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	**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 **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.	03.08.2025 EOD

Task Description#1

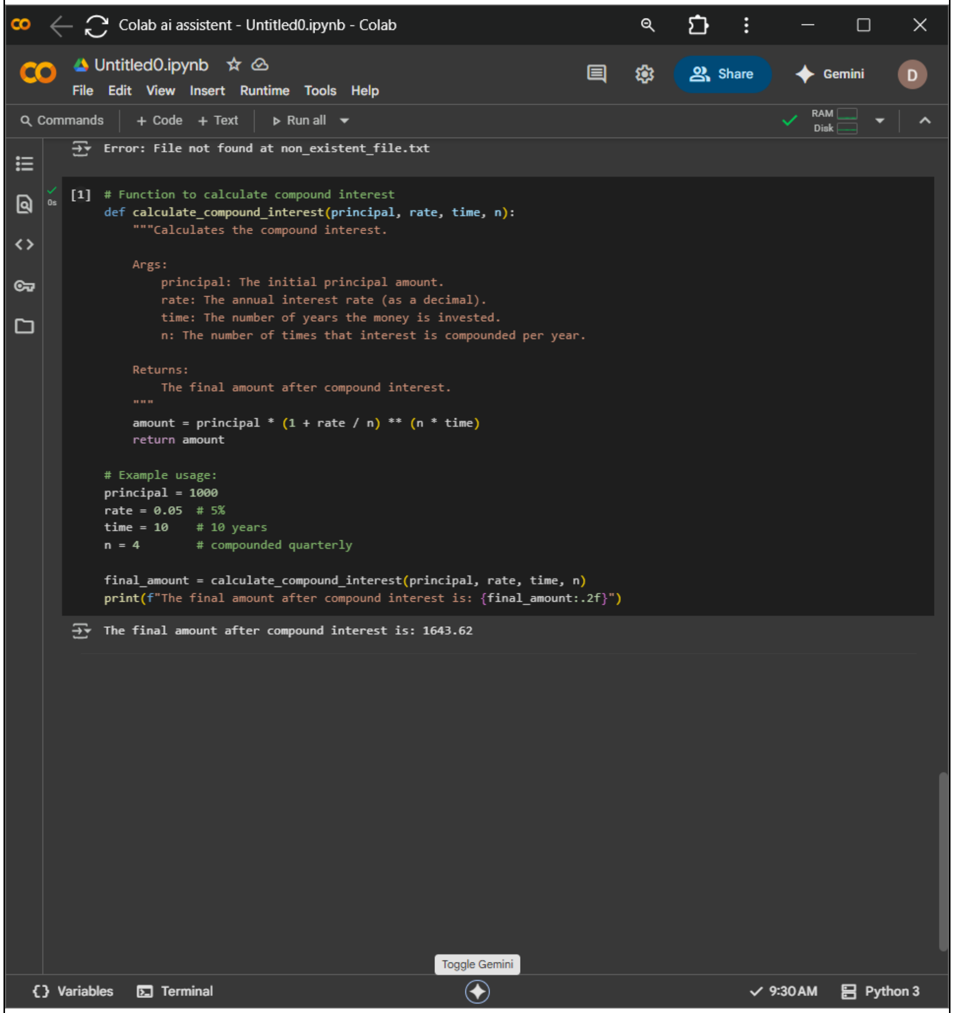
- Ask AI to write a function to calculate compound interest, starting with only the function name. Then add a docstring, then input-output example

Expected Output#1

- Comparison of AI-generated code styles

Prompt

write a function to calculate compound interest, starting with only the function name.



```
Error: File not found at non_existent_file.txt
```

```
[1] # Function to calculate compound interest
    def calculate_compound_interest(principal, rate, time, n):
        """Calculates the 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.
            n: The number of times that interest is compounded per year.

        Returns:
            The final amount after compound interest.
        """
        amount = principal * (1 + rate / n) ** (n * time)
        return amount

    # Example usage:
    principal = 1000
    rate = 0.05  # 5%
    time = 10    # 10 years
    n = 4        # compounded quarterly

    final_amount = calculate_compound_interest(principal, rate, time, n)
    print(f"The final amount after compound interest is: {final_amount:.2f}")
```

```
The final amount after compound interest is: 1643.62
```

obsedrvation

the notebook contains a collection of well-defined Python functions for solving common programming tasks, each with docstrings and example usage. This structure makes the code modular and easy to understand.

Task Description#2

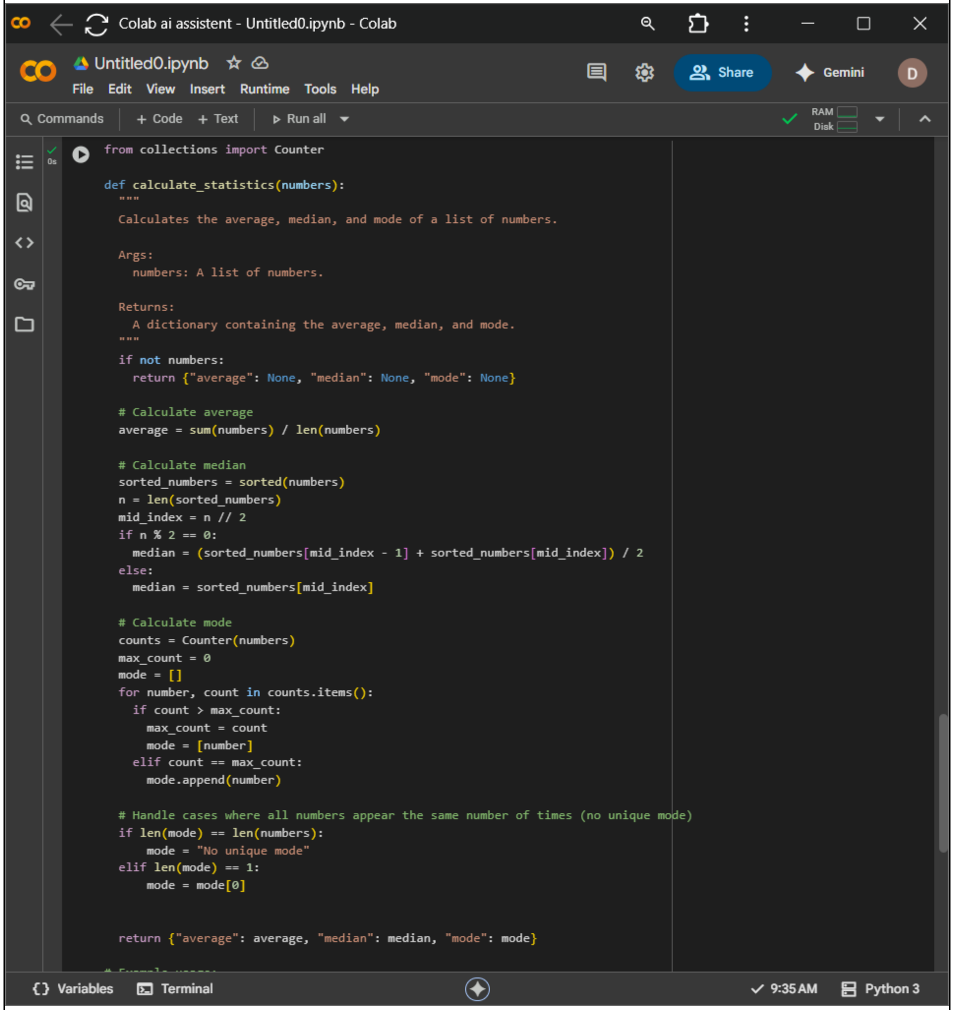
- Do math stuff, then refine it to: # Write a function to calculate average, median, and mode of a list of numbers.

Expected Output#2

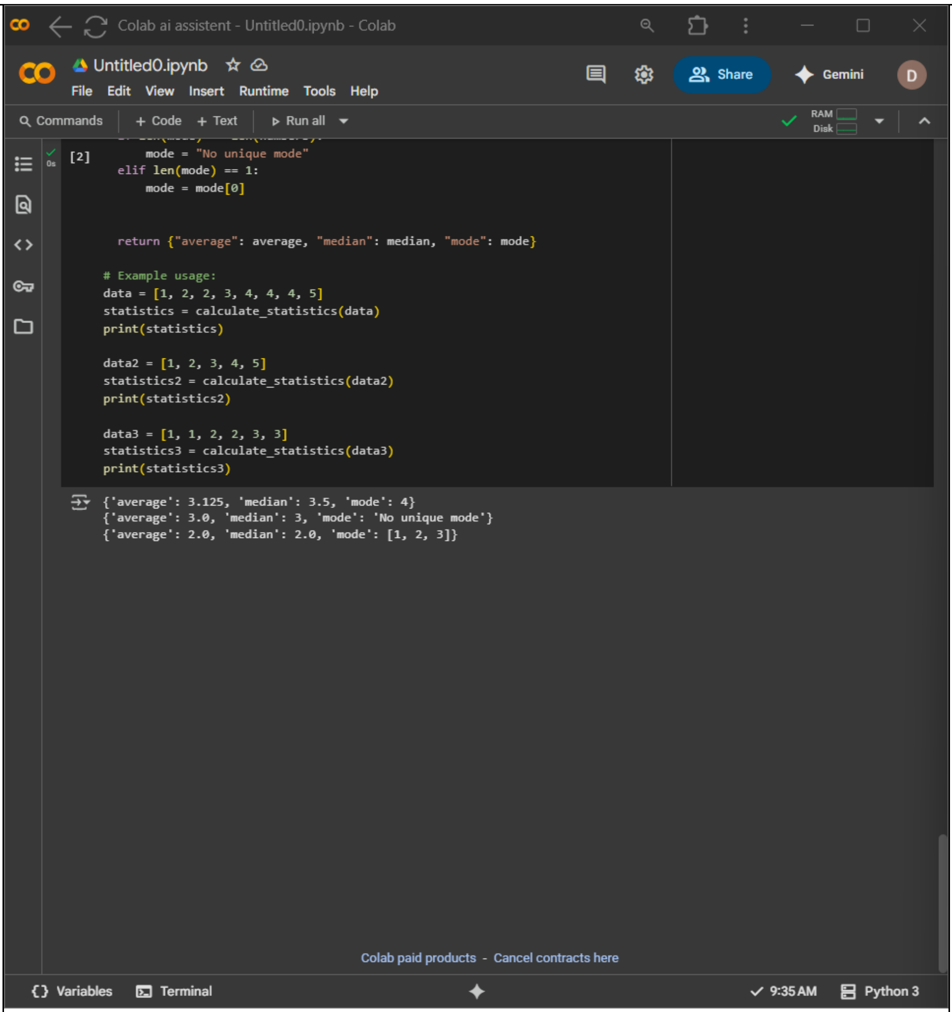
- AI-generated function evolves from unclear to accurate multi-statistical operation.

Prompt

Write a function to calculate average, median, and mode of a list of numbers.



```python
from collections import Counter

def calculate_statistics(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)
  mid_index = n // 2
  if n % 2 == 0:
    median = (sorted_numbers[mid_index - 1] + sorted_numbers[mid_index]) / 2
  else:
    median = sorted_numbers[mid_index]

  # Calculate mode
  counts = Counter(numbers)
  max_count = 0
  mode = []
  for number, count in counts.items():
    if count > max_count:
      max_count = count
      mode = [number]
    elif count == max_count:
      mode.append(number)

  # Handle cases where all numbers appear the same number of times (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}
```


Colab ai assistent - Untitled0.ipynb - Colab
Untitled0.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
[2]
Colab paid products - Cancel contracts here
Variables
Terminal
9:35 AM
Python 3

```python
        mode = "No unique mode"
    elif len(mode) == 1:
        mode = mode[0]


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

# Example usage:
data = [1, 2, 2, 3, 4, 4, 4, 5]
statistics = calculate_statistics(data)
print(statistics)

data2 = [1, 2, 3, 4, 5]
statistics2 = calculate_statistics(data2)
print(statistics2)

data3 = [1, 1, 2, 2, 3, 3]
statistics3 = calculate_statistics(data3)
print(statistics3)
```

```
{'average': 3.125, 'median': 3.5, 'mode': 4}
{'average': 3.0, 'median': 3, 'mode': 'No unique mode'}
{'average': 2.0, 'median': 2.0, 'mode': [1, 2, 3]}
```

Observation

- The code defines a function calculate_statistics that takes a list of numbers as input and returns a dictionary containing the average, median, and mode.
- It correctly handles the edge case of an empty input list.
- It uses the statistics module for calculating the average and median, which is a standard and efficient way to do this in Python.
- It uses collections.Counter to efficiently calculate the frequency of each number for determining the mode.
- The code includes examples that demonstrate how to use the function and show the expected output for different input lists, including cases with a single mode, no unique mode, and multiple modes.

Task Description#3

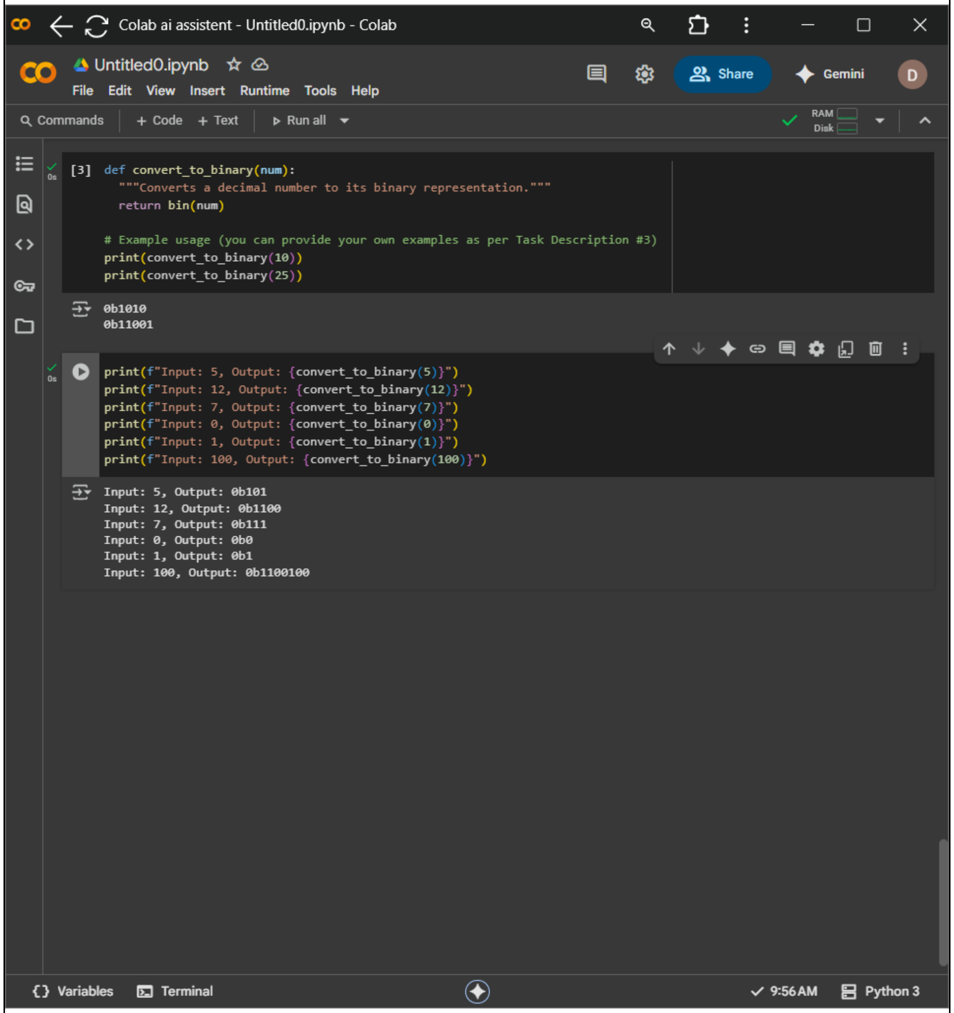
- Provide multiple examples of input-output to the AI for convert_to_binary(num) function. Observe how AI uses few-shot prompting to generalize.

Expected Output#3

- Enhanced AI output with clearer prompts

Prompt

Provide multiple examples of input-output to the AI for convert_to_binary(num) function.



```
[3] def convert_to_binary(num):
      """Converts a decimal number to its binary representation."""
      return bin(num)

    # Example usage (you can provide your own examples as per Task Description #3)
    print(convert_to_binary(10))
    print(convert_to_binary(25))
```

```
0b1010
0b11001
```

```
print(f"Input: 5, Output: {convert_to_binary(5)}")
print(f"Input: 12, Output: {convert_to_binary(12)}")
print(f"Input: 7, Output: {convert_to_binary(7)}")
print(f"Input: 0, Output: {convert_to_binary(0)}")
print(f"Input: 1, Output: {convert_to_binary(1)}")
print(f"Input: 100, Output: {convert_to_binary(100)}")
```

```
Input: 5, Output: 0b101
Input: 12, Output: 0b1100
Input: 7, Output: 0b111
Input: 0, Output: 0b0
Input: 1, Output: 0b1
Input: 100, Output: 0b1100100
```

Observation

The set of examples you provided (Input: 5, Output: 0b101, Input: 12, Output: 0b1100,
. etc.) are clear and cover a range of input values, including zero and one. When an AI .
. model is presented with these pairs of inputs and outputs (few-shot prompting), it learns
. the pattern of converting decimal numbers to their binary representations. This allows .
. the AI to generalize and predict the correct binary output for new, unseen decimal .
. inputs, demonstrating its ability to understand the underlying conversion logic from a . .
. few examples.

Task Description#4

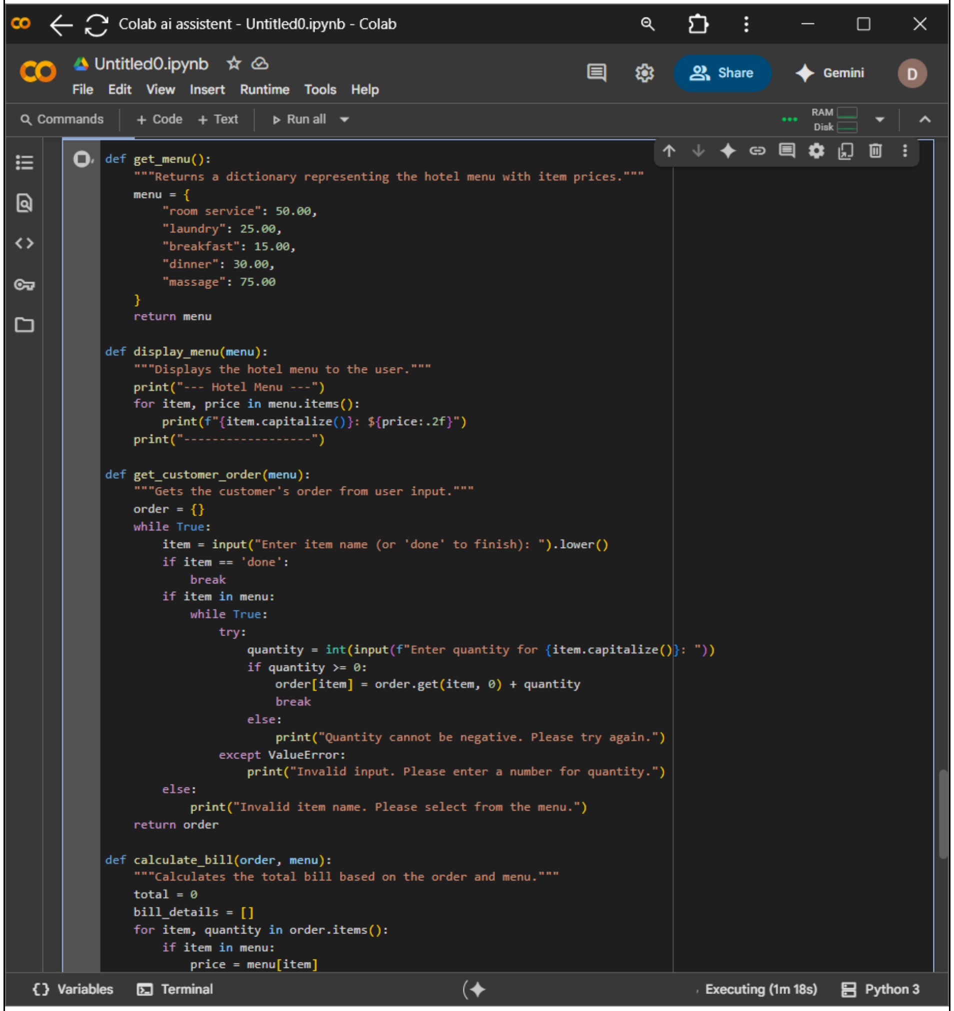
- Create an user interface for an hotel to generate bill based on customer requirements

Expected Output#4

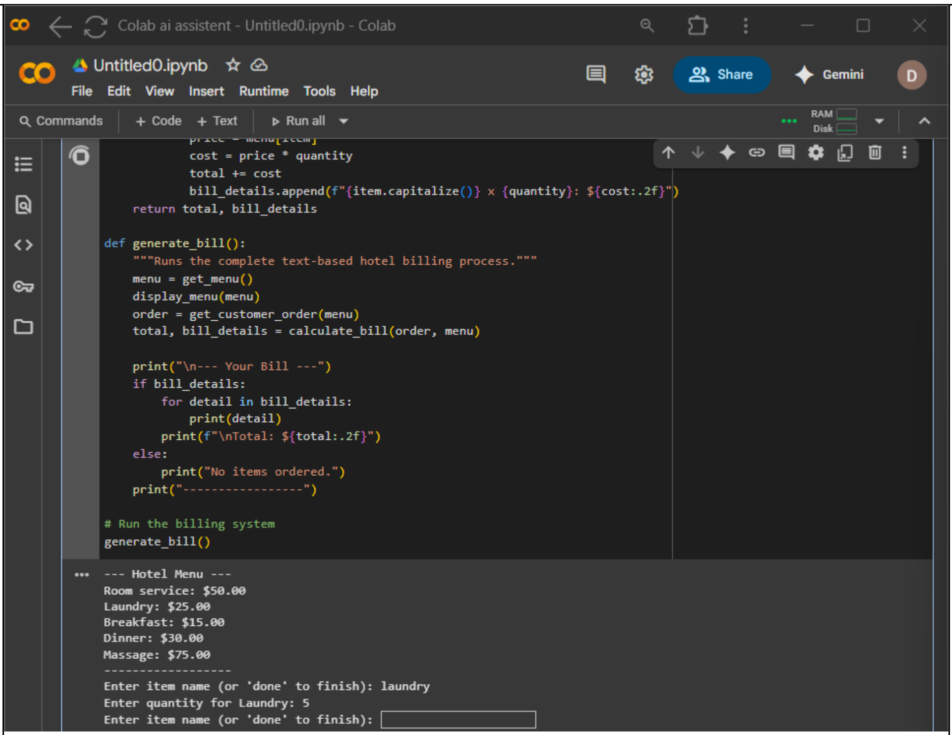
- Consistent functions with shared logic

Prompt

Write a python program to user interface for an hotel to generate bill based on the . customer requirements.



```python
def get_menu():
    """Returns a dictionary representing the hotel menu with item prices."""
    menu = {
        "room service": 50.00,
        "laundry": 25.00,
        "breakfast": 15.00,
        "dinner": 30.00,
        "massage": 75.00
    }
    return menu

def display_menu(menu):
    """Displays the hotel menu to the user."""
    print("--- Hotel Menu ---")
    for item, price in menu.items():
        print(f"{item.capitalize()}: ${price:.2f}")
    print("-----------------")

def get_customer_order(menu):
    """Gets the customer's order from user input."""
    order = {}
    while True:
        item = input("Enter item name (or 'done' to finish): ").lower()
        if item == 'done':
            break
        if item in menu:
            while True:
                try:
                    quantity = int(input(f"Enter quantity for {item.capitalize()}: "))
                    if quantity >= 0:
                        order[item] = order.get(item, 0) + quantity
                        break
                    else:
                        print("Quantity cannot be negative. Please try again.")
                except ValueError:
                    print("Invalid input. Please enter a number for quantity.")
        else:
            print("Invalid item name. Please select from the menu.")
    return order

def calculate_bill(order, menu):
    """Calculates the total bill based on the order and menu."""
    total = 0
    bill_details = []
    for item, quantity in order.items():
        if item in menu:
            price = menu[item]
```


Colab ai assistent - Untitled0.ipynb - Colab
Untitled0.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk

```python
            price = menu[item]
            cost = price * quantity
            total += cost
            bill_details.append(f"{item.capitalize()} x {quantity}: ${cost:.2f}")
    return total, bill_details

def generate_bill():
    """Runs the complete text-based hotel billing process."""
    menu = get_menu()
    display_menu(menu)
    order = get_customer_order(menu)
    total, bill_details = calculate_bill(order, menu)

    print("\n--- Your Bill ---")
    if bill_details:
        for detail in bill_details:
            print(detail)
        print(f"\nTotal: ${total:.2f}")
    else:
        print("No items ordered.")
    print("-----------------")

# Run the billing system
generate_bill()
```

```
--- Hotel Menu ---
Room service: $50.00
Laundry: $25.00
Breakfast: $15.00
Dinner: $30.00
Massage: $75.00
------------------
Enter item name (or 'done' to finish): laundry
Enter quantity for Laundry: 5
Enter item name (or 'done' to finish):
```

Observation

- The code successfully defines functions for getting the menu, displaying it, getting the customer's order, calculating the bill, and generating the complete billing process.
- It uses a dictionary to store the menu items and their prices, which is a good way to manage this data.
- The get_customer_order function includes input validation to ensure the user enters valid menu items and non-negative quantities.
- The code was executed, and it started running the generate_bill() function, displaying the menu and prompting for user input ("Enter item name...").
- The execution was interrupted by a KeyboardInterrupt. This happened because the input() function was waiting for user input, and the execution was stopped manually (likely by clicking the stop button in the notebook). The code itself appears to be logically correct for a text-based system.

Task Description#5

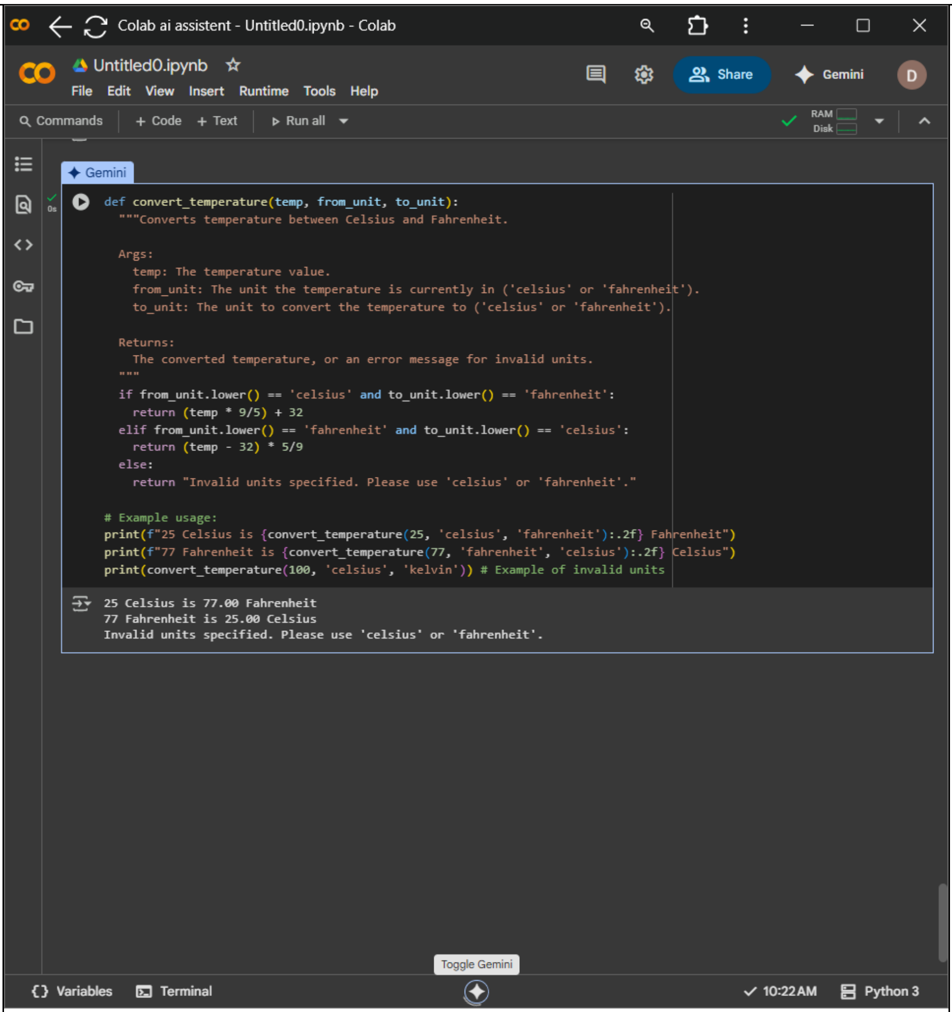
- Analyzing Prompt Specificity: Improving Temperature Conversion Function with Clear Instructions

Expected Output#5

- Code quality difference analysis for various prompts

Prompt

Write a python code to improving temperature conversion finction



```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.

Now, for Task Description #5, the key is to see how providing *more specific* instructions to the AI will influence the *next* version of this code. By comparing this current version with the one generated from a more specific prompt, we can analyze how prompt specificity impacts the resulting code's features and quality.

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**