

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	24CS002P C215	CourseTitle	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment	Week2- Tuesday	Time(s)	
Duration	2 Hours	Applicableto Batches	24CSBTB01 To 24CSBTB39
AssignmentNumber: 3.2(Present assignment number)/24(Total number of assignments)			

	Question		
--	----------	--	--

1	<div><div>Lab 3: Prompt Engineering – Improving Prompts and Context Management</div><div><div>Lab Objectives:</div><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.<div><div>Lab Outcomes (LOs):</div><div>After completing this lab, students will be able to:</div><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.</div><div><div>Task Description#1</div><ul style="list-style-type: none">Ask AI to write a function to calculate compound interest, starting with only the function name. Then add a docstring, then input-output example<div><div>PROMPT : Write a python code for function to calculate compound interest, starting with only the function name. Then add a docstring, then input-output example</div><div><div><pre># Sample inputs principal = 1000 annual_interest_rate = 0.05 compounds_per_year = 4 years = 10 # Calculate compound interest final_amount = calculate_compound_interest(principal, annual_interest_rate, compounds_per_year, years) # Print inputs and output print(f"Principal: {principal}") print(f"Annual Interest Rate: {annual_interest_rate}") print(f"Compounds per Year: {compounds_per_year}") print(f"Years: {years}") print(f"Final Amount: {final_amount}") # Expected output for these inputs is approximately 1648.72</pre></div><div><div>Principal: 1000 Annual Interest Rate: 0.05 Compounds per Year: 4 Years: 10 Final Amount: 1643.6194634870103</div></div></div></div></div><div><div>Expected Output#1</div><ul style="list-style-type: none">Comparison of AI-generated code styles<div><div>OBSERVATION :</div><div>I have added the logic to the function to calculate compound interest and created a code cell with sample inputs and the expected output.</div><div>I have completed the task of creating the <code>calculate_compound_interest</code> function with a docstring and an input-output example.</div></div><div><div>Task Description#2</div><ul style="list-style-type: none">Do math stuff, then refine it to: # Write a function to calculate average, median, and</div></div></div></div>	0 3 . 0 8 . 2 0 2 5 E Q D

mode of a list of numbers.

PROMPT : Write a code for function to calculate average, median, and mode of a list of numbers.

Expected Output#2

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

```
import statistics

def calculate_stats(numbers):
    """
    Calculates the average, median, and mode of a list of numbers.

    Args:
        numbers: A list of numbers.

    Returns:
        A dictionary containing the average, median, and mode of the list.
    """
    average = statistics.mean(numbers)
    median = statistics.median(numbers)
    try:
        mode = statistics.mode(numbers)
    except statistics.StatisticsError:
        mode = "No unique mode found" # Handle cases with no unique mode

    return {
        "average": average,
        "median": median,
        "mode": mode
    }

# Example usage:
my_list = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9]
stats = calculate_stats(my_list)
print(stats)

my_list_no_mode = [1, 2, 3, 4, 5, 6, 7, 8, 9]
stats_no_mode = calculate_stats(my_list_no_mode)
print(stats_no_mode)

{'average': 5, 'median': 5.0, 'mode': 5}
{'average': 5, 'median': 5, 'mode': 1}
```

OBSERVATION :

The code uses the `statistics` module to calculate the average, median, and mode. It includes error handling for cases where there is no unique mode and returns the results in a dictionary. The code is well-structured and includes examples.

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.

Expected Output#3

- Enhanced AI output with clearer prompts

PROMPT :

Take an example of input-output and write a code for `convert_to_binary(num)` function.

```

def convert_to_binary(num):
    """Converts a decimal number to its binary representation.

    Args:
        num: The decimal number to convert.

    Returns:
        A string representing the binary form of the input number.
    """
    if num == 0:
        return "0"
    binary = ""
    while num > 0:
        binary = str(num % 2) + binary
        num //= 2
    return binary

# Example usage:
decimal_number = 25
binary_representation = convert_to_binary(decimal_number)
print(f"The binary representation of {decimal_number} is: {binary_representation}")

decimal_number = 0
binary_representation = convert_to_binary(decimal_number)
print(f"The binary representation of {decimal_number} is: {binary_representation}")

decimal_number = 10
binary_representation = convert_to_binary(decimal_number)
print(f"The binary representation of {decimal_number} is: {binary_representation}")

```

The binary representation of 25 is: 11001
 The binary representation of 0 is: 0
 The binary representation of 10 is: 1010

iables  Terminal

OBSERVATION :

The code successfully converts a decimal number to its binary representation using a loop that repeatedly takes the remainder and integer division by 2, building the binary string in reverse. It handles the case of 0 correctly.

Task Description#4

- Create an user interface for an hotel to generate bill based on customer requirements

Expected Output#4

- Consistent functions with shared logic

PROMPT : Give a python code to an user interface for an hotel to generate bill based on customer requirements

```

# Define available services and their prices
services_prices = {
    "Room Service": 15.00,
    "Laundry": 10.50,
    "Minibar": 5.00,
    "Spa Access": 30.00
}

# Example of customer service entries (replace with actual input)
# This dictionary represents the quantities of each service a customer used
customer_service_quantities = {
    "Room Service": 2,
    "Laundry": 1,
    "Minibar": 3,
    "Spa Access": 0 # Assuming 0 for services not used
}

# Calculate the total bill
total_bill = 0.0
for service_name, quantity in customer_service_quantities.items():
    # Get the price for the service, defaulting to 0 if the service name is not found
    price = services_prices.get(service_name, 0.0)

    # Add the cost for this service (quantity * price) to the total bill
    total_bill += quantity * price

# Print the total bill
print(f"Total Bill: ${total_bill:.2f}")

```

Total Bill: \$55.50

OBSERVATION : This code calculates a total hotel bill by multiplying the quantity of each service used by its predefined price and summing these costs. It uses dictionaries to store service prices and customer usage.

Task Description#5.

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

Expected Output#5

- Code quality difference analysis for various prompts

PROMPT : Write a python code for Improving Temperature Conversion Function

```
def convert_temperature_short(value, unit_from, unit_to):
    """Converts a temperature value between Celsius, Fahrenheit, and Kelvin.

    Args:
        value: The temperature value to convert (float or int).
        unit_from: The unit of the input value ('C', 'F', or 'K').
        unit_to: The desired unit for the output value ('C', 'F', or 'K').

    Returns:
        The converted temperature value (float), or None if the units are invalid.
    """
    # Convert to Celsius first
    if unit_from == 'C':
        celsius = value
    elif unit_from == 'F':
        celsius = (value - 32) * 5/9
    elif unit_from == 'K':
        celsius = value - 273.15
    else:
        print("Invalid unit_from. Use 'C', 'F', or 'K'.")
        return None

    # Convert from Celsius to target unit
    if unit_to == 'C':
        return celsius
    elif unit_to == 'F':
        return (celsius * 9/5) + 32
    elif unit_to == 'K':
        return celsius + 273.15
    else:
        print("Invalid unit_to. Use 'C', 'F', or 'K'.")
        return None

# Example Usage:
print(f"100°C is {convert_temperature_short(100, 'C', 'F'):.2f}°F")
print(f"212°F is {convert_temperature_short(212, 'F', 'C'):.2f}°C")
print(f"0°C is {convert_temperature_short(0, 'C', 'K'):.2f}K")
```

```
100°C is 212.00°F
212°F is 100.00°C
0°C is 273.15K
```

OBSERVATION :

The code effectively converts temperatures between Celsius, Fahrenheit, and Kelvin, with clear steps and basic error handling for invalid units.

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