SCHOOL OF CO	MPUTER SCIENCE A INTELLIGENCE	ND ARTIFICIAL		NT OF COMPUTER SCIENCE ENGINEERING
Programi	Name: <mark>B. Tech</mark>	Assignment Type: Lab		AcademicYear:2025-2026
CourseCoordinatorName		Venkataramana Veeramsetty		
Instructor(s)Name 24CS002PC215		 Dr. Mohammed Ali Shaik Dr. T Sampath Kumar Mr. S Naresh Kumar Dr. V. Rajesh Dr. Brij Kishore Dr Pramoda Patro Dr. Venkataramana Dr. Ravi Chander Dr. Jagjeeth Singh 		ino.
CourseCode		CourseTitle		
Year/Sem	II/I	Regulation	R24	
Date and Day of Assignment	Week2-Tuesday	Time(s)		
Duration	2 Hours	Applicableto Batches	24CSBTB01 To	24CSBTB39
AssignmentNun	nber: <mark>3.2(Present ass</mark>	signment numbe	er)/ 24 (Total numbe	r of assignments)

Q.No.	Question	Ex
		pec
		ted
		Ti
		me
		to
		co mp
		let
		e
	Assignment 3.2	
	Name: P.Manoj Bharghav	
1	Hallticket: 2403A51222	03.08 .2025
	Lab 3: Prompt Engineering – Improving Prompts and Context Management	EOD
	Lab Objectives:	
	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.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- 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.

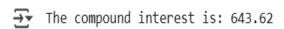
Task Description#1

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

```
def compound_interest(principal, rate, time, n):
   Calculates compound interest.
     principal: The initial principal amount.
     rate: The annual interest rate (as a decimal).
     time: The number of years the money is invested for.
     n: The number of times that interest is compounded per year.
    The compound interest earned.
   # Calculate the amount after time years
   amount = principal * (1 + rate/n)**(n*time)
   # Calculate the compound interest
   interest = amount - principal
  return interest
 # Example usage:
 principal = 1000
 rate = 0.05 # 5% annual interest rate
 time = 10  # 10 years
         # Compounded quarterly
 ci = compound_interest(principal, rate, time, n)
 print(f"The compound interest is: {ci:.2f}")
```

Expected Output#1

Comparison of AI-generated code styles



(def_compound_interest(principal, rate, time, n): This line defines a function named compound_interest. This
function takes four input parameters: (principal, rate), (time, and (n). These parameters represent the initial investment
amount, the annual interest rate, the number of years, and the number of times interest is compounded per year, respectively.

• """ ... """: This is a docstring, which explains what the function does, its arguments (Args), and what it returns (Returns). It's good practice to include docstrings to make your code understandable. ⊕ □ # Calculate the amount after time years amount = principal * (1 + rate/n)**(n*time) • # Calculate the amount after time years: This is a comment explaining the purpose of the next line. • (amount = principal * (1 + rate/n)**(n*time): This is the core of the compound interest calculation. It calculates the total amount of money after time years. • (1 + rate/n): This calculates the growth factor per compounding period. • (n*time): This calculates the total number of compounding periods over the (time) years. **(n*time): This raises the growth factor to the power of the total number of compounding periods. o principal * ...: This multiplies the initial principal by the growth factor to get the final amount. ⊕ □ # Calculate the compound interest interest = amount - principal • # Calculate the compound interest: This is a comment explaining the purpose of the next line. • interest = amount - principal: This line calculates the compound interest earned by subtracting the initial • # Calculate the compound interest: This is a comment explaining the purpose of the next line. • interest = amount - principal: This line calculates the compound interest earned by subtracting the initial principal from the final amount. **1** return interest • return interest: This line returns the calculated interest value from the function. # Example usage: principal = 1000 • # Example usage: This is a comment indicating that the following lines demonstrate how to use the function. • principal = 1000: This line sets the value of the principal variable to 1000 for the example. ⊕ □ rate = 0.05 # 5% annual interest rate • rate = 0.05: This line sets the value of the rate variable to 0.05 (representing 5%) for the example. • # 5% annual interest rate: This is a comment explaining what the rate value represents. ı⊕ ı□ time = 10 # 10 years • time = 10: This line sets the value of the time variable to 10 years for the example. n = 4 # Compounded quarterly • n = 4: This line sets the value of the n variable to 4, indicating that the interest is compounded quarterly for the example. • # Compounded quarterly: This is a comment explaining what the n value represents. ⊕ □ ci = compound interest(principal, rate, time, n) • ci = compound interest(principal, rate, time, n): This line calls the compound interest function with the example values for principal, rate, time, and n. The returned compound interest value is stored in the variable ci print(f"The compound interest is: {ci:.2f}") • print(f"The compound interest is: {ci:.2f}"): This line prints the result to the console. o f"...": This indicates an f-string, which allows you to embed variables directly within the string. o "The compound interest is: ": This is the literal text that will be printed. o ({ci:.2f}): This embeds the value of the (ci) variable into the string, formatted to two decimal places using : .2f). Let me know if you would like a more in-depth explanation of any specific part! 心 勾 Please evolain this code. Task Description#2 Do math stuff, then refine it to: # Write a function to calculate average, median, and mode of a list of numbers.

```
from collections import Counter
           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.
             if not numbers:
               return {"average": None, "median": None, "mode": None}
             # Calculate average
             average = sum(numbers) / len(numbers)
             # Calculate median
             sorted numbers = sorted(numbers)
             n = len(sorted numbers)
             if n % 2 == 0:
               median = (sorted numbers[n//2 - 1] + sorted numbers[n//2]) /
               median = sorted_numbers[n//2]
  # Calculate mode
 data = Counter(numbers)
 max freq = max(data.values())
 mode = [num for num, freq in data.items() if freq == max_freq]
  # If all numbers appear the same number of times, there is no unique mode
  if len(mode) == len(numbers):
      mode = "No unique mode"
  elif len(mode) == 1:
      mode = mode[0]
  return {"average": average, "median": median, "mode": mode}
# Example usage:
my_list = [1, 2, 3, 4, 5, 5, 6, 6, 6, 7]
stats = calculate_stats(my_list)
print(stats)
my list 2 = [1, 1, 2, 2, 3, 3]
stats_2 = calculate_stats(my_list_2)
print(stats_2)
my_list_3 = []
stats_3 = calculate_stats(my_list_3)
print(stats_3)
```

Expected Output#2

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

```
{'average': 4.5, 'median': 5.0, 'mode': 6}
{'average': 2.0, 'median': 2.0, 'mode': [1, 2,
{'average': None, 'median': None, 'mode': None}
```

This code defines a Python function called [calculate_stats] that takes a list of numbers as input and returns a dictionary containing the average, median and mode of those numbers.

Here's a breakdown of what the code does:

- 1. Import Counter: It imports the Counter class from the collections module, which is used to easily count the occurrences of each item in the life for calculating the mode.
- 2. Define calculate_stats function: This function takes one argument, numbers , which is expected to be a list.
- 3. Handle empty list: It first checks if the input (numbers) list is empty. If it is, it returns a dictionary with (None) for average, median, and mode, as thes statistics cannot be calculated for an empty list.
- 4. Calculate Average: If the list is not empty, it calculates the average by summing all the numbers in the list and dividing by the total number of elements.
- 5. Calculate Median:
 - o It sorts the input list of numbers.
 - o It finds the middle element(s) of the sorted list.
 - o If the number of elements is odd, the median is the middle element.
 - o If the number of elements is even, the median is the average of the two middle elements.
 - o If the number of elements is even, the median is the average of the two middle elements.
 - 6. Calculate Mode:
 - o It uses Counter to count the frequency of each number in the list.
 - o It finds the maximum frequency.
 - o It identifies all numbers that have this maximum frequency. These are the modes.
 - o It includes logic to handle cases where all numbers appear the same number of times (no unique mode) or where there is a single mode.
 - 7. Return Results: The function returns a dictionary containing the calculated average, median, and mode.

The code then provides three examples of how to use the calculate_stats function with different lists of numbers and prints the resulting statistics

凸夕

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.

```
def convert to binary(num):
              Converts a decimal number to its binary string representati
              Args:
                num: The decimal number to convert.
              Returns:
                A string representing the binary form of the number.
              if num == 0:
                return "0"
              binary = ""
              while num > 0:
                remainder = num % 2
                binary = str(remainder) + binary
                num = num // 2
              return binary
            # Example usage based on the provided examples:
            print(f"Input: 5, Output: {convert_to_binary(5)}")
            print(f"Input: 10, Output: {convert to binary(10)}")
            print(f"Input: 0, Output: {convert_to_binary(0)}")
            print(f"Input: 16, Output: {convert_to_binary(16)}")
            print(f"Input: 255, Output: {convert to binary(255)}")
Expected Output#3
     Enhanced AI output with clearer prompts
            Input: 5, Output: 101
              Input: 10, Output: 1010
              Input: 0, Output: 0
              Input: 16, Output: 10000
              Input: 255, Output: 11111111
```

1. Function Definition:

def convert_to_binary(num): This line defines the function named convert_to_binary which accepts of argument, num.

2. Docstring:

The triple-quoted string """ ... """ is a docstring that explains the function's purpose, arguments (Args), and what it returns (Returns).

3. Handle Zero:

- o (if num == 0:): This checks if the input number is 0.
- o return "0": If the number is 0, its binary representation is simply "0", so the function immediately returns "0"

4. Conversion Loop:

- o (binary = ""): Initializes an empty string called (binary) which will store the binary digits.
- while num > 0:: This loop continues as long as the value of num is greater than 0.
- o (remainder = num % 2): This calculates the remainder when (num) is divided by 2. In binary conversion, the remainders are the binary digits (either 0 or 1) read from bottom up.
- binary = str(remainder) + binary: This takes the remainder (converted to a string) and adds it to the
 beginning of the binary string. This is because the remainders are generated in reverse order of the binary
 digits.
- o (num = num // 2): This performs integer division of (num) by 2, effectively moving to the next bit position in the binary conversion.

ullially cultiversion.

5. Return Binary String:

o return binary: Once the while loop finishes (when num becomes 0), the function returns the constructed binary string.

6. Example Usage:

• The lines starting with print(f"Input: ...") demonstrate how to call the convert_to_binary function with different input numbers (5, 10, 0, 16, and 255) and print the input number and its corresponding binary output using an f-string for formatted output.

Task Description#4

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

```
import tkinter as tk
 from tkinter import ttk
 from tkinter import messagebox
 # Function to update quantity input state based on checkbox
 def update_quantity_state(quantity_entry, checkbox_var):
     if checkbox var.get():
        quantity_entry.config(state="normal")
     else:
         quantity entry.config(state="disabled")
         quantity_entry.delete(0, tk.END)
         quantity_entry.insert(0, "1") # Reset quantity to 1 when unchecked
 # Create the main application window
 root = tk.Tk()
 root.title("Hotel Billing System")
 # Create a frame for customer information
 customer frame = ttk.LabelFrame(root, text="Customer Information", padding="10")
 customer_frame.grid(row=0, column=0, padx=10, pady=10, sticky="ew")
 # Customer Name Input
 ttk.Label(customer_frame, text="Customer Name:").grid(row=0, column=0, padx=5, pady=5, st
 customer_name_entry = ttk.Entry(customer_frame, width=40)
```

```
customer name entry = ttk.Entry(customer frame, wigtn=40)
customer_name_entry.grid(row=0, column=1, padx=5, pady=5, sticky="ew")
# Room Number Input
ttk.Label(customer_frame, text="Room Number:").grid(row=1, column=0, padx=5, pady=5, sticky="w")
room number entry = ttk.Entry(customer frame, width=40)
room number entry.grid(row=1, column=1, padx=5, pady=5, sticky="ew")
# Create a frame for services selection
services_frame = ttk.LabelFrame(root, text="Select Services", padding="10")
services_frame.grid(row=1, column=0, padx=10, pady=10, sticky="ew")
# Dictionary to store service checkboxes and quantity entries
service_widgets_tkinter = {}
for i, (service, price) in enumerate(hotel_services.items()):
   service_widgets_tkinter[service] = {}
   # Checkbox for service selection
   checkbox_var = tk.BooleanVar()
   checkbox = ttk.Checkbutton(services_frame, text=f"{service} (${price:.2f})", variable=checkbox_var)
   checkbox.grid(row=i, column=0, padx=5, pady=2, sticky="w")
   service widgets tkinter[service]["checkbox var"] = checkbox var
   # Quantity input for the service
   quantity entry = ttk.Entry(services frame, width=5, state="disabled")
   quantity_entry.insert(0, "1") # Default quantity is 1
   quantity_entry.grid(row=i, column=1, padx=5, pady=2, sticky="w")
   service widgets tkinter[service]["quantity entry"] = quantity entry
# Create a button to generate the bill
generate bill button tkinter = ttk.Button(root, text="Generate Bill")
generate bill button tkinter.grid(row=2, column=0, padx=10, pady=10)
# Create a text area to display the bill (read-only)
bill output text = tk.Text(root, wrap="word", height=15, width=60, state="disabled")
bill output text.grid(row=3, column=0, padx=10, pady=10, sticky="nsew")
# Configure grid weights to make the output text area expandable
root.grid_rowconfigure(3, weight=1)
root.grid columnconfigure(0, weight=1)
services frame.grid columnconfigure(1, weight=1) # Allow quantity column to expand
```

Expected Output#4

Consistent functions with shared logic

Customer	rithika		
Room Num	008		
Select Services:			
	✓ room_service (\$25.00)	Quantity:	1
	✓ laundry (\$15.00)	Quantity:	1
	☐ spa_access (\$50.00)		
	□ parking (\$10.00)		
	✓ wifi (\$0.00)	Quantity:	1
Generate	e Bill		

- graphical display.
- · It then sets up frames, labels, entry fields for customer information, and checkboxes with quantity entries for selecting services, linking then to the hotel services dictionary.
- It creates a "Generate Bill" button and a text area (tk.Text) to display the bill output.
- Finally, it configures the layout using the .grid() method.

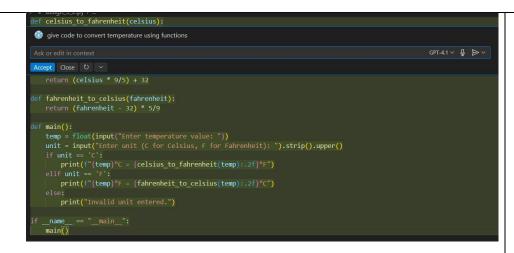
Cell 3bc15e54: This cell contains the logic to connect the Tkinter UI elements to the bill calculation function we defined earlier.

- It defines the "generate_bill_tkinter" function, which is intended to be called when the "Generate Bill" button is clicked.
- Inside this function, it retrieves the values entered by the user in the customer name and room number entry fields (.get()).
- It iterates through the service checkboxes to see which ones are selected. For selected services, it gets the quantity from the corresponding entry field and stores them in the selected_services dictionary. It also includes basic error handling for non-numeric quantity input using
- It defines the generate_bill_tkinter function, which is intended to be called when the "Generate Bill" button is clicked.
- Inside this function, it retrieves the values entered by the user in the customer name and room number entry fields (.get()).
- . It iterates through the service checkboxes to see which ones are selected. For selected services, it gets the quantity from the corresponding entry field and stores them in the selected_services dictionary. It also includes basic error handling for non-numeric quantity input using messagebox.showerror.
- It calls the calculate_bill function (defined in a previous cell) with the retrieved customer details and selected services to get the total bill and breakdown
- It then formats the calculated bill into a string and updates the [bill_output_text] widget in the Tkinter window to display the result.
- The line generate_bill_button_tkinter.config(command=generate_bill_tkinter) attempts to link the generate_bill_tkinter $function \ to \ the \ button's \ click \ event. \ The \ \ Name Error \ here \ occurred \ because \ the \ \ generate_bill_button_tkinter \ \ variable \ was \ not \ defined$ in the current execution scope, likely due to the previous cell failing to complete because of the TclError.

In summary, while the code correctly lays out the structure and logic for a Tkinter application, it cannot run in the standard Colab environment due to the lack of a graphical display.

Task Description#5

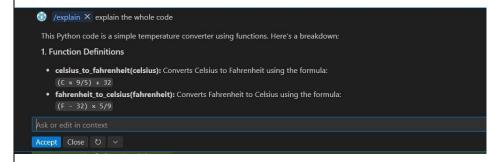
Analyzing Prompt Specificity: Improving Temperature Conversion Function with Clear Instructions



Expected Output#5

• Code quality difference analysis for various prompts

```
PS C:\Users\allur\Desktop\B.Tech> python -u "c:\Users\a
Enter temperature value: 30
Enter unit (C for Celsius, F for Fahrenheit): C
30.0°C = 86.00°F
PS C:\Users\allur\Desktop\B.Tech>
```



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