**NAME : SAI KUSHAL**

**ROLL NO : 2403A51338**

**BATCH : 14**

# LAB ASSIGNMENT-4.1

**Task #1 – Zero-Shot Prompting with Conditional Validation**

Objective

Use zero-shot prompting to instruct an AI tool to generate a function that validates an Indian mobile number.

Requirements

* The function must ensure the mobile number:
  + Starts with 6, 7, 8, or 9
  + Contains exactly 10 digits

# PROGRAM:

# Screenshot 2025-08-13 113521OBSERVATION:

The program runs an infinite loop with while True:.

2. It asks the user to enter an Indian mobile number or type 'quit'.

3. The input is stored in number\_input.

4. If the user types 'quit' (case-insensitive), the loop ends with break.

5. Otherwise, the number is sent to validate\_indian\_mobile\_number(number\_input).

6. This function (defined elsewhere) checks if the input meets Indian mobile number rules.

7. Typical rules:

• Optional +91 or leading 0.

• Exactly 10 digits after removing prefix.

• First digit between 6 and 9.

• All characters are digits.

8. The function returns True if valid, otherwise False.

9. The result is printed in the format: '<number>' is valid: <True/False>.

10. Loop repeats until 'quit' is entered.

**Task #2 – One-Shot Prompting with Edge Case Handling**

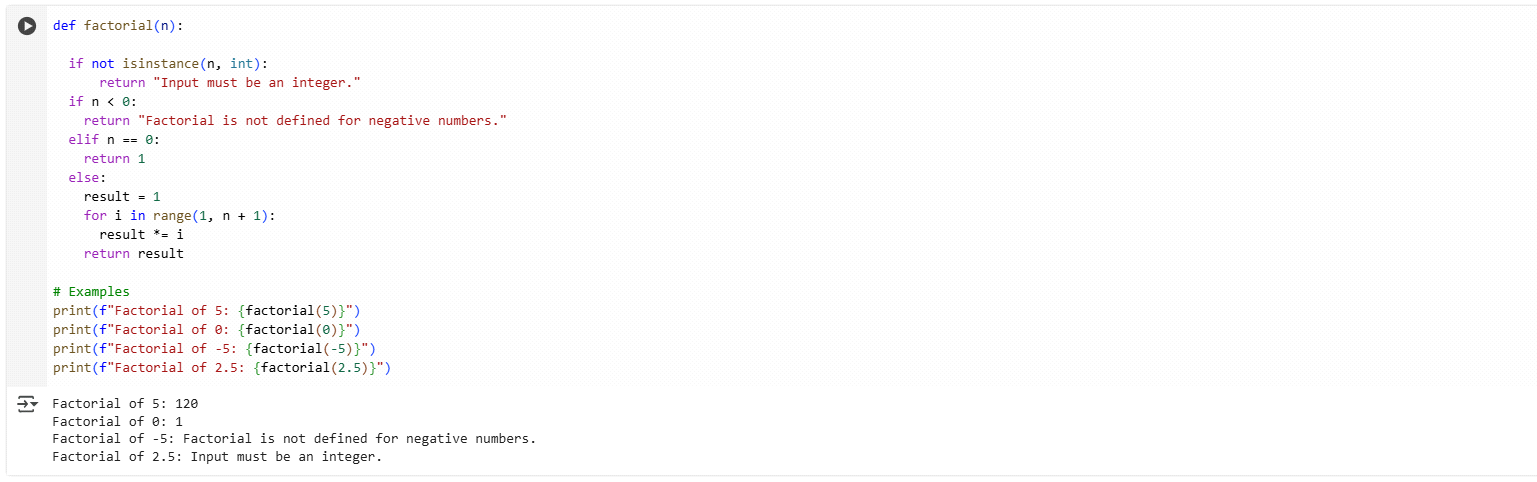
Objective

Use one-shot prompting to generate a Python function that calculates the factorial of a number.

Requirements

* Provide one sample input-output pair in the prompt to guide the AI.
* The function should handle:
  + 0! correctly
  + Negative input by returning an appropriate message

# PROGRAM:



# OBSERVATION:

1.calculate\_factorial(n) takes an integer n and explains its purpose in a docstring.

2. If n < 0, it immediately returns "Factorial is not defined for negative numbers".

3. If n == 0, it returns 1 (by definition, 0! = 1).

4. Otherwise it sets an accumulator: factorial = 1.

5. It loops i from 1 to n inclusive: for i in range(1, n+1).

6. Each iteration multiplies the accumulator: factorial \*= i.

7. After the loop, it returns the final factorial value.

8. calculate\_factorial(5) → 1×2×3×4×5 = 120 (printed).

9. calculate\_factorial(0) → 1; calculate\_factorial(-5) → the error message.

10. calculate\_factorial(3) → 1×2×3 = 6, showing the loop works for small n.

**Task #3 – Few-Shot Prompting for Nested Dictionary Extraction**

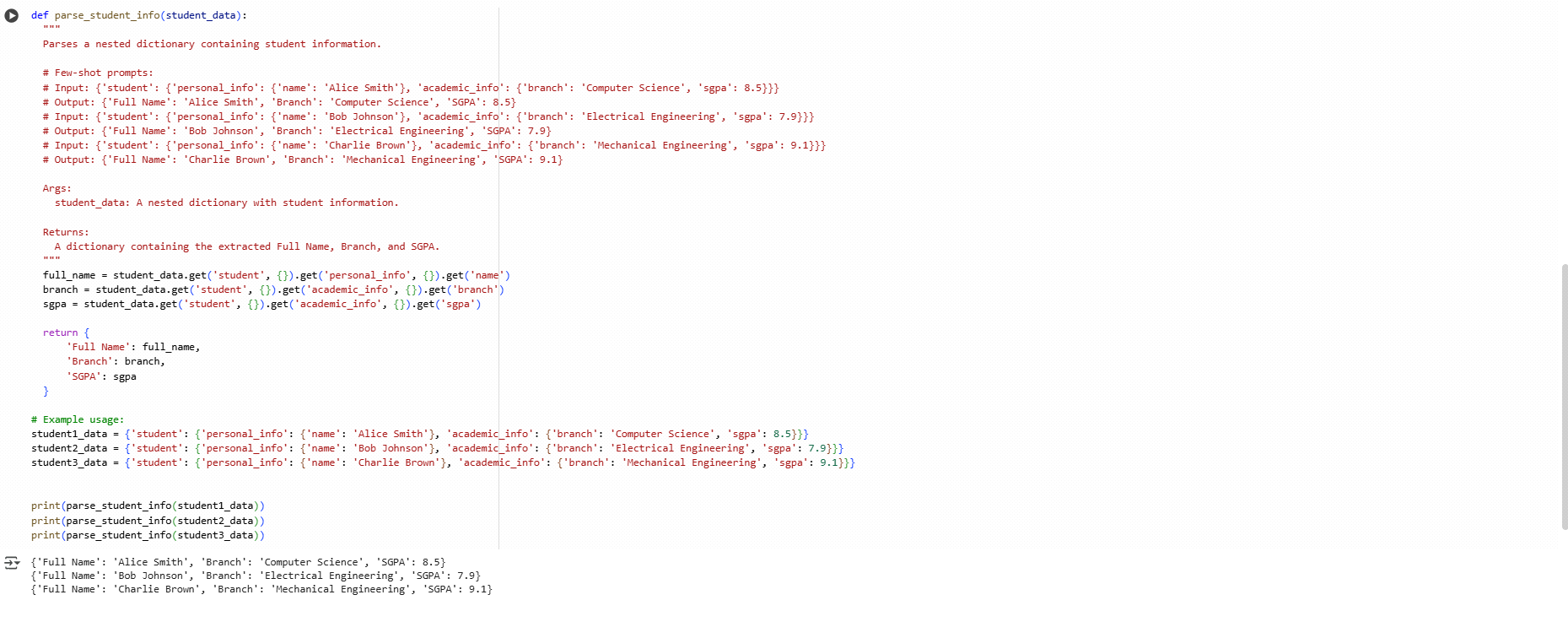
Objective

Use few-shot prompting (2–3 examples) to instruct the AI to create a function that parses a nested dictionary representing student information.

Requirements

* The function should extract and return:
  + Full Name
  + Branch
  + SGPA

# PROGRAM:



# OBSERVATION:

1.The function parse\_student\_info(student\_data) extracts key details from a nested student dictionary.

2. It expects the data under the top key 'student'.

3. From 'personal\_info', it gets the student’s 'name' and stores it in full\_name.

4. From 'academic\_info', it gets the 'branch'.

5. Also from 'academic\_info', it gets the 'sgpa'.

6. The .get() method with {} as a default avoids key errors if data is missing.

7. It returns a new dictionary with 'Full Name', 'Branch', and 'SGPA'.

8. Example dictionaries (student1\_data, student2\_data, student3\_data) are defined with nested personal and academic info.

9. The function is called for each student’s data.

10. The printed output shows extracted information in a clean format.

**Task #4 – Comparing Prompting Styles for File Analysis**

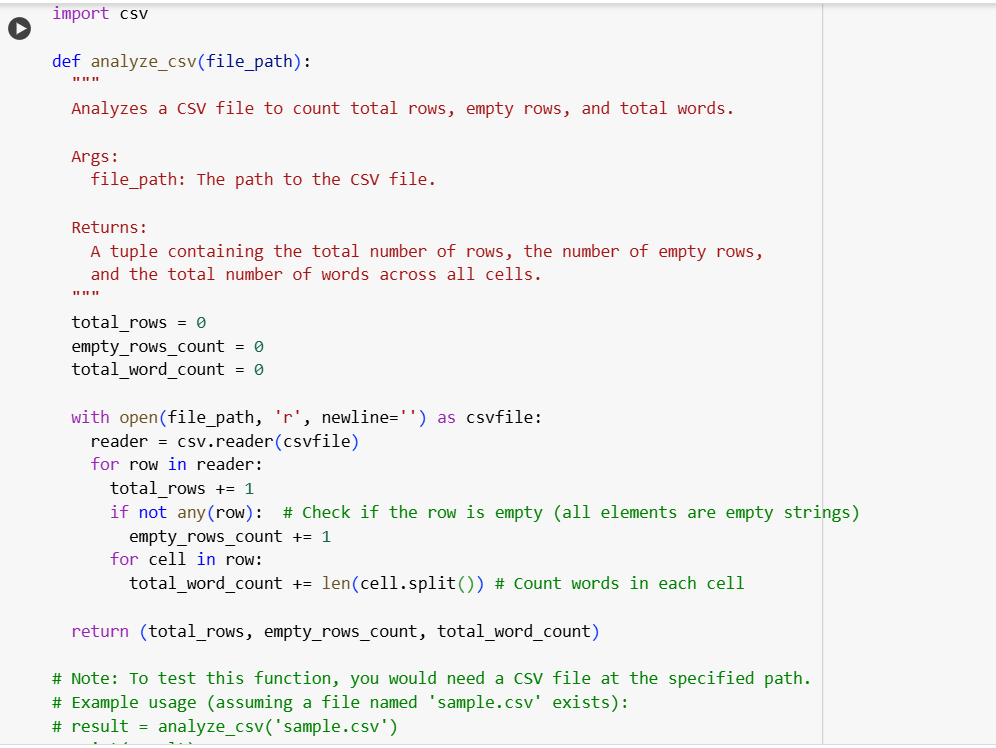
Objective

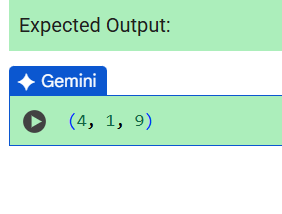
Experiment with zero-shot, one-shot, and few-shot prompting to generate functions for CSV file analysis.

Requirements

* Each generated function should:
  + Read a .csv file
  + Return the total number of rows
  + Count the number of empty rows
  + Count the number of words across the file

# PROGRAM:





# OBSERVATION:

1. analyze\_csv(file\_path) opens the CSV and sets total\_rows, empty\_rows\_count, total\_word\_count to 0.

2. It creates a csv.reader to iterate over rows.

3. For each row it increments total\_rows by 1.

4. if not any(row): checks if all cells are empty strings — if so it increments empty\_rows\_count.

5. It then iterates each cell in the row.

6. For each cell it does cell.split() and adds len(...) to total\_word\_count.

7. That counts whitespace-separated words across every cell.

8. After all rows are processed it returns (total\_rows, empty\_rows\_count, total\_word\_count).

9. Example result (4, 1, 9) means 4 rows total, 1 empty row, 9 words in the file.

10. The function requires a valid CSV file path to run.

**Task #5 – Few-Shot Prompting for Text Processing and WordFrequency**

Objective

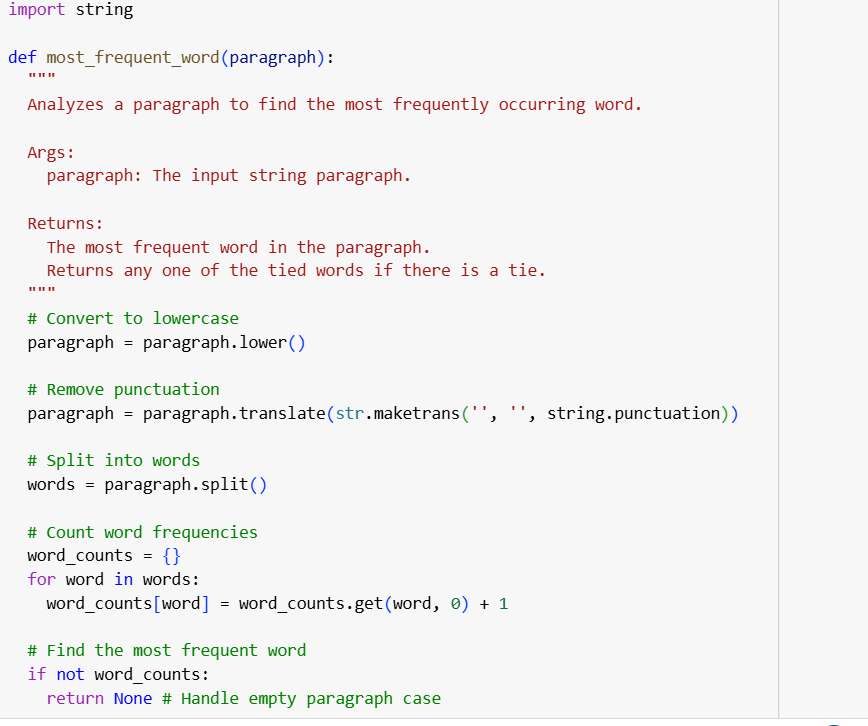
Use few-shot prompting (with at least 3 examples) to generate a Python function that processes text and analyzes word frequency.

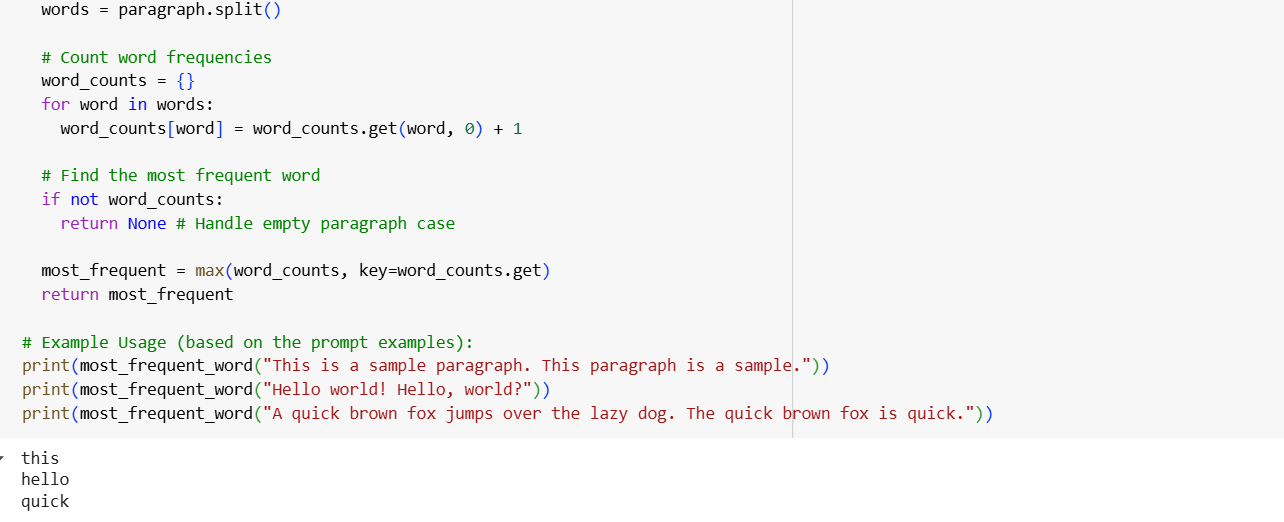
Requirements

The function must:

* Accept a paragraph as input
* Convert all text to lowercase
* Remove punctuation
* Return the most frequently used word

# PROGRAM:





# OBSERVATION:

1.most\_frequent\_word(paragraph) — function to find the word that appears most often in a string.

2. Convert the paragraph to lowercase (paragraph.lower()) so matching is case-insensitive.

3. Remove punctuation using paragraph.translate(str.maketrans('', '', string.punctuation)).

4. Split the cleaned string into words with words = paragraph.split().

5. Initialize an empty dict word\_counts = {}.

6. Loop over words and update counts: word\_counts[word] = word\_counts.get(word, 0) + 1.

7. If word\_counts is empty (no words), return None to handle the empty input case.

8. Find the most frequent word with max(word\_counts, key=word\_counts.get).

9. Return that word as the result (e.g., "this" for "This is a sample paragraph. This paragraph is a sample.").

10. Note: if multiple words tie for top count, the function returns one of them (depends on dictionary key order / first appearance).