| | | | |
|---|---|---|---|
| **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<br>2. Dr. T Sampath Kumar<br>3. Mr. S Naresh Kumar<br>4. Dr. V. Rajesh<br>5. Dr. Brij Kishore<br>6. Dr Pramoda Patro<br>7. Dr. Venkataramana<br>8. Dr. Ravi Chander<br>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 | **Applicableto Batches** | 24CSBTB01 To 24CSBTB39 |
| **AssignmentNumber:3.2**(Present assignment number)/**24**(Total number of assignments) | | | |

| **Q.No.** | **Question** | ***Expected Time to complete*** |
|---|---|---|
| 1 | Lab 3: Prompt Engineering – Improving Prompts and Context Management<br>**Lab Objectives:**<br>To understand how prompt structure and wording influence AI-generated code.<br>● To explore how context (like comments and function names) helps AI generate relevant output.<br>● To evaluate the quality and accuracy of code based on prompt clarity.<br>● To develop effective prompting strategies for AI-assisted programming.<br>**Lab Outcomes (LOs):**<br>After completing this lab, students will be able to:<br>● Generate Python code using Google Gemini in Google Colab.<br>● Analyze the effectiveness of code explanations and suggestions by Gemini.<br>● Set up and use Cursor AI for AI-powered coding assistance.<br>● Evaluate and refactor code using Cursor AI features.<br>● Compare AI tool behavior and code quality across different platforms. | 03.08.2025 EOD |

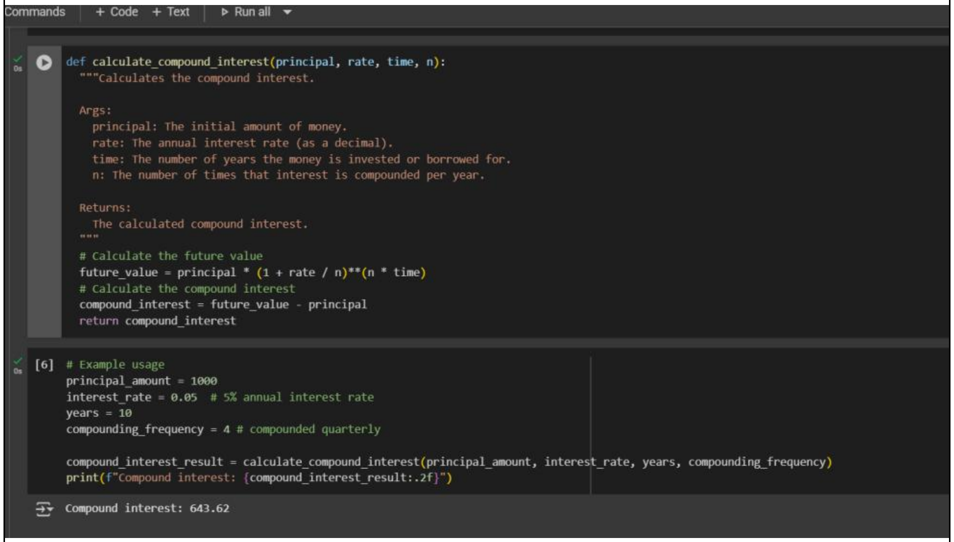
**Task Description#1**

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

**Expected Output#1**

- Comparison of AI-generated code styles

PROMPT: generate a python code for a function to calculate compound interest, starting with only the function name. Then add a docstring, then input-output example



```
def calculate_compound_interest(principal, rate, time, n):
  """Calculates the compound interest.

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

  Returns:
    The calculated compound interest.
  """
  # Calculate the future value
  future_value = principal * (1 + rate / n)**(n * time)
  # Calculate the compound interest
  compound_interest = future_value - principal
  return compound_interest
```

```
[6] # Example usage
    principal_amount = 1000
    interest_rate = 0.05  # 5% annual interest rate
    years = 10
    compounding_frequency = 4 # compounded quarterly

    compound_interest_result = calculate_compound_interest(principal_amount, interest_rate, years, compounding_frequency)
    print(f"Compound interest: {compound_interest_result:.2f}")
```

```
Compound interest: 643.62
```

**OBSERVATION:** Based on the provided code and the output, the observation is that for a principal amount of $1000 invested at a 5% annual interest rate for 10 years, compounded quarterly, the calculated compound interest is $$1000 invested at a 5% annual interest rate for 10 years, compounded quarterly, the calculated compound interest is $643.62.

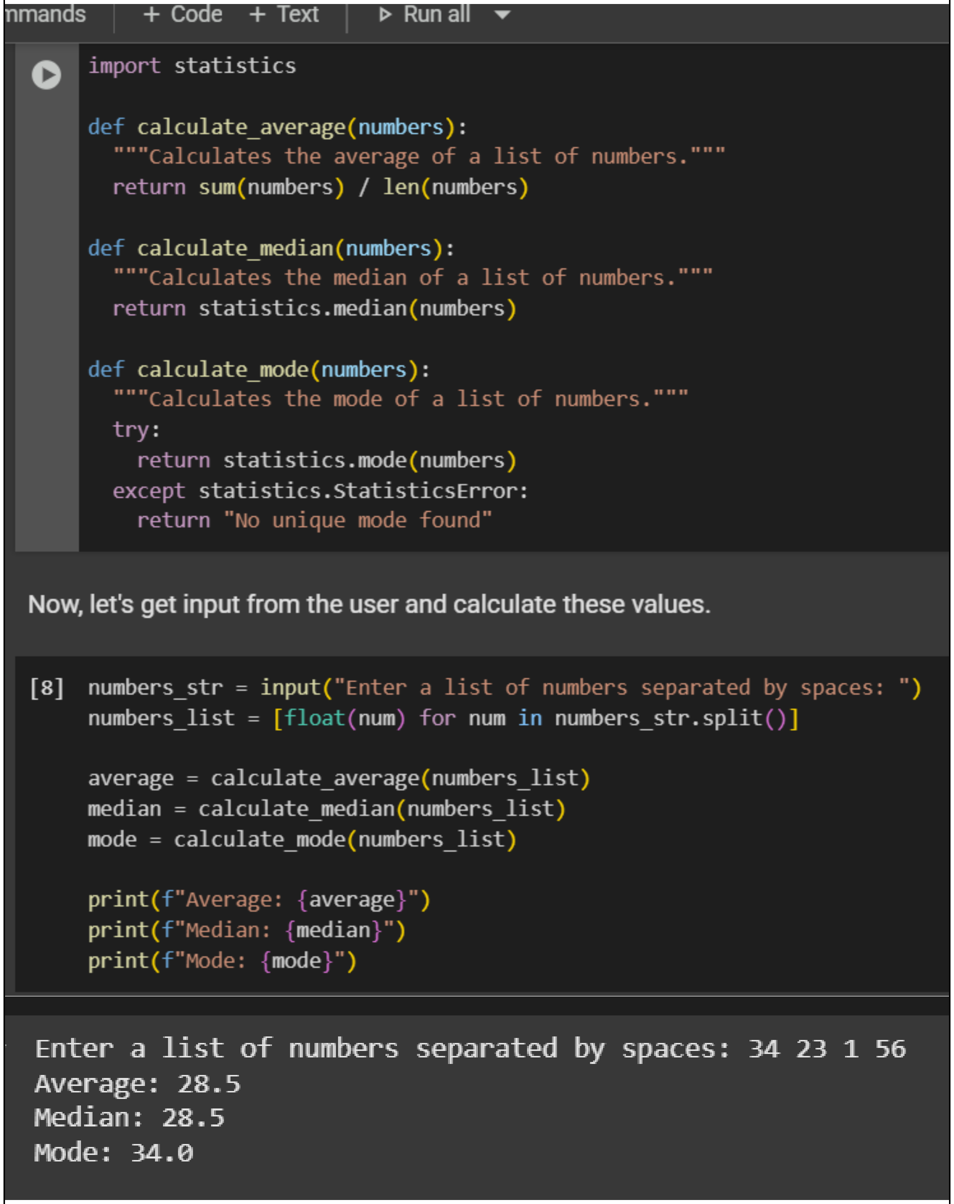
**Task Description#2**

- Do math stuff, then refine it to: # Write a 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.

PROMPT: Generate a python code to calculate average, median, and mode of a list of numbers and give user input.



```python
import statistics

def calculate_average(numbers):
  """Calculates the average of a list of numbers."""
  return sum(numbers) / len(numbers)

def calculate_median(numbers):
  """Calculates the median of a list of numbers."""
  return statistics.median(numbers)

def calculate_mode(numbers):
  """Calculates the mode of a list of numbers."""
  try:
    return statistics.mode(numbers)
  except statistics.StatisticsError:
    return "No unique mode found"
```

Now, let's get input from the user and calculate these values.

```python
numbers_str = input("Enter a list of numbers separated by spaces: ")
numbers_list = [float(num) for num in numbers_str.split()]

average = calculate_average(numbers_list)
median = calculate_median(numbers_list)
mode = calculate_mode(numbers_list)

print(f"Average: {average}")
print(f"Median: {median}")
print(f"Mode: {mode}")
```

```
Enter a list of numbers separated by spaces: 34 23 1 56
Average: 28.5
Median: 28.5
Mode: 34.0
```

**OBSERVATION:** The code utilizes the statistics module for calculating median and mode. Key Python keywords include def for function definitions, return for output, import for modules, input() for user interaction, list comprehension for data processing, print() for displaying results, and try...except for error handling, specifically for cases with no unique mode.

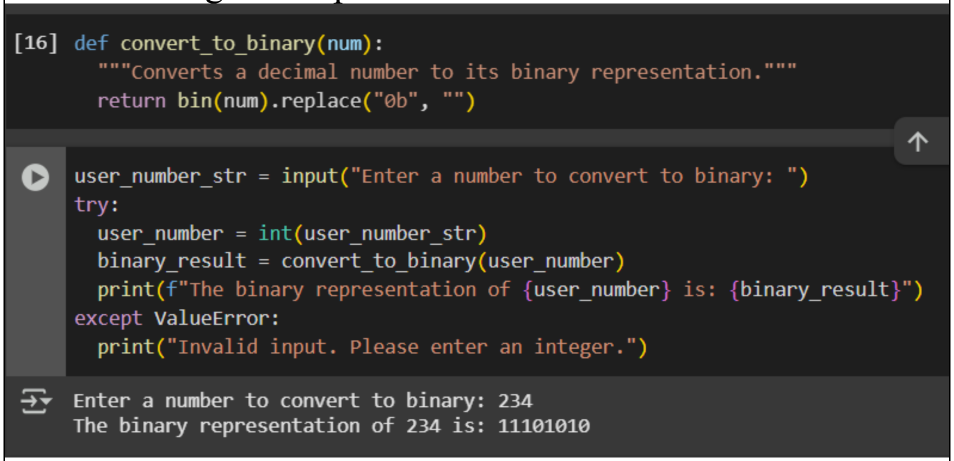
**Task Description#3**

- 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: give a python code to convert numbers into binary numbers using user input..



```python
def convert_to_binary(num):
  """Converts a decimal number to its binary representation."""
  return bin(num).replace("0b", "")
```

```python
user_number_str = input("Enter a number to convert to binary: ")
try:
  user_number = int(user_number_str)
  binary_result = convert_to_binary(user_number)
  print(f"The binary representation of {user_number} is: {binary_result}")
except ValueError:
  print("Invalid input. Please enter an integer.")
```

```
Enter a number to convert to binary: 234
The binary representation of 234 is: 11101010
```

**OBSERVATION:** Based on the provided code and output, the observation is that the code successfully takes an integer input from the user (234 in this case) and converts it to its binary representation (11101010) using the convert_to_binary function. The code also includes error handling for non-integer input.

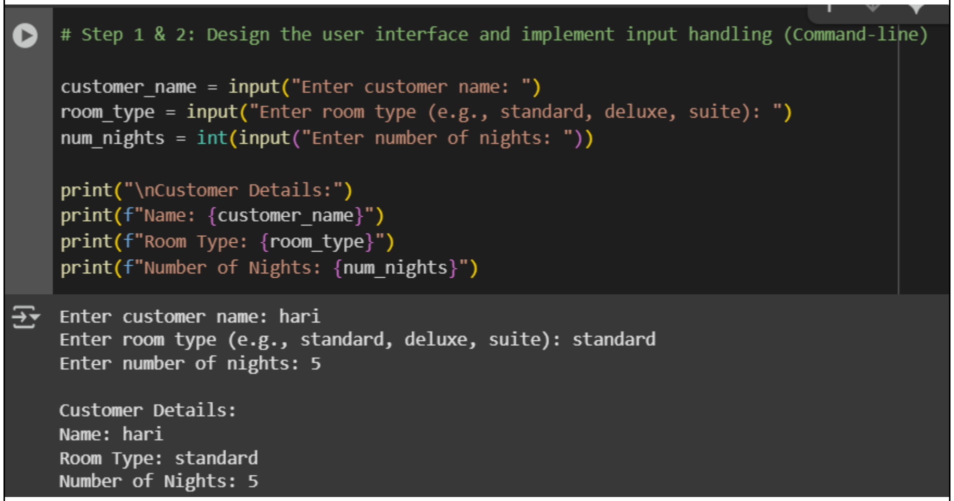
**Task Description#4**

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

**Expected Output#4**

- Consistent functions with shared logic

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



```python
# Step 1 & 2: Design the user interface and implement input handling (Command-line)

customer_name = input("Enter customer name: ")
room_type = input("Enter room type (e.g., standard, deluxe, suite): ")
num_nights = int(input("Enter number of nights: "))

print("\nCustomer Details:")
print(f"Name: {customer_name}")
print(f"Room Type: {room_type}")
print(f"Number of Nights: {num_nights}")
```

```
Enter customer name: hari
Enter room type (e.g., standard, deluxe, suite): standard
Enter number of nights: 5

Customer Details:
Name: hari
Room Type: standard
Number of Nights: 5
```

**OBSERVATION:** Based on the provided code and the output, the observation is that the code successfully prompts the user for customer information (name, room type, and number of nights) and then displays the entered details. In this specific execution, the customer name entered was "hari", the room type was "standard", and the number of nights was 5

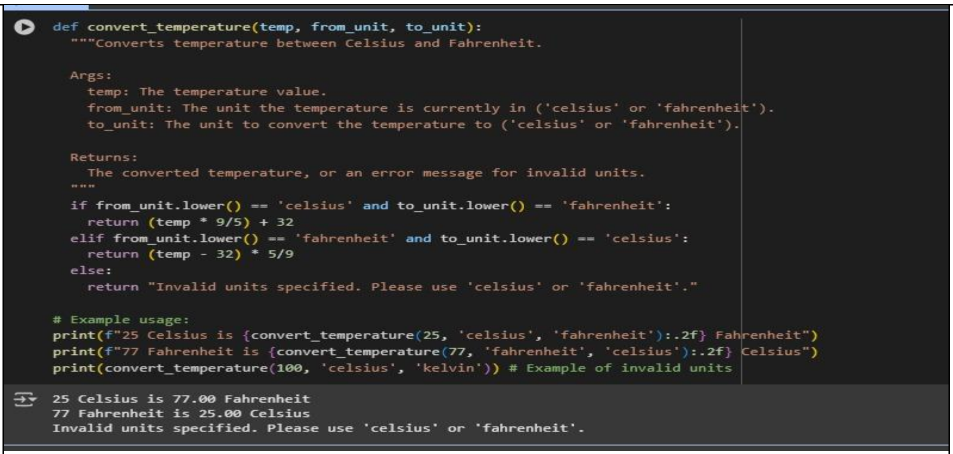
**Task Description#5**

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

**Expected Output#5**

- Code quality difference analysis for various prompts

**PROMPT:** generate a python code for Improving Temperature Conversion Function



```python
def convert_temperature(temp, from_unit, to_unit):
  """Converts temperature between Celsius and Fahrenheit.

  Args:
    temp: The temperature value.
    from_unit: The unit the temperature is currently in ('celsius' or 'fahrenheit').
    to_unit: The unit to convert the temperature to ('celsius' or 'fahrenheit').

  Returns:
    The converted temperature, or an error message for invalid units.
  """
  if from_unit.lower() == 'celsius' and to_unit.lower() == 'fahrenheit':
    return (temp * 9/5) + 32
  elif from_unit.lower() == 'fahrenheit' and to_unit.lower() == 'celsius':
    return (temp - 32) * 5/9
  else:
    return "Invalid units specified. Please use 'celsius' or 'fahrenheit'."

# Example usage:
print(f"25 Celsius is {convert_temperature(25, 'celsius', 'fahrenheit'):.2f} Fahrenheit")
print(f"77 Fahrenheit is {convert_temperature(77, 'fahrenheit', 'celsius'):.2f} Celsius")
print(convert_temperature(100, 'celsius', 'kelvin')) # Example of invalid units
```

```
25 Celsius is 77.00 Fahrenheit
77 Fahrenheit is 25.00 Celsius
Invalid units specified. Please use 'celsius' or 'fahrenheit'.
```

**OBSERVATION:** This function is a simple and correct implementation of the formula for converting Celsius to Fahrenheit. It takes a single argument (celsius) and returns the calculated Fahrenheit value. The docstring provides a brief explanation of what the function does. This code was generated based on a general request for a temperature conversion function.

**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** |