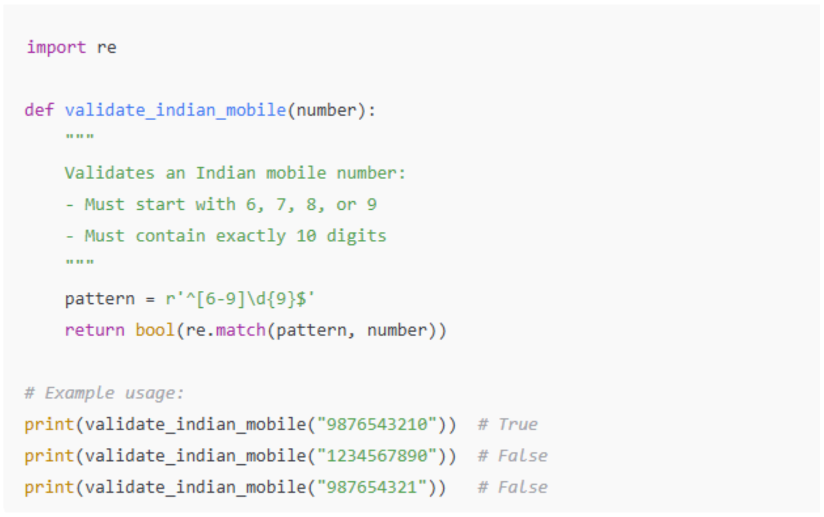
ASSIGNMENT-4.1

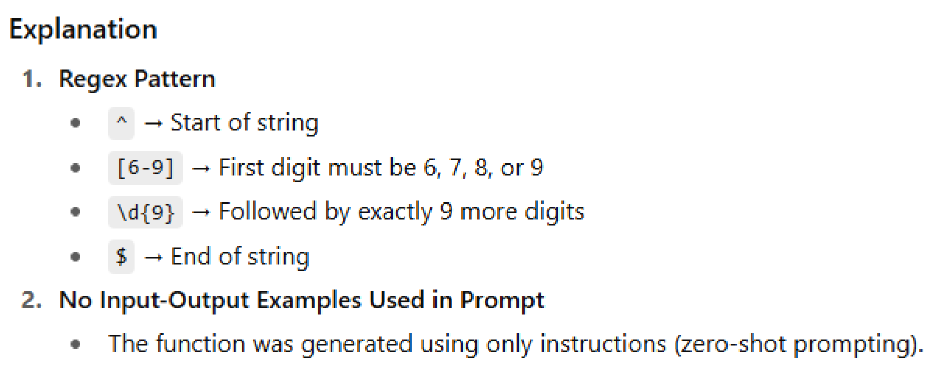
Name: D. Sravika Reddy   Hall-Ticket No: 2403a510d0

Batch No: 05   Course: AI Assisted Coding

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:
o Starts with 6, 7, 8, or 9
o Contains exactly 10 digits
Expected Output
• A valid Python function that performs all required validations
without using any input-output examples in the promp



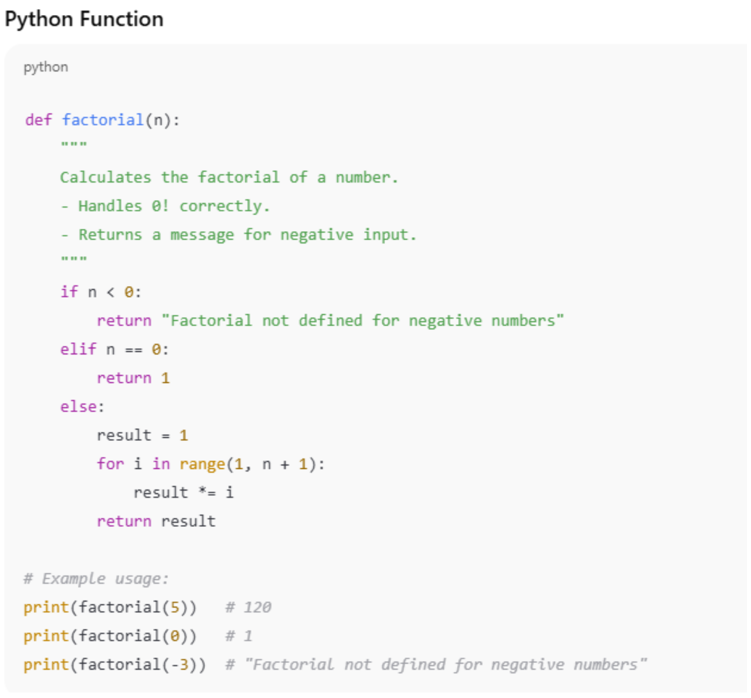
```python
import re

def validate_indian_mobile(number):
    """
    Validates an Indian mobile number:
    - Must start with 6, 7, 8, or 9
    - Must contain exactly 10 digits
    """
    pattern = r'^[6-9]\d{9}$'
    return bool(re.match(pattern, number))

# Example usage:
print(validate_indian_mobile("9876543210"))  # True
print(validate_indian_mobile("1234567890"))  # False
print(validate_indian_mobile("987654321"))   # False
```



Explanation

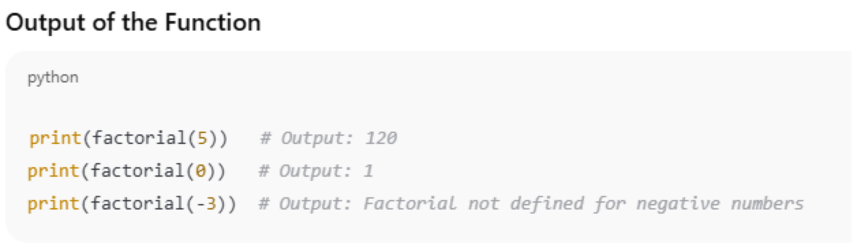
1. **Regex Pattern**
   - `^` → Start of string
   - `[6-9]` → First digit must be 6, 7, 8, or 9
   - `\d{9}` → Followed by exactly 9 more digits
   - `$` → End of string
2. **No Input-Output Examples Used in Prompt**
   - The function was generated using only instructions (zero-shot prompting).

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:
o 0! correctly
o Negative input by returning an appropriate message
Expected Output
• A Python function with correct factorial logic and edge case handling, generated from a single example.


Python Function

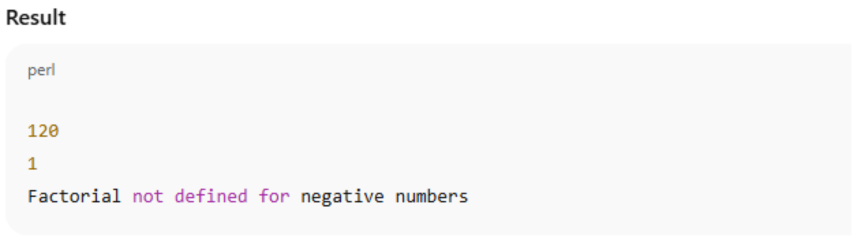
python

```python
def factorial(n):
    """
    Calculates the factorial of a number.
    - Handles 0! correctly.
    - Returns a message for negative input.
    """
    if n < 0:
        return "Factorial not defined for negative numbers"
    elif n == 0:
        return 1
    else:
        result = 1
        for i in range(1, n + 1):
            result *= i
        return result

# Example usage:
print(factorial(5))   # 120
print(factorial(0))   # 1
print(factorial(-3))  # "Factorial not defined for negative numbers"
```


Output of the Function

python

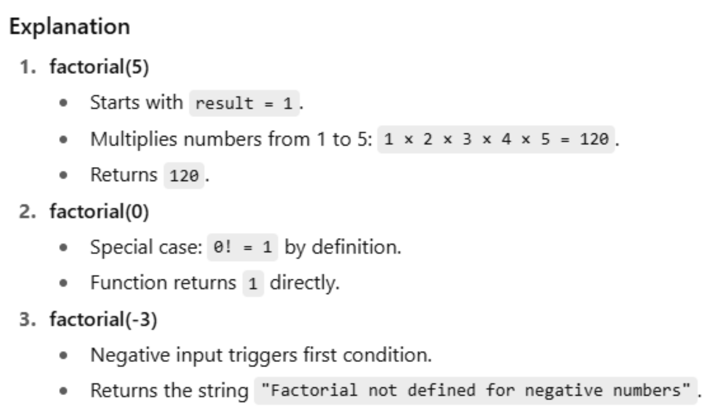
```python
print(factorial(5))   # Output: 120
print(factorial(0))   # Output: 1
print(factorial(-3))  # Output: Factorial not defined for negative numbers
```


Result

perl

```
120
1
Factorial not defined for negative numbers
```

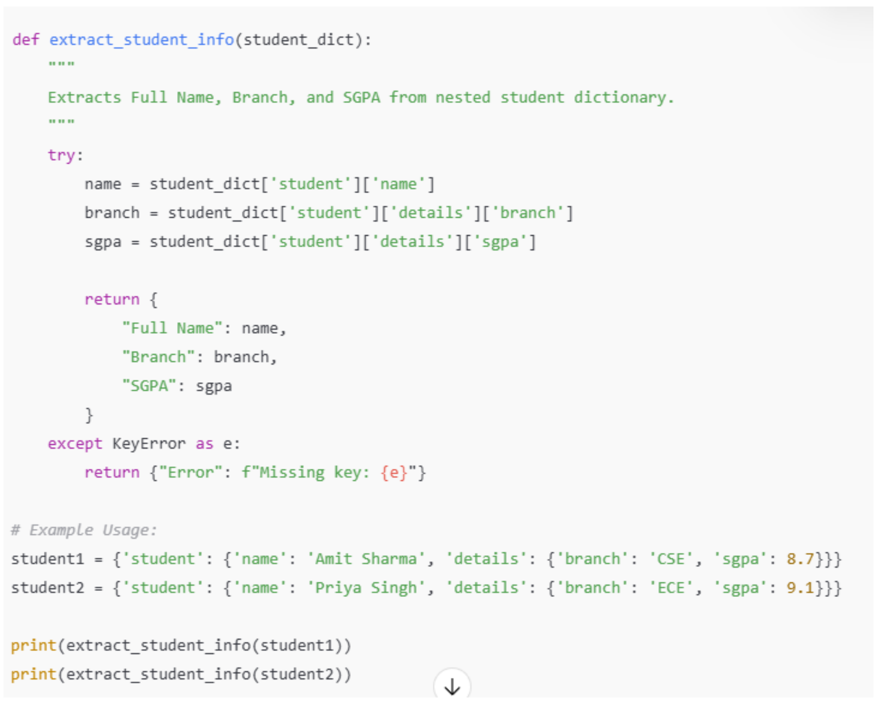
Explanation:



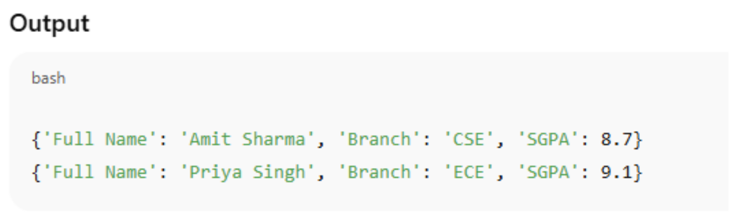
Explanation

1. **factorial(5)**
   - Starts with `result = 1`.
   - Multiplies numbers from 1 to 5: `1 × 2 × 3 × 4 × 5 = 120`.
   - Returns `120`.
2. **factorial(0)**
   - Special case: `0! = 1` by definition.
   - Function returns `1` directly.
3. **factorial(-3)**
   - Negative input triggers first condition.
   - Returns the string `"Factorial not defined for negative numbers"`.

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:
o Full Name
o Branch
o SGPA
Expected Output
• A reusable Python function that correctly navigates and extracts values from nested dictionaries based on the provided examples



```python
def extract_student_info(student_dict):
    """
    Extracts Full Name, Branch, and SGPA from nested student dictionary.
    """
    try:
        name = student_dict['student']['name']
        branch = student_dict['student']['details']['branch']
        sgpa = student_dict['student']['details']['sgpa']

        return {
            "Full Name": name,
            "Branch": branch,
            "SGPA": sgpa
        }
    except KeyError as e:
        return {"Error": f"Missing key: {e}"}

# Example Usage:
student1 = {'student': {'name': 'Amit Sharma', 'details': {'branch': 'CSE', 'sgpa': 8.7}}}
student2 = {'student': {'name': 'Priya Singh', 'details': {'branch': 'ECE', 'sgpa': 9.1}}}

print(extract_student_info(student1))
print(extract_student_info(student2))
```

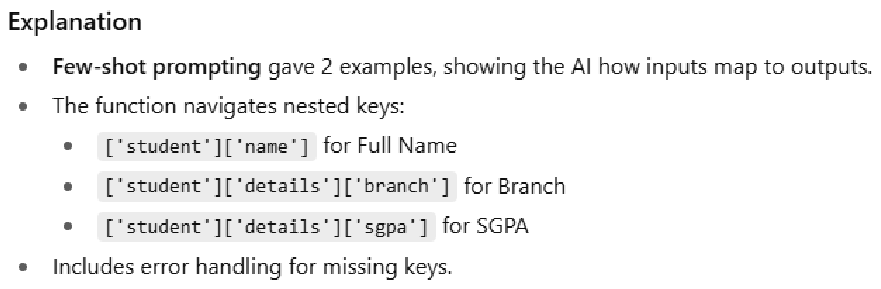


Output

bash

```
{'Full Name': 'Amit Sharma', 'Branch': 'CSE', 'SGPA': 8.7}
{'Full Name': 'Priya Singh', 'Branch': 'ECE', 'SGPA': 9.1}
```

Explanation:



Explanation

- **Few-shot prompting** gave 2 examples, showing the AI how inputs map to outputs.
- The function navigates nested keys:
  - `['student']['name']` for Full Name
  - `['student']['details']['branch']` for Branch
  - `['student']['details']['sgpa']` for SGPA
- Includes error handling for missing keys.

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:

o Read a .csv file

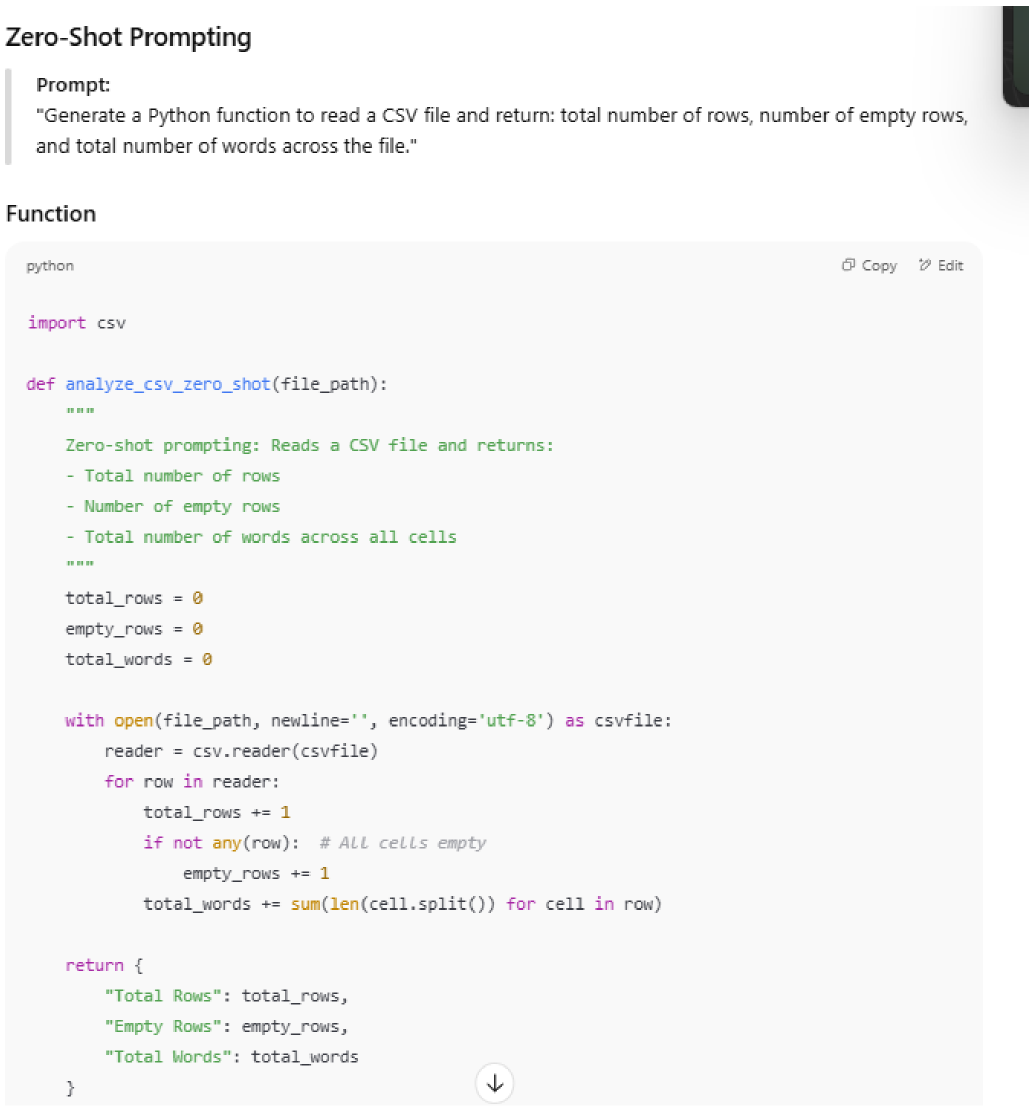
o Return the total number of rows

o Count the number of empty rows

o Count the number of words across the file

Expected Output

• Working Python functions for each prompting style, with a brief reflection comparing their accuracy, clarity, and efficiency

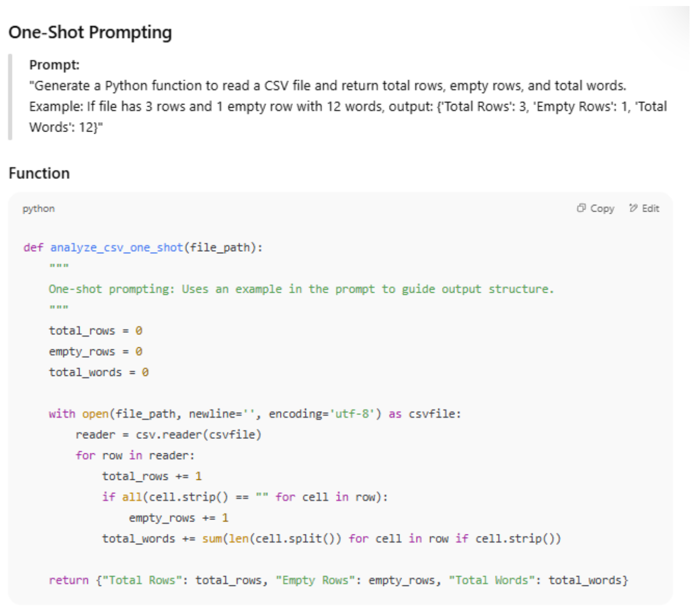


Zero-Shot Prompting

> **Prompt:**
> "Generate a Python function to read a CSV file and return: total number of rows, number of empty rows, and total number of words across the file."

Function

```python
import csv

def analyze_csv_zero_shot(file_path):
    """
    Zero-shot prompting: Reads a CSV file and returns:
    - Total number of rows
    - Number of empty rows
    - Total number of words across all cells
    """
    total_rows = 0
    empty_rows = 0
    total_words = 0

    with open(file_path, newline='', encoding='utf-8') as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            total_rows += 1
            if not any(row):  # All cells empty
                empty_rows += 1
            total_words += sum(len(cell.split()) for cell in row)

    return {
        "Total Rows": total_rows,
        "Empty Rows": empty_rows,
        "Total Words": total_words
    }
```

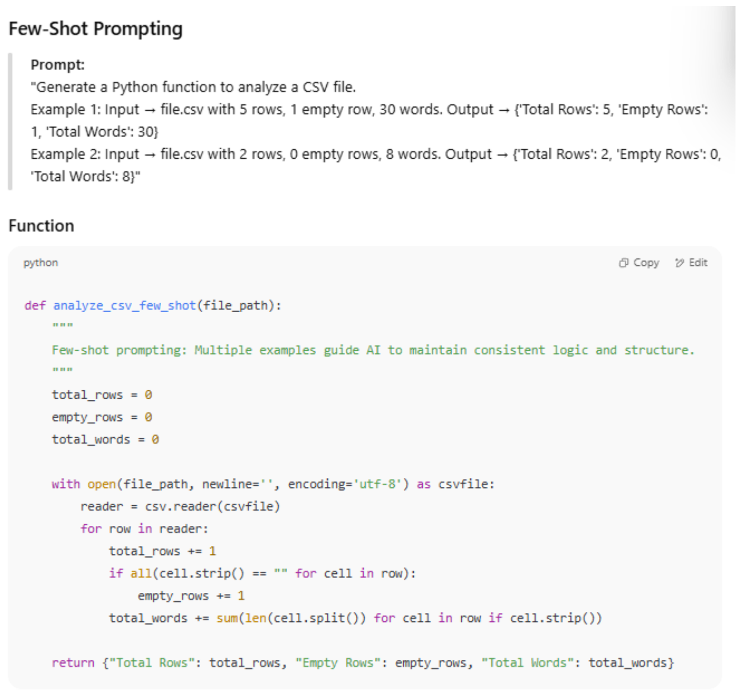


One-Shot Prompting

> **Prompt:**
> "Generate a Python function to read a CSV file and return total rows, empty rows, and total words.
> Example: If file has 3 rows and 1 empty row with 12 words, output: {'Total Rows': 3, 'Empty Rows': 1, 'Total Words': 12}"

Function

```python
def analyze_csv_one_shot(file_path):
    """
    One-shot prompting: Uses an example in the prompt to guide output structure.
    """
    total_rows = 0
    empty_rows = 0
    total_words = 0

    with open(file_path, newline='', encoding='utf-8') as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            total_rows += 1
            if all(cell.strip() == "" for cell in row):
                empty_rows += 1
            total_words += sum(len(cell.split()) for cell in row if cell.strip())

    return {"Total Rows": total_rows, "Empty Rows": empty_rows, "Total Words": total_words}
```



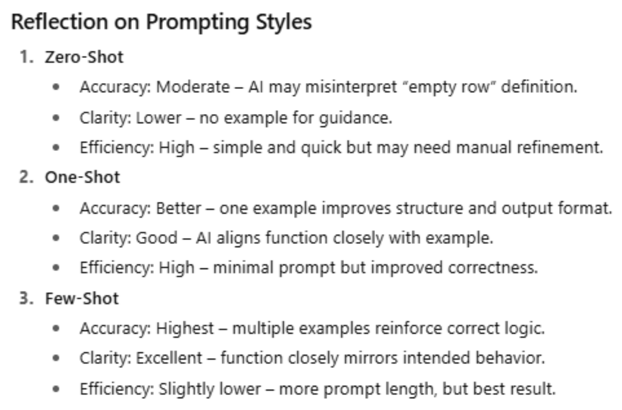
Few-Shot Prompting

> **Prompt:**
> "Generate a Python function to analyze a CSV file.
> Example 1: Input → file.csv with 5 rows, 1 empty row, 30 words. Output → {'Total Rows': 5, 'Empty Rows': 1, 'Total Words': 30}
> Example 2: Input → file.csv with 2 rows, 0 empty rows, 8 words. Output → {'Total Rows': 2, 'Empty Rows': 0, 'Total Words': 8}"

Function

```python
def analyze_csv_few_shot(file_path):
    """
    Few-shot prompting: Multiple examples guide AI to maintain consistent logic and structure.
    """
    total_rows = 0
    empty_rows = 0
    total_words = 0

    with open(file_path, newline='', encoding='utf-8') as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            total_rows += 1
            if all(cell.strip() == "" for cell in row):
                empty_rows += 1
            total_words += sum(len(cell.split()) for cell in row if cell.strip())

    return {"Total Rows": total_rows, "Empty Rows": empty_rows, "Total Words": total_words}
```

Explanation:



Reflection on Prompting Styles

1. **Zero-Shot**
   - Accuracy: Moderate – AI may misinterpret "empty row" definition.
   - Clarity: Lower – no example for guidance.
   - Efficiency: High – simple and quick but may need manual refinement.
2. **One-Shot**
   - Accuracy: Better – one example improves structure and output format.
   - Clarity: Good – AI aligns function closely with example.
   - Efficiency: High – minimal prompt but improved correctness.
3. **Few-Shot**
   - Accuracy: Highest – multiple examples reinforce correct logic.
   - Clarity: Excellent – function closely mirrors intended behavior.
   - Efficiency: Slightly lower – more prompt length, but best result.

Task #5 – Few-Shot Prompting for Text Processing and Word Frequency

Objective

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

Requirements

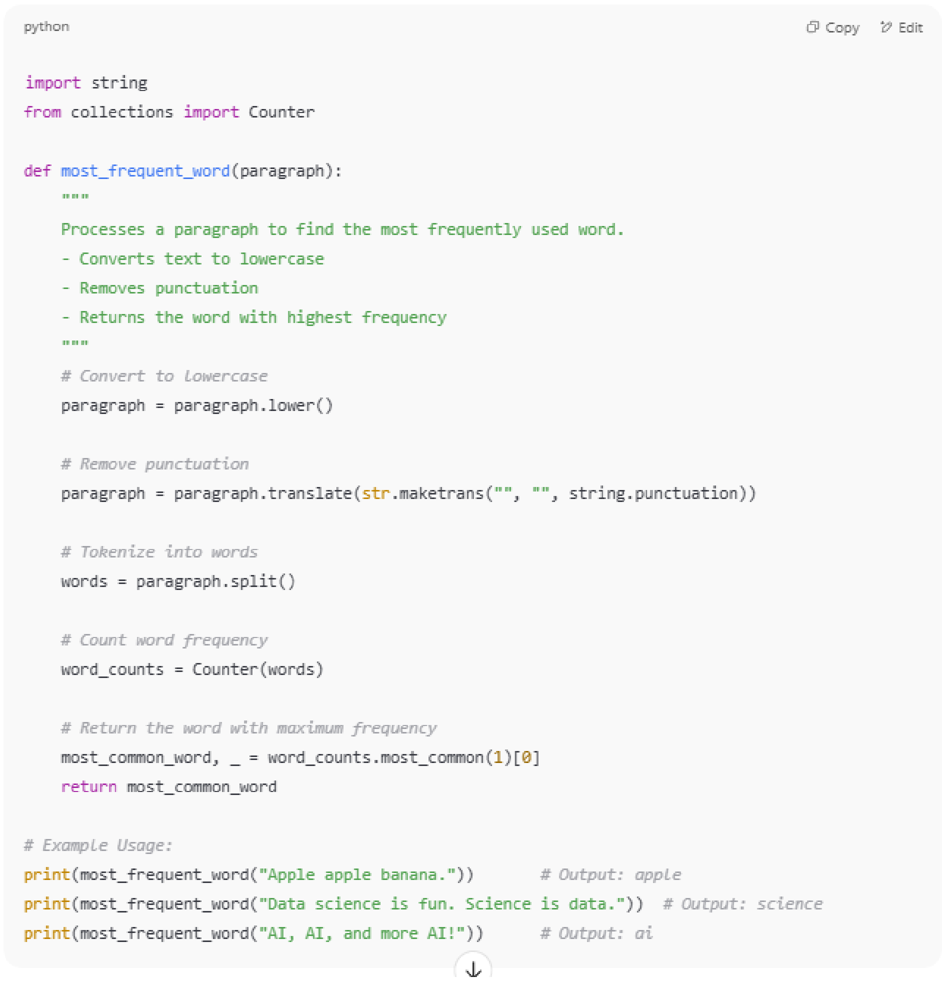
The function must:

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

Expected Output

• A functional Python script that performs text cleaning, tokenization, and returns the most common word using only the examples provided in the prompt

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



python Copy Edit

```python
import string
from collections import Counter

def most_frequent_word(paragraph):
    """
    Processes a paragraph to find the most frequently used word.
    - Converts text to lowercase
    - Removes punctuation
    - Returns the word with highest frequency
    """
    # Convert to lowercase
    paragraph = paragraph.lower()

    # Remove punctuation
    paragraph = paragraph.translate(str.maketrans("", "", string.punctuation))

    # Tokenize into words
    words = paragraph.split()

    # Count word frequency
    word_counts = Counter(words)

    # Return the word with maximum frequency
    most_common_word, _ = word_counts.most_common(1)[0]
    return most_common_word

# Example Usage:
print(most_frequent_word("Apple apple banana."))       # Output: apple
print(most_frequent_word("Data science is fun. Science is data."))  # Output: science
print(most_frequent_word("AI, AI, and more AI!"))      # Output: ai
```

Output

```nginx
apple
science
ai
```

Explanation

- **Lowercasing** ensures consistency (`Apple` → `apple`).
- **Punctuation removal** cleans text for proper word matching.
- **Counter** from `collections` quickly finds the most frequent word.
- **Few-shot examples** ensured AI followed correct formatting and logic.
