

<b>SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE</b>		<b>DEPARTMENT OF COMPUTER SCIENCE ENGINEERING</b>	
<b>ProgramName:</b> B. Tech		<b>Assignment Type:</b> Lab	<b>AcademicYear:</b> 2025-2026
<b>CourseCoordinatorName</b>		Venkataramana Veeramsetty	
<b>Instructor(s)Name</b>		1. Dr. Mohammed Ali Shaik 2. Dr. T Sampath Kumar 3. Mr. S Naresh Kumar 4. Dr. V. Rajesh 5. Dr. Brij Kishore 6. Dr Pramoda Patro 7. Dr. Venkataramana 8. Dr. Ravi Chander 9. Dr. Jagjeeth Singh	
<b>CourseCode</b>	24CS002PC215	<b>CourseTitle</b>	AI Assisted Coding
<b>Year/Sem</b>	II/I	<b>Regulation</b>	R24
<b>Date and Day of Assignment</b>		<b>Time(s)</b>	
<b>Duration</b>	2 Hours	<b>Applicableto Batches</b>	
<b>AssignmentNumber:</b> 3.3(Present assignment number)/24(Total number of assignments)			

Q.No.	Question	Expected Time to complete
1	<p>Lab 3: Prompt Engineering – Improving Prompts and Context Management</p> <p><b>Lab Objectives:</b></p> <ul style="list-style-type: none"> <li>To understand how prompt structure and wording influence AI-generated code.</li> <li>To explore how context (like comments and function names) helps AI generate relevant output.</li> <li>To evaluate the quality and accuracy of code based on prompt clarity.</li> <li>To develop effective prompting strategies for AI-assisted programming.</li> </ul> <p><b>Lab Outcomes (LOs):</b> After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> <li>Generate Python code using Google Gemini in Google Colab.</li> <li>Analyze the effectiveness of code explanations and suggestions by Gemini.</li> <li>Set up and use Cursor AI for AI-powered coding assistance.</li> <li>Evaluate and refactor code using Cursor AI features.</li> <li>Compare AI tool behavior and code quality across different platforms.</li> </ul>	03.08.2025 EOD

### Task Description#1

- Try 3 different prompts to generate a factorial function.

**Prompt:**

Generate a factorial function.

Input:

```
Task1.py X
Task1.py > ...
1 def factorial(n):
2     if n < 0:
3         raise ValueError("Factorial is not defined for negative numbers")
4
5     if n == 0 or n == 1:
6         return 1
7
8     result = 1
9     for i in range(2, n + 1):
10         result *= i
11
12     return result
13
14 # Example usage
15 if __name__ == "__main__":
16     # Test the function
17     test_numbers = [0, 1, 5, 10]
18
19     for num in test_numbers:
20         print(f"{num}! = {factorial(num)}")
21
22     # Interactive example
23     try:
24         user_input = int(input("Enter a number to calculate factorial: "))
25         result = factorial(user_input)
26         print(f"{user_input}! = {result}")
27     except ValueError as e:
28         print(f"Error: {e}")
29
30
```

Output:

[illegible]

**Prompt 2:**

write a python program to generate factorial functions.

```

Task1.py task1.2.py
task1.2.py > factorial_iterative
1 import math
2
3 def factorial_iterative(n):
4     if n < 0:
5         raise ValueError("Factorial is not defined for negative numbers")
6
7     if n == 0 or n == 1:
8         return 1
9
10    result = 1
11    for i in range(2, n + 1):
12        result *= i
13
14    return result
15
16 > def factorial_recursive(n): ...
17
18
19 def factorial_math(n):
20     if n < 0:
21         raise ValueError("Factorial is not defined for negative numbers")
22
23     return math.factorial(n)
24
25 def generate_factorial_function(style="iterative"):
26     if style == "iterative":
27         return factorial_iterative
28     elif style == "recursive":
29         return factorial_recursive
30     elif style == "math":
31         return factorial_math
32     else:
33         raise ValueError(f"Unknown style: {style}")
34
35 def compare_methods(n):
36     """Compare all factorial methods."""
37     methods = {
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55 task1.2.py > factorial_iterative
56 if __name__ == "__main__":
57     print("Factorial Function Generator")
58     print("=- * 30)
59
60     test_numbers = [0, 1, 5, 10]
61
62     for num in test_numbers:
63         print(f"\nTesting factorial({num}):")
64         results = compare_methods(num)
65
66         for method, result in results.items():
67             print(f"    {method}: {result}")
68
69     print(f"\nInteractive Function Generation:")
70     print("Available styles: iterative, recursive, math")
71
72     try:
73         style = input("Enter (function) def generate_factorial_function(style: str = 'iterative') -> (((n: Any) -> int) | ((n: Any) ->
74         if style in ["iterati (Any | Literal[1])]]):
75             factorial_func = generate_factorial_function(style)
76             print(f"Generated {style} factorial function")
77
78             test_num = int(input("Enter a number to test: "))
79             result = factorial_func(test_num)
80             print(f"test_num! = {result}")
81         else:
82             print("Invalid style. Using iterative as default.")
83             factorial_func = generate_factorial_function("iterative")
84             test_num = int(input("Enter a number to test: "))
85             result = factorial_func(test_num)
86             print(f"test_num! = {result}")
87     except Exception as e:
88         print(f"Error: {e}")
89
90     print(f"\nAll factorial functions are ready to use!")

```

**Output:**

```

=====

Testing factorial(0):
  Iterative: 1
  Recursive: 1
  Math Module: 1

Testing factorial(1):
  Iterative: 1
  Recursive: 1
  Math Module: 1

Testing factorial(5):
  Iterative: 120
  Recursive: 120
  Math Module: 120

Testing factorial(10):
  Iterative: 3628800
  Recursive: 3628800
  Math Module: 3628800

Interactive Function Generation:
Available styles: iterative, recursive, math
Enter factorial style: iterative
Generated iterative factorial function
Enter a number to test: 44
44! = 2658271574788448768043625811014615890319638528000000000

All factorial functions are ready to use!
PS C:\Users\pc\OneDrive\Documents\AIAC\LAB>

```

### Prompt 3:

Write a python function to compute the factorial of a non-negative integer using recursion

```

task1.py task1.2.py task1.3.py X
task1.3.py > parse_int
1 import sys
2 from typing import Optional
3
4
5 def compute_factorial(n: int) -> int:
6
7     if n < 0:
8         raise ValueError("Factorial is not defined for negative numbers")
9
10    if n == 0 or n == 1:
11        return 1
12
13    result = 1
14    for i in range(2, n + 1):
15        result *= i
16    return result
17
18
19 def parse_int(value: str) -> int:
20     try:
21         return int(value)
22     except ValueError as error:
23         raise ValueError(f"Invalid integer: {value}") from error
24
25
26 def main(argv: Optional[list[str]] = None) -> int:
27     argv = list(sys.argv[1:]) if argv is None else argv
28
29     try:
30         if argv:
31             n = parse_int(argv[0])
32         else:
33             raw = input("Enter a non-negative integer: ")
34             n = parse_int(raw)
35

```

```

29     try:
30         if argv:
31             n = parse_int(argv[0])
32         else:
33             raw = input("Enter a non-negative integer: ")
34             n = parse_int(raw)
35
36         result = compute_factorial(n)
37         print(f"{n}! = {result}")
38         return 0
39     except Exception as exc:
40         print(f"Error: {exc}")
41         return 1
42
43
44 if __name__ == "__main__":
45     raise SystemExit(main())
46
47

```

## Output:

```

Problems  Output  Debug Console  Terminal  Ports
Enter a non-negative integer: 5
5! = 120
PS C:\Users\pc\OneDrive\Documents\AIAC\LAB>
PS C:\Users\pc\OneDrive\Documents\AIAC\LAB>

```

#

# This program contains two main parts:

# 1. A recursive function to calculate the factorial of a non-negative integer.

# 2. A higher-order function that generates and returns a new factorial function.

#

import sys

# Set a higher recursion limit for demonstration, as Python's default is often 1000.

# This is not necessary for small numbers but good practice for deep recursion.

# sys.setrecursionlimit(2000)

# --- Demonstration of the functions ---

# Part 1: Demonstrate the recursive factorial function

number = 5

```

try:
    result_recursive = factorial_recursive(number)
    print(f"The factorial of {number} using the direct recursive function is:
{result_recursive}")
except ValueError as e:
    print(f"Error: {e}")

```

# Part 2: Demonstrate the function generator

# Generate a new factorial function using the 'generate\_factorial\_function' program

my\_new\_factorial\_function = generate\_factorial\_function(method="recursive")

# Now, use the newly generated function to compute a factorial

another\_number = 7

```

try:
    result_generated = my_new_factorial_function(another_number)
    print(f"The factorial of {another_number} using the generated function is:
{result_generated}")
except ValueError as e:
    print(f"Error: {e}")

```

# Example of generating an iterative function instead

my\_iterative\_function = generate\_factorial\_function(method="iterative")

print(f"The factorial of {another\_number} using the generated iterative function is:
{my\_iterative\_function(another\_number)}")

### Expected Output#1

- Comparison of AI-generated code styles

### Task Description#2

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

#### Prompt:

Write a python function that generates a sorting function.

```
task2.py > _merge_sorted_lists
1 from typing import List
2 def sort_numbers(numbers: List[int]) -> List[int]:
3
4     if len(numbers) <= 1:
5         return numbers.copy()
6
7     middle_index = len(numbers) // 2
8     left_sorted = sort_numbers(numbers[:middle_index])
9     right_sorted = sort_numbers(numbers[middle_index:])
10    return _merge_sorted_lists(left_sorted, right_sorted)
11 def _merge_sorted_lists(left_values: List[int], right_values: List[int]) -> List[int]:
12     result: List[int] = []
13     left_index = 0
14     right_index = 0
15
16     while left_index < len(left_values) and right_index < len(right_values):
17         if left_values[left_index] <= right_values[right_index]:
18             result.append(left_values[left_index])
19             left_index += 1
20         else:
21             result.append(right_values[right_index])
22             right_index += 1
23     if left_index < len(left_values):
24         result.extend(left_values[left_index:])
25     if right_index < len(right_values):
26         result.extend(right_values[right_index:])
27
28     return result
29 if __name__ == "__main__":
30     example = [5, 2, 9, 1, 5, 6]
31     print("Input:", example)
32     print("Sorted:", sort_numbers(example))
```

#### Output:

```
task2.py > ...
1
Problems Output Debug Console Terminal Ports
Input: [5, 2, 9, 1, 5, 6]
Sorted: [1, 2, 5, 5, 6, 9]
PS C:\Users\pc\OneDrive\Documents\AIAC\LAB>
```

### Expected Output#2

- Functional sorting code from AI

### Task Description#3

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

#### Prompt:

write a python code that calculate the power consumption bill.

```

Task1.py Task1.2.py Task1.3.py Task2.py task3.py
task3.py > ...
1 # Step 1: Simple power bill calculation (fixed rate)
2 def calculate_power_bill(units):
3     """
4     Calculate the power bill for a given number of units at a fixed rate.
5     Args:
6     units (float): Number of electricity units consumed.
7     Returns:
8     float: Total bill amount.
9     """
10    rate_per_unit = 5.0 # Example fixed rate per unit
11    return units * rate_per_unit
12
13 # Step 2: Add input validation
14 def calculate_power_bill_v2(units):
15     """
16     Calculate the power bill with input validation.
17     Args:
18     units (float): Number of electricity units consumed.
19     Returns:
20     float: Total bill amount.
21     Raises:
22     ValueError: If units is negative.
23     """
24    if units < 0:
25        raise ValueError("Units consumed cannot be negative.")
26    rate_per_unit = 5.0
27    return units * rate_per_unit
28
29 # Step 3: Add slab-wise billing (tiered rates)

```

```

Task1.py Task1.2.py Task1.3.py Task2.py task3.py
task3.py > ...
14 def calculate_power_bill_v2(units):
15     return units * rate_per_unit
16
17 # Step 3: Add slab-wise billing (tiered rates)
18 def calculate_power_bill_v3(units):
19     """
20     Calculate the power bill using slab rates.
21     Args:
22     units (float): Number of electricity units consumed.
23     Returns:
24     float: Total bill amount.
25     """
26    if units < 0:
27        raise ValueError("Units consumed cannot be negative.")
28    bill = 0
29    if units <= 100:
30        bill = units * 3.0
31    elif units <= 200:
32        bill = 100 * 3.0 + (units - 100) * 5.0
33    else:
34        bill = 100 * 3.0 + 100 * 5.0 + (units - 200) * 8.0
35    return bill
36
37 # Step 4: Add taxes and fixed charges
38 def calculate_power_bill_v4(units):
39     """
40     Calculate the power bill with slab rates, fixed charges, and tax.
41     Args:
42     units (float): Number of electricity units consumed.

```

```

Task1.py Task1.2.py Task1.3.py Task2.py task3.py
task3.py > ...
50 def calculate_power_bill_v4(units):
51     """
52     float: Total bill amount including fixed charges and tax.
53     """
54    if units < 0:
55        raise ValueError("Units consumed cannot be negative.")
56    fixed_charge = 50.0
57    tax_rate = 0.10 # 10% tax
58    # Slab calculation
59    if units <= 100:
60        bill = units * 3.0
61    elif units <= 200:
62        bill = 100 * 3.0 + (units - 100) * 5.0
63    else:
64        bill = 100 * 3.0 + 100 * 5.0 + (units - 200) * 8.0
65    bill += fixed_charge
66    bill += bill * tax_rate
67    return bill
68
69 # Step 5: Interactive usage
70 if __name__ == "__main__":
71    try:
72        units = float(input("Enter the number of electricity units consumed: "))
73        print("Simple Bill (fixed rate):", calculate_power_bill(units))
74        print("Validated Bill (fixed rate):", calculate_power_bill_v2(units))
75        print("Slab-wise Bill:", calculate_power_bill_v3(units))
76        print("Final Bill (with fixed charges and tax):", calculate_power_bill_v4(units))
77    except Exception as e:
78        print("Error:", e)
79
80

```

Output:

```

Problems Output Debug Console Terminal Ports
PS C:\Users\pc\OneDrive\Documents\AIAC\LAB> & C:/Users/pc/AppData/Local/Programs/Python/Python313/python.exe c:/Users/pc/OneDrive/Documents/AIAC/LAB/task3.py
Enter the number of electricity units consumed: 55
Simple Bill (fixed rate): 275.0
Validated Bill (fixed rate): 275.0
Slab-wise Bill: 165.0
Final Bill (with fixed charges and tax): 236.5
PS C:\Users\pc\OneDrive\Documents\AIAC\LAB>

```

This Python code provides a comprehensive electricity bill calculator, starting with a simple model and progressively adding features to make it more realistic.

- `calculate_power_bill`: Calculates a basic bill using a fixed rate per unit.
- `calculate_power_bill_v2`: Adds crucial input validation to prevent negative unit values.
- `calculate_power_bill_v3`: Implements a slab-wise or tiered billing system, where different rates apply to different consumption brackets.
- `calculate_power_bill_v4`: The most complete version, incorporating fixed charges and a percentage-based tax on top of the slab rates.
- The `if __name__ == "__main__":` block allows for interactive use, prompting the user for input and displaying the results from each function, demonstrating the progression from a simple to a complex calculation.

### Expected Output#3

- Enhanced AI output with clearer prompts

### Task Description#4

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

Prompt:

Using python generate two linked functions :

```

task1.py task1.2.py task1.3.py task2.py task3.py task4.py
task4.py > register_user
1 def register_user(users_db, username, password):
2     Ctrl+L to chat Ctrl+K to generate
3     if username in users_db:
4         return False # Username already exists
5     users_db[username] = password
6     return True
7
8 def login_user(users_db, username, password):
9
10    if username in users_db and users_db[username] == password:
11        return True
12    return False
13
14 # Example usage
15 if __name__ == "__main__":
16     users = {}
17     # Register users
18     print("Registering user 'alice':", register_user(users, "alice", "password123"))
19     print("Registering user 'bob':", register_user(users, "bob", "securepass"))
20     print("Registering user 'alice' again:", register_user(users, "alice", "newpass")) # Should fail
21
22     # Login attempts
23     print("Login 'alice' with correct password:", login_user(users, "alice", "password123"))
24     print("Login 'alice' with wrong password:", login_user(users, "alice", "wrongpass"))
25     print("Login 'charlie' (not registered):", login_user(users, "charlie", "nopass"))

```

### Output

```

PS C:\Users\pc\OneDrive\Documents\AIAC\LAB> & C:/Users/pc/AppData/Local/Programs/Python/Python313/python.exe c:/Users/pc/OneDrive/Documents/AIAC/LAB/task4.py
Registering user 'alice': True
Registering user 'alice': True
Registering user 'bob': True
Registering user 'alice' again: False
Login 'alice' with correct password: True
Login 'alice' with wrong password: False
Login 'charlie' (not registered): False
PS C:\Users\pc\OneDrive\Documents\AIAC\LAB>

```



#### Expected Output#4

- Consistent functions with shared logic

#### Task Description#5

- Analyzing Prompt Specificity: Improving Temperature Conversion Function with Clear Instructions

```
task5.py > convert_temperature
1 def convert_temperature(value, from_unit, to_unit):
2     from_unit = from_unit.upper()
3     to_unit = to_unit.upper()
4     if from_unit not in ['C', 'F', 'K'] or to_unit not in ['C', 'F', 'K']:
5         raise ValueError("Units must be 'C', 'F', or 'K'.")
6
7     # Convert input to Celsius first
8     if from_unit == 'C':
9         celsius = value
10    elif from_unit == 'F':
11        celsius = (value - 32) * 5/9
12    elif from_unit == 'K':
13        celsius = value - 273.15
14
15    # Convert from Celsius to target unit
16    if to_unit == 'C':
17        return celsius
18    elif to_unit == 'F':
19        return celsius * 9/5 + 32
20    elif to_unit == 'K':
21        return celsius + 273.15
22
23
24 if __name__ == "__main__":
25     print("Temperature Converter")
26     print("Supported units: C (Celsius), F (Fahrenheit), K (Kelvin)")
27     try:
28         value = float(input("Enter the temperature value: "))
29         from_unit = input("Enter the unit to convert from (C/F/K): ").strip()
30         to_unit = input("Enter the unit to convert to (C/F/K): ").strip()
```

```
task5.py > convert_temperature
26 print("Supported units: C (Celsius), F (Fahrenheit), K (Kelvin)")
27 try:
28     value = float(input("Enter the temperature value: "))
29     from_unit = input("Enter the unit to convert from (C/F/K): ").strip()
30     to_unit = input("Enter the unit to convert to (C/F/K): ").strip()
31     result = convert_temperature(value, from_unit, to_unit)
32     print(f"{value} {from_unit.upper()} is {result:.2f} {to_unit.upper()}")
33 except Exception as e:
34     print("Error:", e)
```

#### Output:

```
PS C:\Users\pc\OneDrive\Documents\AIAC\LAB> & C:/Users/pc/AppData/Local/Programs/Python/Python313/python.exe c:/Users/pc/OneDrive/Documents/AIAC/LAB/task5.py
Temperature Converter
Supported units: C (Celsius), F (Fahrenheit), K (Kelvin)
Enter the temperature value: 40
Enter the unit to convert from (C/F/K): c
Enter the unit to convert to (C/F/K): k
40.0 C is 313.15 K
PS C:\Users\pc\OneDrive\Documents\AIAC\LAB>
```

#### Expected Output#5

- Code quality difference analysis for various prompts

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

#### Evaluation Criteria:

Criteria	Max Marks
Factorial Function (Task#1)	0.5

	Sorting Function (Task#2)	0.5		
	Vogue Vs. Specific Prompting (Task #3)	0.5		
	Linked Functions (Task #4)	0.5		
	Temperature Conversion Function (Task #5)	0.5		
	<b>Total</b>	<b>2.5 Marks</b>		