

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
ProgramName:B. Tech		Assignment Type: Lab	AcademicYear:2025-2026
CourseCoordinatorName		Venkataramana Veeramsetty	
Instructor(s)Name		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	
CourseCode	24CS002PC215 <th>CourseTitle</th> <td>AI Assisted Coding</td>	CourseTitle	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment		Time(s)	
Duration	2 Hours	Applicableto Batches	
AssignmentNumber: 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>Lab Objectives:</p> <ul style="list-style-type: none"> To understand how prompt structure and wording influence AI-generated code. To explore how context (like comments and function names) helps AI generate relevant output. To evaluate the quality and accuracy of code based on prompt clarity. To develop effective prompting strategies for AI-assisted programming. <p>Lab Outcomes (LOs):</p> <p>After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> Generate Python code using Google Gemini in Google Colab. Analyze the effectiveness of code explanations and suggestions by Gemini. Set up and use Cursor AI for AI-powered coding assistance. Evaluate and refactor code using Cursor AI features. Compare AI tool behavior and code quality across different platforms. 	03.08.2025 EOD

	<p>Task Description#1</p> <ul style="list-style-type: none"> Try 3 different prompts to generate a factorial function. <p>Expected Output#1</p> <ul style="list-style-type: none"> Comparison of AI-generated code styles <p>Task Description#2</p> <ul style="list-style-type: none"> Provide a clear example input-output prompt to generate a sorting function. <p>Expected Output#2</p> <ul style="list-style-type: none"> Functional sorting code from AI <p>Task Description#3</p> <ul style="list-style-type: none"> Start with the vague prompt “Generate python code to calculate power bill” and improve it step-by-step <p>Expected Output#3</p> <ul style="list-style-type: none"> Enhanced AI output with clearer prompts <p>Task Description#4</p> <ul style="list-style-type: none"> Write structured comments to help AI generate two linked functions (e.g., login_user() and register_user()). <p>Expected Output#4</p> <ul style="list-style-type: none"> Consistent functions with shared logic <p>Task Description#5</p> <ul style="list-style-type: none"> Analyzing Prompt Specificity: Improving Temperature Conversion Function with Clear Instructions <p>Expected Output#5</p> <ul style="list-style-type: none"> Code quality difference analysis for various prompts <p>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</p> <p>Evaluation Criteria:</p> <table border="1"> <thead> <tr> <th>Criteria</th><th>Max Marks</th></tr> </thead> <tbody> <tr> <td>Factorial Function (Task#1)</td><td>0.5</td></tr> <tr> <td>Sorting Function (Task#2)</td><td>0.5</td></tr> <tr> <td>Vague Vs. Specific Prompting (Task #3)</td><td>0.5</td></tr> <tr> <td>Linked Functions (Task #4)</td><td>0.5</td></tr> <tr> <td>Temperature Conversion Function (Task #5)</td><td>0.5</td></tr> <tr> <td>Total</td><td>2.5 Marks</td></tr> </tbody> </table>	Criteria	Max Marks	Factorial Function (Task#1)	0.5	Sorting Function (Task#2)	0.5	Vague Vs. Specific Prompting (Task #3)	0.5	Linked Functions (Task #4)	0.5	Temperature Conversion Function (Task #5)	0.5	Total	2.5 Marks	
Criteria	Max Marks															
Factorial Function (Task#1)	0.5															
Sorting Function (Task#2)	0.5															
Vague Vs. Specific Prompting (Task #3)	0.5															
Linked Functions (Task #4)	0.5															
Temperature Conversion Function (Task #5)	0.5															
Total	2.5 Marks															

Task Description#1

- Try 3 different prompts to generate a factorial function.

PROMPT: generate different types of prompts to calculate factorial of a function.

```
● factorial.py > factorial_iterative
 1 def factorial_iterative(n):
 2     """
 3         Calculate factorial using iterative approach
 4     """
 5     if n < 0:
 6         return "Factorial is not defined for negative numbers"
 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    return result
14
15 def factorial_recursive(n):
16     """
17         Calculate factorial using recursive approach
18     """
19     if n < 0:
20         return "Factorial is not defined for negative numbers"
21     if n == 0 or n == 1:
22         return 1
23     return n * factorial_recursive(n - 1)
24
25 def factorial_math_module(n):
26
27     def factorial_math_module(n):
28         Calculate factorial using math module
29         """
30
31             import math
32             if n < 0:
33                 return "Factorial is not defined for negative numbers"
34             return math.factorial(n)
35
36     def main():
37         print("=" * 50)
38         print("FACTORIAL CALCULATOR")
39         print("=" * 50)
40
41         while True:
42             try:
43                 # Get user input
44                 user_input = input("\nEnter a number to calculate factorial (or 'quit' to exit): ")
45
46                 if user_input.lower() in ['quit', 'exit', 'q']:
47                     print("Thank you for using the Factorial Calculator!")
48                     break
49
50                 # Convert input to integer
51                 n = int(user_input)
52
53             except ValueError:
54                 print("Please enter a valid integer value."))
```

```
def main():

    # Calculate factorial using different methods
    print(f"\nCalculating factorial of {n}:")
    print("-" * 30)

    # Iterative method
    result_iterative = factorial_iterative(n)
    print(f"Iterative method: {n}! = {result_iterative}")

    # Recursive method
    result_recursive = factorial_recursive(n)
    print(f"Recursive method: {n}! = {result_recursive}")

    # Math module method
    result_math = factorial_math_module(n)
    print(f"Math module method: {n}! = {result_math}")

    # Show the calculation process for smaller numbers
    if n <= 10 and n >= 0:
        print(f"\nCalculation process:")
        process = " ".join(str(i) for i in range(1, n + 1))
        print(f"{n}! = {process} = {result_iterative}")


except ValueError:

    except ValueError:
        print("Error: Please enter a valid integer!")
    except RecursionError:
        print("Error: Number too large for recursive calculation!")
    except Exception as e:
        print(f"Error: {e}")

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

```

Enter a number to calculate factorial (or 'quit' to exit): 5

Calculating factorial of 5:
-----
Iterative method: 5! = 120
Recursive method: 5! = 120
Math module method: 5! = 120

Calculation process:
5! = 1 × 2 × 3 × 4 × 5 = 120

```

Task Description#2

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

PROMPT: generate a sorting function using python code so that we can provide a list of numbers to sort integers in ascending and descending order

```

def sort_numbers(numbers, order='ascending'):
    """
    Sort a list of numbers in ascending or descending order.

    Args:
        numbers (list): A list of integers to sort
        order (str): 'ascending' or 'descending' (default: 'ascending')

    Returns:
        list: A new sorted list (original list is not modified)

    Raises:
        ValueError: If order is not 'ascending' or 'descending'
        TypeError: If numbers contains non-integer values

    Examples:
        >>> sort_numbers([64, 34, 25, 12, 22, 11, 90])
        [11, 12, 22, 25, 34, 64, 90]

        >>> sort_numbers([64, 34, 25, 12, 22, 11, 90], 'descending')
        [90, 64, 34, 25, 22, 12, 11]
    """
    # Validate input
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")

```

```
def sort_numbers(numbers, order='ascending'):

    if not numbers:
        return []

    # Check if all elements are integers
    for num in numbers:
        if not isinstance(num, int):
            raise TypeError(f"All elements must be integers. Found: {type(num).__name__}")

    # Validate order parameter
    if order not in ['ascending', 'descending']:
        raise ValueError("Order must be 'ascending' or 'descending')

    # Create a copy to avoid modifying the original list
    sorted_numbers = numbers.copy()

    # Sort the list
    if order == 'ascending':
        sorted_numbers.sort()
    else: # descending
        sorted_numbers.sort(reverse=True)

    return sorted_numbers
```

Review next file >

```
def sort_numbers_manual(numbers, order='ascending'):
    """
    Manual implementation of sorting using bubble sort algorithm.
    This function demonstrates how sorting works internally.

    Args:
        numbers (list): A list of integers to sort
        order (str): 'ascending' or 'descending' (default: 'ascending')

    Returns:
        list: A new sorted list (original list is not modified)
    """

    # Validate input
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")

    if not numbers:
        return []

    # Check if all elements are integers
    for num in numbers:
        if not isinstance(num, int):
            raise TypeError(f"A [ Review next file > be integers. Found: {type(num).__name__}
```

```
def sort_numbers_manual(numbers, order='ascending'):
    # Validate order parameter
    if order not in ['ascending', 'descending']:
        raise ValueError("Order must be 'ascending' or 'descending'")

    # Create a copy to avoid modifying the original list
    arr = numbers.copy()
    n = len(arr)

    # Bubble sort implementation
    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            # For ascending order: swap if current > next
            # For descending order: swap if current < next
            should_swap = (arr[j] > arr[j + 1]) if order == 'ascending' else (arr[j]

                if should_swap:
                    arr[j], arr[j + 1] = arr[j + 1], arr[j]
                    swapped = True

    # If no swapping occurred, array is sorted
    if not swapped:
        break
```

Review next file >

```
return arr
```

```
# Example usage and demonstration
if __name__ == "__main__":
    # Test data
    test_numbers = [64, 34, 25, 12, 22, 11, 90]

    print("=" * 50)
    print("SORTING FUNCTION DEMONSTRATION")
    print("=" * 50)

    print(f"Original list: {test_numbers}")
    print()

    # Test ascending sort (using built-in sort)
    ascending_result = sort_numbers(test_numbers, 'ascending')
    print(f"Ascending order (built-in): {ascending_result}")

    # Test descending sort (using built-in sort)
    descending_result = sort_numbers(test_numbers, 'descending')
    print(f"Descending order (built-in): {descending_result}")

    print()
```

Review next file >

```

# Test manual implementation
ascending_manual = sort_numbers_manual(test_numbers, 'ascending')
print(f"Ascending order (manual): {ascending_manual}")

descending_manual = sort_numbers_manual(test_numbers, 'descending')
print(f"Descending order (manual): {descending_manual}")

print()
print("=" * 50)
print("INTERACTIVE DEMO")
print("=" * 50)

# Interactive demo
while True:
    try:
        print("\nEnter numbers separated by spaces (e.g., 64 34 25 12 22 11 90):")
        print("Or type 'quit' to exit")

        user_input = input("Numbers: ").strip()

        if user_input.lower() == 'quit':
            print("Goodbye!")
            break

```

Review next file >

```

# Parse input
numbers = [int(x) for x in user_input.split()]

if not numbers:
    print("Please enter at least one number!")
    continue

print(f"\nOriginal list: {numbers}")

# Sort in both orders
ascending = sort_numbers(numbers, 'ascending')
descending = sort_numbers(numbers, 'descending')

print(f"Ascending order: {ascending}")
print(f"Descending order: {descending}")

except ValueError:
    print("Error: Please enter valid integers separated by spaces!")
except KeyboardInterrupt:
    print("\n\nGoodbye!")
    break
except Exception as e:
    print(f"Error: {e}") Review next file >

```

Review next file >

```
Enter numbers separated by spaces (e.g., 64 34 25 12 22 11 90):
Or type 'quit' to exit
Numbers: 2 6 8 9 4
```

```
Original list: [2, 6, 8, 9, 4]
Ascending order: [2, 4, 6, 8, 9]
Descending order: [9, 8, 6, 4, 2]
```

```
Enter numbers separated by spaces (e.g., 64 34 25 12 22 11 90):
Or type 'quit' to exit
Numbers: []
```

Task Description#3

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

PROMPT: Generate python code to calculate power bill step by step

```
1 """
2 Power Bill Calculator
3 A comprehensive tool to calculate electricity bills step by step
4 """
5
6 class PowerBillCalculator:
7     def __init__(self):
8         # Default rate structure (can be customized)
9         self.rate_structure = {
10             'residential': {
11                 'base_charge': 10.00, # Monthly base charge
12                 'tiers': [
13                     {'limit': 500, 'rate': 0.12}, # First 500 kWh at $0.12/kWh
14                     {'limit': 1000, 'rate': 0.15}, # Next 500 kWh at $0.15/kWh
15                     {'limit': float('inf'), 'rate': 0.18} # Above 1000 kWh at $0.18/kWh
16                 ]
17             },
18             'commercial': {
19                 'base_charge': 25.00,
20                 'tiers': [
21                     {'limit': 1000, 'rate': 0.10},
22                     {'limit': 5000, 'rate': 0.13},
23                     {'limit': float('inf'), 'rate': 0.16}
24                 ]
25             }
}
```

```

6   class PowerBillCalculator:
7       def __init__(self):
8           """
9               # Tax rates
10              self.tax_rate = 0.08 # 8% tax rate
11
12      def calculate_bill(self, customer_type, usage_kwh, customer_name="Customer"):
13          """
14              Calculate power bill step by step
15
16          Args:
17              customer_type (str): 'residential' or 'commercial'
18              usage_kwh (float): Total kWh usage
19              customer_name (str): Customer name for display
20
21          Returns:
22              dict: Detailed bill breakdown
23          """
24
25          print(f"\n{'='*60}")
26          print(f"POWER BILL CALCULATION FOR {customer_name.upper()}")
27          print(f"{'='*60}")
28          print(f"Customer Type: {customer_type.title()}")
29          print(f"Usage: {usage_kwh} Review next file >")
30          print(f"{'='*60}")

```

```

50
51      # Step 1: Get rate structure
52      rates = self.rate_structure[customer_type.lower()]
53      print(f"\nSTEP 1: RATE STRUCTURE")
54      print(f"Base Charge: ${rates['base_charge']:.2f}")
55      print("Usage Tiers:")
56      for i, tier in enumerate(rates['tiers'], 1):
57          if tier['limit'] == float('inf'):
58              print(f"  Tier {i}: Above {rates['tiers'][i-2]['limit']} kWh @ ${tier['rate']:.2f}/kWh")
59          else:
60              print(f"  Tier {i}: Up to {tier['limit']} kWh @ ${tier['rate']:.2f}/kWh")
61
62      # Step 2: Calculate usage charges by tier
63      print(f"\nSTEP 2: USAGE CALCULATION BY TIER")
64      total_usage_charge = 0
65      remaining_usage = usage_kwh
66
67      for i, tier in enumerate(rates['tiers']):
68          if remaining_usage <= 0:
69              break
70
71          # Calculate how much to this tier
72          if i == 0:

```

```

70     # Calculate how much usage falls into this tier
71     if i == 0:
72         # First tier
73         tier_usage = min(remaining_usage, tier['limit'])
74         tier_limit_display = tier['limit']
75     else:
76         # Subsequent tiers
77         previous_limit = rates['tiers'][i-1]['limit']
78         tier_usage = min(remaining_usage, tier['limit'] - previous_limit)
79         tier_limit_display = f"{previous_limit} - {tier['limit']}"
80
81     if tier_usage > 0:
82         tier_charge = tier_usage * tier['rate']
83         total_usage_charge += tier_charge
84         remaining_usage -= tier_usage
85
86         print(f"  Tier {i+1} ({tier_limit_display} kWh): {tier_usage:.2f} kWh")
87
88     # Step 3: Calculate subtotal
89     print(f"\nSTEP 3: SUBTOTAL CALCULATION")
90     subtotal = rates['base_charge'] + usage_charge
91     print(f"Base Charge: ${r[0] Review next file > base_charge:.2f}")
92     print(f"Usage Charge: ${total_usage_charge:.2f}")

93     def calculate_bill(self, customer_type, usage_kwh, customer_name):
94         print(f"Usage Charge: ${total_usage_charge:.2f}")
95         print(f"Subtotal: ${subtotal:.2f}")
96
97         # Step 4: Calculate taxes
98         print(f"\nSTEP 4: TAX CALCULATION")
99         tax_amount = subtotal * self.tax_rate
100        print(f"Tax Rate: {self.tax_rate * 100:.1f}%")
101        print(f"Tax Amount: ${tax_amount:.2f}")
102
103        # Step 5: Calculate total
104        print(f"Subtotal: ${subtotal:.2f}")
105        print(f"Tax: ${tax_amount:.2f}")
106        print(f"{'='*40}")
107        print(f"TOTAL BILL: ${total_bill:.2f}")
108        print(f"{'='*40}")
109
110        # Return detailed breakdown
111        return {
112            'customer_name': cus[0] Review next file > customer_name,
113            'customer_type': cus[1] Review next file > customer_type,
114            'usage_kwh': usage_kwh,

```

```

115     'base_charge': rates['base_charge'],
116     'usage_charge': total_usage_charge,
117     'subtotal': subtotal,
118     'tax_rate': self.tax_rate,
119     'tax_amount': tax_amount,
120     'total_bill': total_bill
121   }
122
123   def compare_plans(self, usage_kwh, customer_name="Customer"):
124     """
125       Compare residential vs commercial plans
126     """
127     print(f"\n{'='*70}")
128     print(f"PLAN COMPARISON FOR {customer_name.upper()}")
129     print(f"{'='*70}")
130     print(f"Usage: {usage_kwh:.2f} kWh")
131     print(f"{'='*70}")
132
133     residential_bill = self.calculate_bill('residential', usage_kwh, f"{customer_na
134     commercial_bill = self.calculate_bill('commercial', usage_kwh, f"{customer_name
135
136     print(f"\n{'='*70}")           Review next file >
137     print(f"COMPARISON SUMMARY")
```

```

123   def compare_plans(self, usage_kwh, customer_name="Customer"):
124     print(f"{'='*70}")
125     print(f"{'Plan':<15} {'Base':<8} {'Usage':<10} {'Tax':<8} {'Total':<10}")
126     print(f"{'-'*70}")
127     print(f"{'Residential':<15} ${residential_bill['base_charge']:<7.2f} ${resident
128     print(f"{'Commercial':<15} ${commercial_bill['base_charge']:<7.2f} ${commercial
129
130     savings = residential_bill['total_bill'] - commercial_bill['total_bill']
131     if savings > 0:
132       print(f"\nCommercial plan saves: ${savings:.2f}")
133     elif savings < 0:
134       print(f"\nResidential plan saves: ${abs(savings):.2f}")
135     else:
136       print(f"\nBoth plans cost the same")
137
138     return residential_bill, commercial_bill
139
140   def main():
141     """Main function to demonstrate the power bill calculator"""
142     calculator = PowerBillCalculator()
143
144     print("POWER BILL CALCULATOR")
145     print("=" * 50)           Review next file >
```

```
154 def main():
161     while True:
162         print("\nOptions:")
163         print("1. Calculate Residential Bill")
164         print("2. Calculate Commercial Bill")
165         print("3. Compare Plans")
166         print("4. Custom Rate Structure")
167         print("5. Exit")
168
169         choice = input("\nEnter your choice (1-5): ").strip()
170
171         if choice == '1':
172             try:
173                 usage = float(input("Enter usage in kWh: "))
174                 name = input("Enter customer name (optional): ").strip() or "Residential"
175                 calculator.calculate_bill('residential', usage, name)
176             except ValueError:
177                 print("Invalid input. Please enter a valid number.")
178
179         elif choice == '2':
180             try:
181                 usage = float(input("Enter usage in kWh: "))
182                 name = input("Enter customer name (optional): ").strip() or "Commercial"
183                 calculator.calculate_bill('commercial', usage, name)
184             except ValueError:
```

```
154 def main():
184     except ValueError:
185         print("Invalid input. Please enter a valid number.")
186
187     elif choice == '3':
188         try:
189             usage = float(input("Enter usage in kWh: "))
190             name = input("Enter customer name (optional): ").strip() or "Customer"
191             calculator.compare_plans(usage, name)
192         except ValueError:
193             print("Invalid input. Please enter a valid number.")
194
195     elif choice == '4':
196         print("\nCustom Rate Structure Example:")
197         print("This would allow you to modify the rate structure.")
198         print("For now, using default rates.")
199
200     elif choice == '5':
201         print("Thank you for using the Power Bill Calculator!")
202         break
203
204     else:
205         print("Invalid choice. Please enter a number between 1-5.")
```

Options:

1. Calculate Residential Bill
2. Calculate Commercial Bill
3. Compare Plans
4. Custom Rate Structure
5. Exit

Enter your choice (1-5): 1

Enter usage in kWh: 30

Enter customer name (optional): kt

=====

POWER BILL CALCULATION FOR KT

=====

Customer Type: Residential

Usage: 30.00 kWh

STEP 1: RATE STRUCTURE

Base Charge: \$10.00

Usage Tiers:

Tier 1: Up to 500 kWh @ \$0.12/kWh

Tier 2: Up to 1000 kWh @ \$0.15/kWh

Tier 3: Above 1000 kWh @ \$0.18/kWh

STEP 2: USAGE CALCULATION BY TIER

Tier 1 (500 kWh): $30.00 \text{ kWh} \times \$0.12 = \$3.60$

STEP 3: SUBTOTAL CALCULATION

Base Charge: \$10.00

Usage Charge: \$3.60

Subtotal: \$13.60

STEP 4: TAX CALCULATION

Tax Rate: 8.0%

Tax Amount: \$1.09

STEP 5: FINAL TOTAL

Subtotal: \$13.60

Tax: \$1.09

=====

TOTAL BILL: \$14.69

Task Description#4

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

PROMPT: generate a python code to help AI generate two linked functions (e.g., login_user() and register_user()).

```
1 """
2 Simple AI Function Generator
3 Generate linked functions for AI assistance
4 """
5
6 def generate_linked_functions(function_type="login_register"):
7     """Generate two linked functions based on type"""
8
9     if function_type == "login_register":
10         return {
11             "login_user": ''
12         }
13     def login_user(username, password):
14         """Login user with username and password"""
15         # Validate input
16         if not username or not password:
17             return {"success": False, "error": "Missing credentials"}
18
19         # Check credentials (placeholder)
20         if check_credentials(username, password):
21             session = create_session(username)
22             return {"success": True, "session": session}
23         else:
24             return {"success": False, "error": 'Review next file > credentials'}
```

```

6  def generate_linked_functions(function_type="login_register"):
25     |     "register_user": ''
26 def register_user(username, email, password):
27     """Register a new user"""
28     # Validate input
29     if not username or not email or not password:
30         return {"success": False, "error": "Missing required fields"}
31
32     # Check if user exists
33     if user_exists(username):
34         return {"success": False, "error": "Username already exists"}
35
36     # Create user (placeholder)
37     user_id = create_user(username, email, password)
38     return {"success": True, "user_id": user_id}
39 ...
40     }
41
42 elif function_type == "file_ops":
43     return {
44         "create_file": ''
45 def create_file(filename, content):
46     """Create a new file with content"""
47     try:
48         with open(filename, 'w') as f:
49             f.write(content)
50         return {"success": True, "filename": filename}
51     except Exception as e:
52         return {"success": False, "error": str(e)}
53 ...
54     "read_file": ''
55 def read_file(filename):
56     """Read content from file"""
57     try:
58         with open(filename, 'r') as f:
59             content = f.read()
60         return {"success": True, "content": content}
61     except Exception as e:
62         return {"success": False, "error": str(e)}
63 ...
64     }
65
66     return {"error": "Unknown function type"}
67
68 def generate_helper_functions():
69     """Generate helper functions for the Review next file > is"""
70     return ''
71
72 # Helper Functions

```

```

6  def generate_linked_functions(function_type= login_register ):
48     with open(filename, 'w') as f:
49         f.write(content)
50     return {"success": True, "filename": filename}
51 except Exception as e:
52     return {"success": False, "error": str(e)}
53 ...
54     "read_file": ''
55 def read_file(filename):
56     """Read content from file"""
57     try:
58         with open(filename, 'r') as f:
59             content = f.read()
60         return {"success": True, "content": content}
61     except Exception as e:
62         return {"success": False, "error": str(e)}
63 ...
64     }
65
66     return {"error": "Unknown function type"}
67
68 def generate_helper_functions():
69     """Generate helper functions for the Review next file > is"""
70     return ''
71
72 # Helper Functions

```

```
68     def generate_helper_functions():
69         return ''
70
71     # Helper functions
72     def check_credentials(username, password):
73         """Check if username and password are valid"""
74         # Placeholder - replace with actual authentication logic
75         return username == "admin" and password == "password"
76
77     def create_session(username):
78         """Create a session for logged in user"""
79         # Placeholder - replace with actual session creation
80         return {"user": username, "token": "session_token_123"}
81
82     def user_exists(username):
83         """Check if username already exists"""
84         # Placeholder - replace with actual database check
85         return False
86
87     def create_user(username, email, password):
88         """Create a new user account"""
89         # Placeholder - replace with actual user creation
90         return "user_id_123"
91     ...
```

[Review next file >](#)

```
93     # Example usage
94     if __name__ == "__main__":
95         # Generate Login/register functions
96         functions = generate_linked_functions("login_register")
97
98         print("Generated Login/Register Functions:")
99         print("=" * 40)
100        for name, code in functions.items():
101            print(f"\n{name}:")

102            print(code)

103
104        print("\nHelper Functions:")
105        print("=" * 40)
106        print(generate_helper_functions())
107
108        # Generate file operation functions
109        file_functions = generate_linked_functions("file_ops")
110
111        print("\nGenerated File Operation Functions:")
112        print("=" * 40)
113        for name, code in file_functions.items():
114            print(f"\n{name}:")

115            print(code)
```

[Review next file >](#)

```
=====

login_user:

def login_user(username, password):
    """Login user with username and password"""
    # Validate input
    if not username or not password:
        return {"success": False, "error": "Missing credentials"}

    # Check credentials (placeholder)
    if check_credentials(username, password):
        session = create_session(username)
        return {"success": True, "session": session}
    else:
        return {"success": False, "error": "Invalid credentials"}


register_user:

def register_user(username, email, password):
    """Register a new user"""
    # Validate input
    if not username or not email or not password:
        return {"success": False, "error": "Missing required fields"}

    # Check if user exists
    if user_exists(username):
        return {"success": False, "error": "Username already exists"}

    # Create user (placeholder)
    user_id = create_user(username, email, password)
    return {"success": True, "user_id": user_id}
```

Helper Functions:

```
=====
# Helper functions
def check_credentials(username, password):
    """Check if username and password are valid"""
    # Placeholder - replace with actual authentication logic
    return username == "admin" and password == "password"

def create_session(username):
    """Create a session for logged in user"""
    # Placeholder - replace with actual session creation
    return {"user": username, "token": "session_token_123"}

def user_exists(username):
    """Check if username already exists"""
    # Placeholder - replace with actual database check
    return False

def create_user(username, email, password):
    """Create a new user account"""
    # Placeholder - replace with actual user creation
    return "user_id_123"
```

Generated File Operation Functions:

```
=====
```

```
create_file:

def create_file(filename, content):
    """Create a new file with content"""
    try:
        with open(filename, 'w') as f:
            f.write(content)
        return {"success": True, "filename": filename}
    except Exception as e:
        return {"success": False, "error": str(e)}
```

```

read_file:

def read_file(filename):
    """Read content from file"""
    try:
        with open(filename, 'r') as f:
            content = f.read()
        return {"success": True, "content": content}
    except Exception as e:
        return {"success": False, "error": str(e)}

```

Task Description#5

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

PROMPT: generate a python code to calculate Temperature Conversion

```

"""
Temperature Converter
A comprehensive tool to convert temperatures between Celsius, Fahrenheit, and Kelvin
"""

class TemperatureConverter:
    def __init__(self):
        self.units = {
            'C': 'Celsius',
            'F': 'Fahrenheit',
            'K': 'Kelvin'
        }

    def celsius_to_fahrenheit(self, celsius):
        """
        Convert Celsius to Fahrenheit step by step

        Formula: F = (C × 9/5) + 32
        Args:
            celsius (float): Temperature in Celsius
        Returns:
            dict: Conversion result with steps
        """
        steps = []

```

```
# Step 1: Multiply by 9/5
step1_result = celsius * (9/5)
steps.append(f"Step 1: Multiply {celsius}°C by 9/5 = {step1_result}")

# Step 2: Add 32
fahrenheit = step1_result + 32
steps.append(f"Step 2: Add 32 to {step1_result} = {fahrenheit}°F")

return {
    'from_unit': 'Celsius',
    'to_unit': 'Fahrenheit',
    'input_value': celsius,
    'output_value': round(fahrenheit, 2),
    'steps': steps,
    'formula': 'F = (C × 9/5) + 32'
}

def fahrenheit_to_celsius(self, fahrenheit):
    """
    Convert Fahrenheit to Celsius step by step
    Formula: C = (F - 32) × 5/9
    Review next file >

```

```
Args:
    fahrenheit (float): Temperature in Fahrenheit

Returns:
    dict: Conversion result with steps
"""
steps = []

# Step 1: Subtract 32
step1_result = fahrenheit - 32
steps.append(f"Step 1: Subtract 32 from {fahrenheit}°F = {step1_result}")

# Step 2: Multiply by 5/9
celsius = step1_result * (5/9)
steps.append(f"Step 2: Multiply {step1_result} by 5/9 = {celsius}°C")

return {
    'from_unit': 'Fahrenheit',
    'to_unit': 'Celsius',
    'input_value': fahrenheit,
    'output_value': round(celsius, 2),
    'steps': steps,
    'formula': 'C = (F - 32) / 9 * 5 + 32' }
```

Review next file > [x]

```
def celsius_to_kelvin(self, celsius):
    """
    Convert Celsius to Kelvin step by step

    Formula: K = C + 273.15

    Args:
        celsius (float): Temperature in Celsius

    Returns:
        dict: Conversion result with steps
    """
    steps = []

    # Step 1: Add 273.15
    kelvin = celsius + 273.15
    steps.append(f"Step 1: Add 273.15 to {celsius}°C = {kelvin}K")

    return {
        'from_unit': 'Celsius',
        'to_unit': 'Kelvin',
        'input_value': celsius,
        'output_value': round(kelvi Review next file >
        'steps': steps,
        'formula': 'K = C + 273.15'
```

```
    }

def kelvin_to_celsius(self, kelvin):
    """
    Convert Kelvin to Celsius step by step

    Formula: C = K - 273.15

    Args:
        kelvin (float): Temperature in Kelvin

    Returns:
        dict: Conversion result with steps
    """
    steps = []

    # Step 1: Subtract 273.15
    celsius = kelvin - 273.15
    steps.append(f"Step 1: Subtract 273.15 from {kelvin}K = {celsius}°C")

    return {
        'from_unit': 'Kelvin',
        'to_unit': 'Celsius',
        'input_value': kelvin,
```

[Review next file >](#)

```

        'output_value': round(celsius, 2),
        'steps': steps,
        'formula': 'C = K - 273.15'
    }

def fahrenheit_to_kelvin(self, fahrenheit):
    """
    Convert Fahrenheit to Kelvin step by step

    Formula: K = (F - 32) × 5/9 + 273.15
    """

    Args:
        fahrenheit (float): Temperature in Fahrenheit

    Returns:
        dict: Conversion result with steps
    """
    steps = []

    # Step 1: Subtract 32
    step1_result = fahrenheit - 32
    steps.append(f"Step 1: Subtract {fahrenheit}°F = {step1_result}°C")
    Review next file >

    # Step 2: Multiply by 5/9
    step2_result = step1_result * (5/9)
    steps.append(f"Step 2: Multiply {step1_result} by 5/9 = {step2_result}°C")

    # Step 3: Add 273.15
    kelvin = step2_result + 273.15
    steps.append(f"Step 3: Add 273.15 to {step2_result}°C = {kelvin}K")

    return {
        'from_unit': 'Fahrenheit',
        'to_unit': 'Kelvin',
        'input_value': fahrenheit,
        'output_value': round(kelvin, 2),
        'steps': steps,
        'formula': 'K = (F - 32) × 5/9 + 273.15'
    }

def kelvin_to_fahrenheit(self, kelvin):
    """
    Convert Kelvin to Fahrenheit step by step

    Formula: F = (K - 273.15) × 9/5
    """
    Review next file >

```

```
Args:
    kelvin (float): Temperature in Kelvin

Returns:
    dict: Conversion result with steps
"""
steps = []

# Step 1: Subtract 273.15
step1_result = kelvin - 273.15
steps.append(f"Step 1: Subtract 273.15 from {kelvin}K = {step1_result}°C")

# Step 2: Multiply by 9/5
step2_result = step1_result * (9/5)
steps.append(f"Step 2: Multiply {step1_result}°C by 9/5 = {step2_result}")

# Step 3: Add 32
fahrenheit = step2_result + 32
steps.append(f"Step 3: Add 32 to {step2_result} = {fahrenheit}°F")

return {
    'from_unit': 'Kelvin',
    'to_unit': 'Fahrenheit',
    'input_value': kelvin,
```

[Review next file >](#)

```
        'output_value': round(fahrenheit, 2),
        'steps': steps,
        'formula': 'F = (K - 273.15) * 9/5 + 32'
    }

def convert_temperature(self, value, from_unit, to_unit):
    """
    Convert temperature between any two units

    Args:
        value (float): Temperature value
        from_unit (str): Source unit ('C', 'F', or 'K')
        to_unit (str): Target unit ('C', 'F', or 'K')

    Returns:
        dict: Conversion result
    """
    from_unit = from_unit.upper()
    to_unit = to_unit.upper()

    if from_unit == to_unit:
        return {
            'from_unit': self.units[Review next file >],
            'to_unit': self.units[to_unit],
```

```

        'input_value': value,
        'output_value': value,
        'steps': [f"No conversion needed: {value}{from_unit} = {value}{to_unit}"],
        'formula': 'No conversion required'
    }

# Define conversion methods
conversions = {
    ('C', 'F'): self.celsius_to_fahrenheit,
    ('F', 'C'): self.fahrenheit_to_celsius,
    ('C', 'K'): self.celsius_to_kelvin,
    ('K', 'C'): self.kelvin_to_celsius,
    ('F', 'K'): self.fahrenheit_to_kelvin,
    ('K', 'F'): self.kelvin_to_fahrenheit
}

if (from_unit, to_unit) in conversions:
    return conversions[(from_unit, to_unit)](value)
else:
    return {'error': f'Invalid conversion from {from_unit} to {to_unit}'}

def get_common_temperatures(self):
    """Get common temperature refer Review next file >
    return {

```

```

        'Absolute Zero': {'K': 0, 'C': -273.15, 'F': -459.67},
        'Freezing Point of Water': {'K': 273.15, 'C': 0, 'F': 32},
        'Boiling Point of Water': {'K': 373.15, 'C': 100, 'F': 212},
        'Room Temperature': {'K': 293.15, 'C': 20, 'F': 68},
        'Body Temperature': {'K': 310.15, 'C': 37, 'F': 98.6}
    }

def main():
    """Interactive temperature converter"""
    converter = TemperatureConverter()

    print("Temperature Converter")
    print("=" * 40)
    print("Available units: C (Celsius), F (Fahrenheit), K (Kelvin)")
    print()

    while True:
        try:
            # Get user input
            value = float(input("Enter temperature value: "))
            from_unit = input("Enter source unit (C/F/K): ").upper()
            to_unit = input("Enter target unit (C/F/K): ").upper()

```

```

# Validate units
if from_unit not in ['C', 'F', 'K'] or to_unit not in ['C', 'F', 'K']:
    print("X Invalid unit. Please use C, F, or K.")
    continue

# Perform conversion
result = converter.convert_temperature(value, from_unit, to_unit)

if 'error' in result:
    print(f"X {result['error']}")
else:
    print(f"\n✓ Conversion Result:")
    print(f"  {result['input_value']}°{from_unit} = {result['output_value']}°{to_unit}")
    print(f"\n💡 Formula: {result['formula']}")
    print(f"\n🔍 Steps:")
    for i, step in enumerate(result['steps'], 1):
        print(f"  {i}. {step}")

except ValueError:
    print("X Please enter a valid number.")
except KeyboardInterrupt:
    print("\n\n👋 Goodbye!")
    break

print("\n" + "=" * 40)

```

Review next file >

```

print("\n\n👋 Goodbye!")
break

print("\n" + "=" * 40)

# Ask if user wants to see common temperatures
show_common = input("Show common temperature reference points? (y/n): ").lower()
if show_common == 'y':
    print("\n💡 Common Temperature Reference Points:")
    print("-" * 50)
    common = converter.get_common_temperatures()
    for name, temps in common.items():
        print(f"{name}:")
        print(f"  {temps['K']}K = {temps['C']}°C = {temps['F']}°F")
    print()

continue_choice = input("Convert another temperature? (y/n): ").lower()
if continue_choice != 'y':
    print("👋 Goodbye!")
    break

if __name__ == "__main__":
    main()

```

Review next file >

```
🌡 Temperature Converter
=====
Available units: C (Celsius), F (Fahrenheit), K (Kelvin)

Enter temperature value: & c:/Users/solle/AppData/Local/Programs/Python/Python313/python.exe
c:/Users/solle/OneDrive/Desktop/AIAC/LAB-03/temperature_converter.py
✖ Please enter a valid number.

=====
Show common temperature reference points? (y/n): y

🌡 Common Temperature Reference Points:
-----
Absolute Zero:
    0K = -273.15°C = -459.67°F
Freezing Point of Water:
    273.15K = 0°C = 32°F
Boiling Point of Water:
    373.15K = 100°C = 212°F
Room Temperature:
    293.15K = 20°C = 68°F
Body Temperature:
    310.15K = 37°C = 98.6°F

Convert another temperature? (y/n): y
Enter temperature value: 100
Enter source unit (C/F/K): c
Enter target unit (C/F/K): f

✅ Conversion Result:
    100.0°C = 212.0°F

💡 Formula: F = (C × 9/5) + 32
💡 Formula: F = (C × 9/5) + 32

🔍 Steps:
1. Step 1: Multiply 100.0°C by 9/5 = 180.0
2. Step 2: Add 32 to 180.0 = 212.0°F

=====
Show common temperature reference points? (y/n): n
Convert another temperature? (y/n):
👉 Goodbye!
```