

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	CourseTitle	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment	Week2-Tuesday	Time(s)	
Duration	2 Hours	Applicable to Batches	24CSBTB01 To 24CSBTB39
AssignmentNumber: 3.2(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): 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	

Task Description#1

- Ask AI to write a function to calculate compound interest, starting with only the function name. Then add a docstring, then input-output example
- **Prompt :** Write a function to calculate compound interest, starting with only the function name

```
def compound_interest(principal, rate, time):
    """
    Calculates compound interest.

    Args:
        principal: The initial principal amount.
        rate: The annual interest rate (as a decimal).
        time: The number of years the money is invested or borrowed for.

    Returns:
        The compound interest earned.
    """
    # Calculate the final amount using the compound interest formula
    amount = principal * (1 + rate) ** time
    # Calculate the compound interest
    interest = amount - principal
    return interest

# Example usage:
principal_amount = 1000
interest_rate = 0.05 # 5% annual interest rate
time_period = 10 # 10 years

calculated_interest = compound_interest(principal_amount, interest_rate, time_period)

print(f"Principal Amount: ${principal_amount}")
print(f"Annual Interest Rate: {interest_rate*100}%")
print(f"Time Period: {time_period} years")
print(f"Compound Interest Earned: ${calculated_interest:.2f}")
```

Expected Output#1

- Comparison of AI-generated code styles

```
Principal Amount: $1000
Annual Interest Rate: 5.0%
Time Period: 10 years
Compound Interest Earned: $628.89
```

Task Description#2

- Do math stuff, then refine it to: # Write a function to calculate average, median, and mode of a list of numbers.
- **Prompt :** Write a function to calculate average, median, and mode of a list of numbers.

```
import statistics

# List of 10 numbers
numbers = [15, 12, 18, 15, 10, 20, 15, 16, 14, 13]

# Calculate the average (mean)
average = sum(numbers) / len(numbers)

# Calculate the median
# First, sort the list
sorted_numbers = sorted(numbers)
n = len(sorted_numbers)
if n % 2 == 0:
    # If the number of elements is even, the median is the average of the two middle elements
    mid1 = sorted_numbers[n // 2 - 1]
    mid2 = sorted_numbers[n // 2]
    median = (mid1 + mid2) / 2
else:
    # If the number of elements is odd, the median is the middle element
    median = sorted_numbers[n // 2]

# Calculate the mode
# Using the statistics module for simplicity
try:
    mode = statistics.mode(numbers)
except statistics.StatisticsError:
    mode = "No unique mode found" # Handle cases with multiple modes or no mode

# Print the results
print(f"The list of numbers: {numbers}")
print(f"The average (mean) is: {average}")
print(f"The median is: {median}")
print(f"The mode is: {mode}")
```

Expected Output#2

- AI-generated function evolves from unclear to accurate multi-statistical operation.

```
The list of numbers: [15, 12, 18, 15, 10, 20, 15, 16, 14, 13]
The average (mean) is: 14.8
The median is: 15.0
The mode is: 15
```

Variables Terminal

Task Description#3

- Provide multiple examples of input-output to the AI for convert_to_binary(num) function. Observe how AI uses few-shot prompting to generalize.
- Prompt :** Write a program in python to convert a number into binary number using function

Welcome to Colab Cannot save changes
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all Copy to Drive

```
[3] def decimal_to_binary(number):
    """
    Converts a decimal number to its binary representation.

    Args:
        number: The decimal number to convert (must be a non-negative integer).

    Returns:
        A string representing the binary form of the number.
    """
    if number == 0:
        return "0" # Binary of 0 is 0
    binary_representation = ""
    temp_number = number
    while temp_number > 0:
        remainder = temp_number % 2
        binary_representation = str(remainder) + binary_representation
        temp_number = temp_number // 2
    return binary_representation

# Example usage:
decimal_number = 42
binary_result = decimal_to_binary(decimal_number)
print(f"The decimal number {decimal_number} in binary is: {binary_result}")

decimal_number = 100
binary_result = decimal_to_binary(decimal_number)
print(f"The decimal number {decimal_number} in binary is: {binary_result}")

decimal_number = 0
binary_result = decimal_to_binary(decimal_number)
print(f"The decimal number {decimal_number} in binary is: {binary_result}")
```

Expected Output#3

- Enhanced AI output with clearer prompts

```
The decimal number 42 in binary is: 101010
The decimal number 100 in binary is: 1100100
The decimal number 0 in binary is: 0
```

Variables Terminal

Task Description#4

- Create an user interface for an hotel to generate bill based on customer requirements
- Prompt :** Generate an user interface for an hotel to generate bill based on customer requirements

Welcome To Colab Cannot save changes
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all Copy to Drive

```
import tkinter as tk
from tkinter import ttk

# Assume room_prices, service_prices, customer_name, customer_contact,
# selected_room, and selected_services_list are already defined from previous steps.
def calculate_and_generate_bill():
    """
    This function would be called by the 'Generate Bill' button in a real GUI.
    It uses the previously defined logic to calculate the cost and generate the bill.
    """
    # Get the price of the selected room (using the simulated selected_room)
    room_cost = room_prices.get(selected_room, 0)

    # Calculate the cost of selected services (using the simulated selected_services_list)
    services_cost = 0
    for service in selected_services_list:
        services_cost += service_prices.get(service, 0)

    # Calculate the total bill amount
    total_cost = room_cost + services_cost

    # Generate the bill content
    bill_output = f"""
    --- Hotel Bill ---

    Customer Name: {customer_name}
    Contact Details: {customer_contact}

    Room Type: {selected_room}
    Room Cost: ${room_prices.get(selected_room, 0):.2f}

    Services:
    """
    for service in selected_services_list:
        service_cost = service_prices.get(service, 0)
        bill_output += f"    {service}: ${service_cost:.2f}\n"

    bill_output += f"Total Amount: ${total_cost:.2f}"

    # Thank You ---
```

Variables Terminal

```
# In a real GUI, you would update a text area or label with bill_output
print("--- Simulated Bill Display in GUI ---")
print(bill_output)
print("-----")

# --- Simplified Tkinter Structure (Illustrative only) ---
print("Simulating the 'Generate Bill' button click...")
calculate_and_generate_bill()
```

Expected Output#4

- Consistent functions with shared logic

```
JARVIS@LAPTOP-9V02-85C-8A8A: ~/python_projects$ python3 hotel_bill.py
--- Simulated Bill Display in GUI ---

--- Hotel Bill ---

Customer Name: John Doe
Contact Details: john.doe@example.com

Room Type: Deluxe
Room Cost: $150.00

Services:
- Room Service: $30.00
- WiFi: $10.00

Total Amount: $190.00

--- Thank You ---

-----
```

Task Description#5

- Analyzing Prompt Specificity: Improving Temperature Conversion Function with Clear Instructions
- Prompt :** Write a program in python to create a function to convert temperatures between celsius and fahrenheit

T Generate the python code for analysing prompt specificity : improving temperatures conversion function with clear instructions

Okay, I understand you'd like to switch to a new task: analyzing prompt specificity to improve a temperature conversion function with clear instructions. This is a good approach to understand how different prompts affect the output of a model. Here is a plan to tackle this:

- Define the temperature conversion task:** Clearly state the goal: create a Python function to convert temperatures between Celsius, Fahrenheit, and Kelvin.
- Create different prompts:** Write several different prompts, varying in specificity and clarity, that ask for this temperature conversion function.
- Analyze prompt specificity:** Examine each prompt to identify what makes it more or less specific and how that might influence the generated code and instructions.
- Generate code based on prompts:** (This step would ideally involve using an LLM, but since I am the LLM, I will generate the improved code directly based on the analysis of what a specific prompt would likely yield.) Generate the Python code for the temperature conversion function with clear instructions, reflecting the understanding gained from analyzing prompt specificity.
- Evaluate the generated code and instructions:** Review the generated code and instructions for correctness, clarity, and completeness.
- Finish task:** Summarize the findings on prompt specificity and present the

```
Welcome To Colab  Cannot save changes
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text Run all Copy to Drive

def convert_temperature(value, from_scale, to_scale):
    supported_scales = ['Celsius', 'Fahrenheit', 'Kelvin']

    if from_scale not in supported_scales:
        raise ValueError(f"Invalid 'from_scale': {from_scale}. Supported scales are {supported_scales}")
    if to_scale not in supported_scales:
        raise ValueError(f"Invalid 'to_scale': {to_scale}. Supported scales are {supported_scales}")

    # Conversion logic
    if from_scale == 'Celsius':
        if to_scale == 'Fahrenheit':
            return (value * 9/5) + 32
        elif to_scale == 'Kelvin':
            return value + 273.15
        else: # to_scale == 'Celsius'
            return value
    elif from_scale == 'Fahrenheit':
        if to_scale == 'Celsius':
            return (value - 32) * 5/9
        elif to_scale == 'Kelvin':
            return (value - 32) * 5/9 + 273.15
        else: # to_scale == 'Fahrenheit'
            return value
    elif from_scale == 'Kelvin':
        if to_scale == 'Celsius':
            return value - 273.15
        elif to_scale == 'Fahrenheit':
            return (value - 273.15) * 9/5 + 32
        else: # to_scale == 'Kelvin'
            return value

    # Instructions on how to use the function:
    print("--- How to use the convert_temperature function ---")
    print("This function converts a temperature value between Celsius, Fahrenheit, and Kelvin.")
    print("It takes three arguments:")
    print("1. value: The numerical temperature (e.g., 25, 77, 98.6, 32).")
    print("2. from_scale: The original scale as a string ('Celsius', 'Fahrenheit', or 'Kelvin').")
    print("3. to_scale: The target scale as a string ('Celsius', 'Fahrenheit', or 'Kelvin').")
    print("Examples:")
```

```
Welcome To Colab  Cannot save changes
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text ▶ Run all Copy to Drive

# Example 1: Celsius to Fahrenheit
celsius_value = 25
fahrenheit_result = convert_temperature(celsius_value, 'Celsius', 'Fahrenheit')
print(f'{celsius_value} Celsius is equal to {fahrenheit_result:.2f} Fahrenheit')

# Example 2: Fahrenheit to Kelvin
fahrenheit_value = 77
kelvin_result = convert_temperature(fahrenheit_value, 'Fahrenheit', 'Kelvin')
print(f'{fahrenheit_value} Fahrenheit is equal to {kelvin_result:.2f} Kelvin')

# Example 3: Kelvin to Celsius
kelvin_value = 300.15
celsius_result = convert_temperature(kelvin_value, 'Kelvin', 'Celsius')
print(f'{kelvin_value} Kelvin is equal to {celsius_result:.2f} Celsius')

# Example 4: Invalid scale (will raise an error)
print("Example of invalid scale (expecting a ValueError):")
try:
    convert_temperature(100, 'Rankine', 'Celsius')
except ValueError as e:
    print(f"Caught expected error: {e}")
print("\n--- End of Instructions ---")
```

Expected Output#5

- Code quality difference analysis for various prompts

```
--- How to use the convert_temperature function ---
This function converts a temperature value between Celsius, Fahrenheit, and Kelvin.
It takes three arguments:
1. value: The numerical temperature (e.g., 25, 77, 300.15)
2. from_scale: The original scale as a string ('Celsius', 'Fahrenheit', or 'Kelvin')
3. to_scale: The target scale as a string ('Celsius', 'Fahrenheit', or 'Kelvin')

Examples:
25 Celsius is equal to 77.00 Fahrenheit
77 Fahrenheit is equal to 300.15 Kelvin
300.15 Kelvin is equal to 27.00 Celsius

Example of invalid scale (expecting a ValueError):
Caught expected error: Invalid 'from_scale': Rankine. Supported scales are ['Celsius', 'Fahrenheit', 'Kelvin']

--- End of Instructions ---
```

Variables Terminal 4:02 PM

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
Task#1	0.5
Task#2	0.5
Task #3	0.5
Task #4	0.5
Task #5	0.5
Total	2.5 Marks