_user(cursor, user_name, user_password):
    cursor.execute("INSERT INTO users (username, password) VALUES (%s, %s)", (user_name, user_password))
    print(f"User '{user_name}' registered successfully (placeholder).")

def login_user(cursor, user_name, user_password):
    cursor.execute("SELECT * FROM users WHERE username = %s AND password = %s", (user_name, user_password))
    user_data = cursor.fetchone()
    if user_data:
        print(f"User '{user_name}' logged in successfully (placeholder).")
        return True
    else:
        print("Invalid username or password (placeholder).")
        return False

def main():
    # Initialize database connection
    conn = psycopg2.connect(dbname='mydb', user='myuser', password='mypassword', host='localhost', port=5432)
    cursor = conn.cursor()

    # Register new user
    register_user(cursor, 'new_user', 'new_password')

    # Log in new user
    if login_user(cursor, 'new_user', 'new_password'):
        print("Login successful for new user: True")
    else:
        print("Login successful for failed attempt: False")

    # Log in test user
    if login_user(cursor, 'test_user', 'test_password'):
        print("Login successful for test user: True")
    else:
        print("Login successful for failed attempt: False")

    # Log in incorrect credentials
    if login_user(cursor, 'test_user', 'incorrect_password'):
        print("Login successful for failed attempt: False")
    else:
        print("Login successful for failed attempt: True")

    # Close database connection
    cursor.close()
    conn.close()

if __name__ == "__main__":
    main()
```



```

["info_out", "msg_PowerBill", "msg_error"]
["consumed_user", "consumedID", "Non-existent user"]
["info_out", "consumedID", "Another valid logic"]
["info_out", "consumedID", "Empty consumer"]
["info_out", "empty"]
[1]

active_sessions = []
for session, password, description in auth_sessions:
    print("In description:")
    session, message = auth_login_user(session, password)
    print("Result: " + "Success" if session else "Failed")
    print("Message: " + message)

if success and message == "Success":
    session_id = message[message.rfind("Session ID: ") + 14]
    active_sessions.append(session_id)

# Test to show information and session management
print("User with INFORMATION & SESSION MANAGEMENT user")

print("Unregistered users: " + auth_list_users())
print("Active sessions: " + auth_get_active_sessions())

# Check user login
for user_id in auth_list_users():
    user_info = auth_get_user(user_id)
    print("User info for: " + user_info['name'])
    for key, value in user_info.items():
        print(key + " : " + str(value))

# Test of logout function
print("Logout user")

# Session ID in active_sessions
success, message = auth_login_user(session_id)
print("Logout (session_id): " + "Success" if success else "Failed")
print("Message: " + message)

print("Selective sessions after Logout: " + auth_get_active_sessions())

# Session ID
#-----#
#-----#_name__= "user_id"
#-----#_user power Bill calculations
print("n=" + "n+100")
print("Power Bill CALCULATION OVER")
print("n=" + "n+100")

# Example usage with different scenarios
print("User with Power Bill Examples user")

# Example 1: Standard residential usage
bill, rate, fixed_charge = 100
rate_lowest = 0.12 # $0.12 per kWh
fixed_lowest = 10 # $10 fixed charge
bill_lowest = calculate_power_bill(bill, rate_lowest, rate_lowest, fixed_lowest)
print("Standard residential bill: " + str(bill))
print("Consumption: " + str(bill_lowest))
print("Rate: " + str(rate_lowest) + " per kWh")
print("Fixed charge: " + str(fixed_lowest))
print("Total bill: " + str(bill_lowest))

# Example 2: No fixed charge
bill, rate, fixed_charge = 100, 0.12, 0
bill_no_fix = calculate_power_bill(bill, rate, fixed_charge)
print("Without fixed charge: " + str(bill_no_fix))

# Example 3: Consumption scenario
bill_low = 100
rate_low = 0.12
fixed_low = 10
bill_low_low = calculate_power_bill(bill_low, rate_low, fixed_low)
print("Low consumption scenario")
print("Consumption: " + str(bill_low_low))
print("Rate: " + str(rate_low))
print("Fixed charge: " + str(fixed_low))
print("Total bill: " + str(bill_low_low))

# Example 4: Invalid input handling
print("User with Error Handling Examples user")
invalid_name = ["-100", "0.15", "10", "Negative consumption"], \
            ["100", "-0.15", "10", "Negative rate"], \
            ["100", "0.15", "-10", "Negative fixed charge"], \
            ["-100", "0.15", "-10", "All negative values"]
[1]

for bill, rate, fixed, description in invalid_name:
    result = calculate_power_bill(bill, rate, fixed)
    print(description + ": " + str(result))

# Example function testing
print("User with Function Testing user")
testable_bills = [100, 0.12, 10, "Low consumption"], \
                [100, 0.15, 10, "Medium consumption"], \
                [100, 0.18, 10, "High consumption"], \
                [1000, 0.15, 10, "Very high consumption"]

for bill, rate, fixed, description in testable_bills:
    result = calculate_power_bill(bill, rate, fixed)
    print(description + ": " + str(result))

# Residential power Bill calculator
print("User with Residential Power Bill Calculator user")
http://localhost:5000/calculate_power_bill
print("Enter your consumption details")
user_bill = float(input("Electricity hours consumed: "))
user_rate = float(input("Rate per kWh (k): "))
user_fixed = float(input("Fixed charge (F) [press Enter for 0]: ") or 0)
user_M11 = calculate_power_bill(user_bill, user_rate, user_fixed)

if len(str(user_M11)) == 1:
    print("Error: Please enter valid numeric values!")
else:
    print("Your power bill breakdown:")
    print("Energy consumption: " + str(user_bill) + " kWh * " + str(user_rate) + " kP = " + str(user_bill * user_rate) + "P")
    print("Fixed charge: " + str(user_fixed) + "F")
    print("Total bill: " + str(user_M11))

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

print("n=" + "n+100")
print("Power Bill CALCULATION OVER")
print("n=" + "n+100")

# User authentication demonstration
authenticate_authentication()
print("n=" + "n+100")
print("ALL SYSTEM COMPLETE")
print("n=" + "n+100")

```

Differences:

Line by line Differences:

Function Declaration	Only one function using def	Uses both def and lambda for the same task
Lambda Version	✗ Not available	<input checked="" type="checkbox"/> 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	<input checked="" type="checkbox"/> Full interactive mode via input()
Output Detail	Simple print()	Structured print with labeled values and formatted prices
Error Checking	One invalid input tested	<input checked="" type="checkbox"/> Full invalid scenarios (negative kWh, rate, fixed)
Formatting	Basic f-strings	<input checked="" type="checkbox"/> Rich formatting, consistent alignment, explanation of steps
Final Touch	Just prints results	<input checked="" type="checkbox"/> 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

```
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

Use Case	Winner	Reason
Lambda usage & functional compactness	Cursor AI 	Includes lambda version
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 with Clear 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.py
1 def celsius_to_fahrenheit(celsius):
2     """
3         Converts Celsius to Fahrenheit.
4
5     Args:
6         celsius (float): Temperature in Celsius
7
8     Returns:
9         float: Temperature in Fahrenheit
10
11    Raises:
12        ValueError: If input is not a valid number
13
14    try:
15        celsius = float(celsius)
16        fahrenheit = (celsius * 9/5) + 32
17        return round(fahrenheit, 2)
18    except (ValueError, TypeError):
19        raise ValueError("Please enter a valid number for Celsius temperature")
20
21
22 def fahrenheit_to_celsius(fahrenheit):
23     """
24         Converts Fahrenheit to Celsius.
25
26     Args:
27         fahrenheit (float): Temperature in Fahrenheit
28
29     Returns:
30         float: Temperature in Celsius
31
32    Raises:
33        ValueError: If input is not a valid number
34
35    try:
36        fahrenheit = float(fahrenheit)
37        celsius = (fahrenheit - 32) * 5/9
38        return round(celsius, 2)
39    except (ValueError, TypeError):
40        raise ValueError("Please enter a valid number for Fahrenheit temperature")
41
42
43 def display_conversion_menu():
44     """Displays the main menu for temperature conversion."""
45     print("\n" + "*50)
46     print("TEMPERATURE CONVERTER")
47     print("-*50)
48     print("1. Celsius to Fahrenheit")
49     print("2. Fahrenheit to Celsius")
50     print("3. Quick Examples")
51     print("4. Exit")
52     print("-*50)
53
54
55 def get_user_input(prompt):
56     """Gets and validates user input."""
57     while True:
58         try:
59             user_input = input(prompt).strip()
60             if user_input.lower() in ['quit', 'exit', 'q']:
61                 return None
62             return float(user_input)
63         except ValueError:
64             print("X Invalid input! Please enter a valid number.")
65
66
67 def run_converter():
68     """Main function to run the temperature converter."""
69     print("Welcome to the Temperature Converter! 🔥")
70
71     while True:
72         display_conversion_menu()
73         choice = input("Enter your choice (1-4): ").strip()
74
75         if choice == '1':
76             print("Celsius to Fahrenheit Conversion")
```

```

def get_user_input():
    user_input = input("Enter temperature: ")
    try:
        return float(user_input)
    except ValueError:
        print("X 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("\nCelsius 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"X Error: {e}")

        elif choice == '2':
            print("\nFahrenheit 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"X Error: {e}")

        elif choice == '3':
            print("\nQuick 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("\nThank you for using the Temperature Converter!")
            print("Stay cool! ☀️")
            break

        else:
            print("X 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 spacing

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