SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Program Name: B. Tech	**Assignment Type: Lab**	**Academic Year:**2025-2026

Course Coordinator Name	Venkataraman Veeramsetty
Instructor(s) Name	Dr. V. Venkataramana (Co-Ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)

Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week2 - Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	

Assignment Number:4.1(Present assignment number)/**24**(Total number of assignments)

Q.No.	Question	*Expected Time to complete*
1	Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques **Lab Objectives:** • To explore and apply different levels of prompt examples in AI-assisted code generation. • To understand how zero-shot, one-shot, and few-shot	Week2 - Monday

prompting affect AI output quality.
- To evaluate the impact of context richness and example quantity on AI performance.
- To build awareness of prompt strategy effectiveness for different problem types.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Use zero-shot prompting to instruct AI with minimal context.
- Use one-shot prompting with a single example to guide AI code generation.
- Apply few-shot prompting using multiple examples to improve AI responses.
- Compare AI outputs across the three prompting strategies.

Task #1 – Zero-Shot Prompting with Conditional Validation

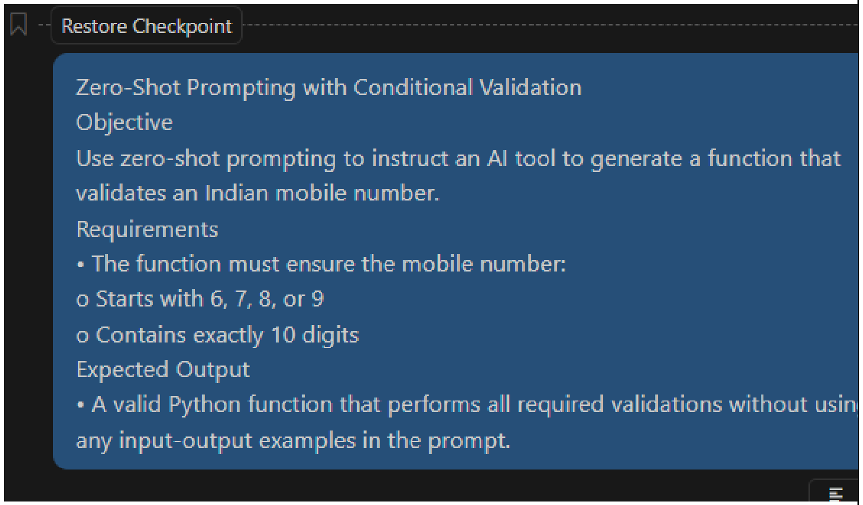
Objective

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

Requirements

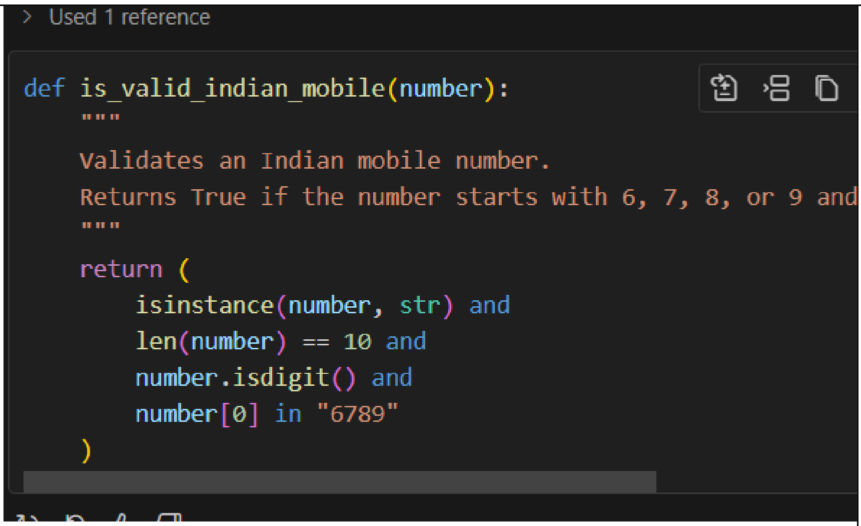
- The function must ensure the mobile number:
  - Starts with 6, 7, 8, or 9
  - Contains exactly 10 digits

Prompt:



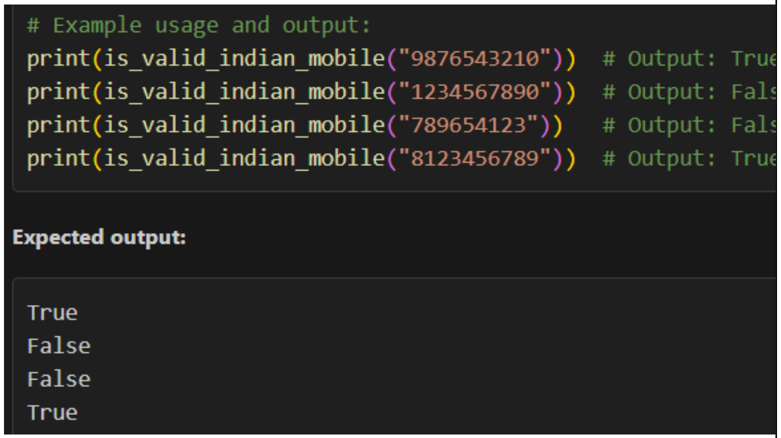
Zero-Shot Prompting with Conditional Validation
Objective
Use zero-shot prompting to instruct an AI tool to generate a function that validates an Indian mobile number.
Requirements
• The function must ensure the mobile number:
o Starts with 6, 7, 8, or 9
o Contains exactly 10 digits
Expected Output
• A valid Python function that performs all required validations without usin
any input-output examples in the prompt.

Code:



```
> Used 1 reference

def is_valid_indian_mobile(number):
    """
    Validates an Indian mobile number.
    Returns True if the number starts with 6, 7, 8, or 9 and
    """
    return (
        isinstance(number, str) and
        len(number) == 10 and
        number.isdigit() and
        number[0] in "6789"
    )
```

Expected Output

- A valid Python function that performs all required validations without using any input-output examples in the prompt.
- 

```
# Example usage and output:
print(is_valid_indian_mobile("9876543210"))  # Output: True
print(is_valid_indian_mobile("1234567890"))  # Output: Fals
print(is_valid_indian_mobile("789654123"))   # Output: Fals
print(is_valid_indian_mobile("8123456789"))  # Output: True
```

Expected output:

```
True
False
False
True
```

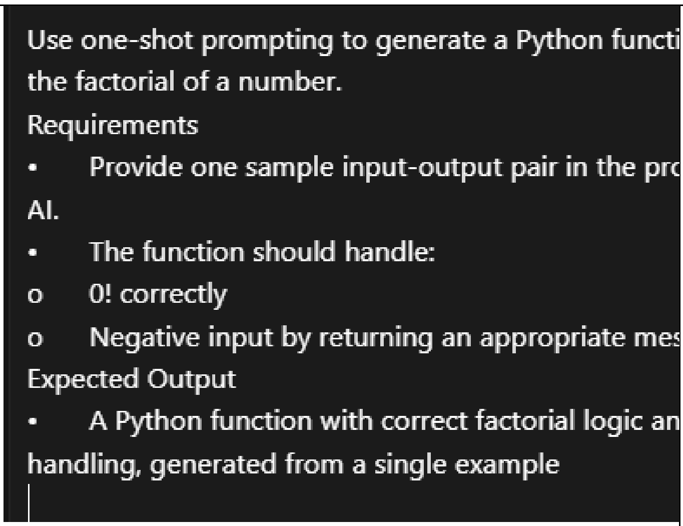
Task #2 – One-Shot Prompting with Edge Case Handling

Objective

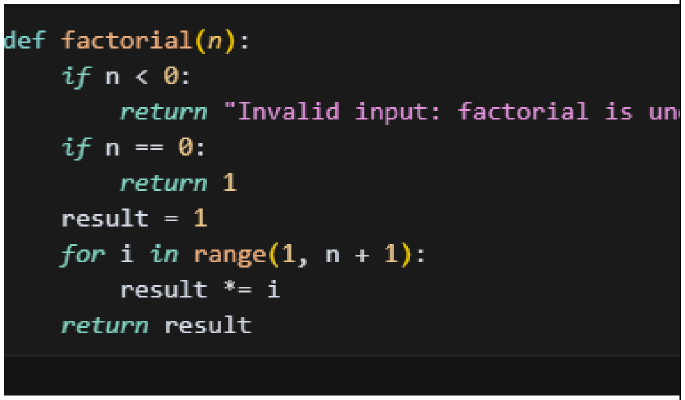
Use one-shot prompting to generate a Python function that calculates the factorial of a number.

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
  - Prompt:

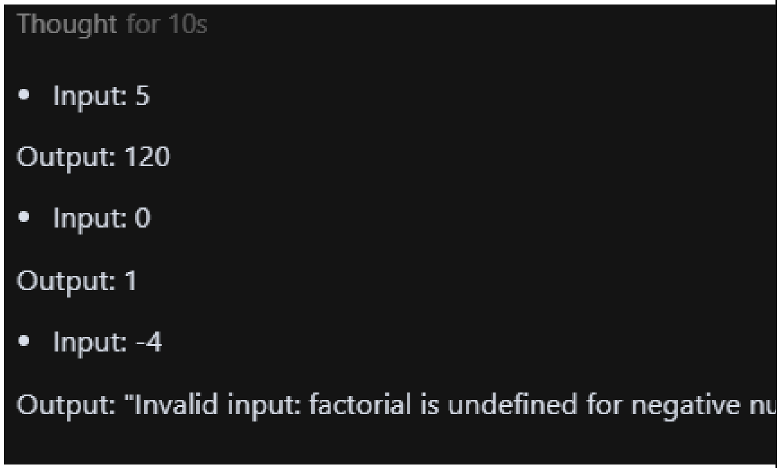

Use one-shot prompting to generate a Python functi
the factorial of a number.
Requirements
•	Provide one sample input-output pair in the pro
AI.
•	The function should handle:
o	0! correctly
o	Negative input by returning an appropriate mes
Expected Output
•	A Python function with correct factorial logic an
handling, generated from a single example

- Code:


```
def factorial(n):
    if n < 0:
        return "Invalid input: factorial is und
    if n == 0:
        return 1
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

Expected Output

- A Python function with correct factorial logic and edge case handling, generated from a single example.


Thought for 10s

- Input: 5

Output: 120

- Input: 0

Output: 1

- Input: -4

Output: "Invalid input: factorial is undefined for negative nu

-

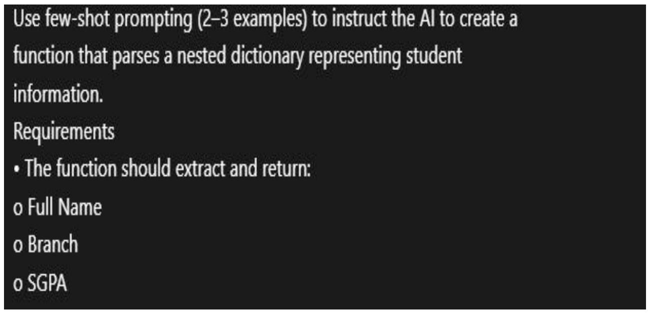
Task #3 – Few-Shot Prompting for Nested Dictionary Extraction

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
  - Prompt:
  -



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

Requirements

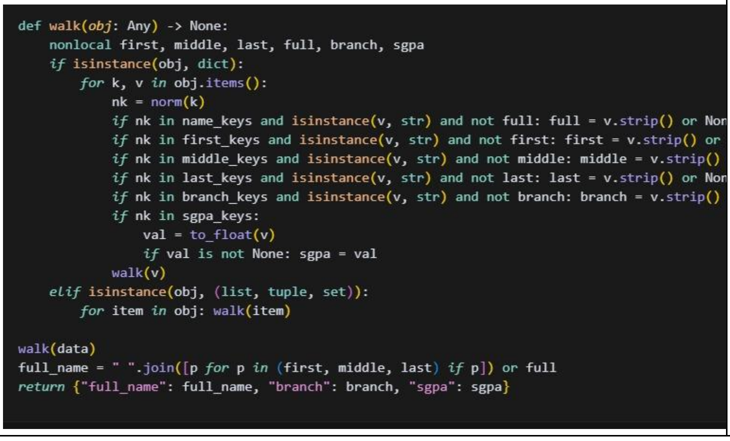
• The function should extract and return:

o Full Name

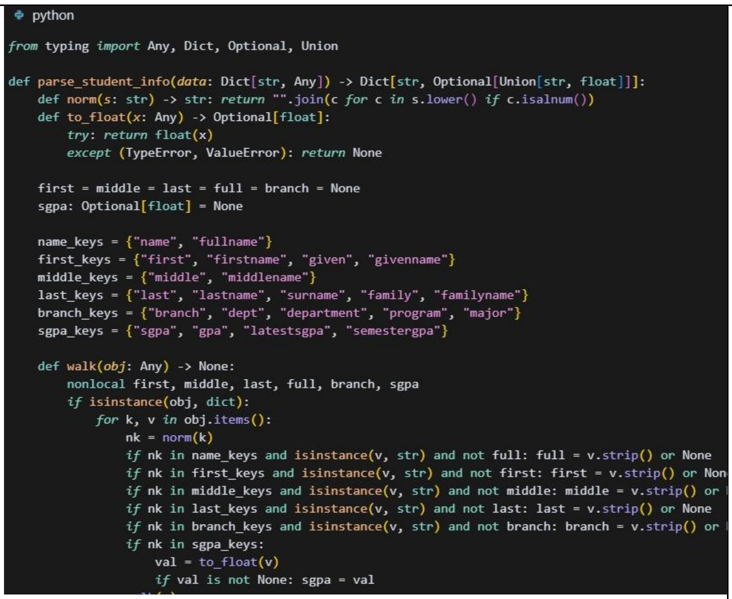
o Branch

o SGPA

Code:



```python
def walk(obj: Any) -> None:
    nonlocal first, middle, last, full, branch, sgpa
    if isinstance(obj, dict):
        for k, v in obj.items():
            nk = norm(k)
            if nk in name_keys and isinstance(v, str) and not full: full = v.strip() or Non
            if nk in first_keys and isinstance(v, str) and not first: first = v.strip() or
            if nk in middle_keys and isinstance(v, str) and not middle: middle = v.strip()
            if nk in last_keys and isinstance(v, str) and not last: last = v.strip() or Non
            if nk in branch_keys and isinstance(v, str) and not branch: branch = v.strip()
            if nk in sgpa_keys:
                val = to_float(v)
                if val is not None: sgpa = val
            walk(v)
    elif isinstance(obj, (list, tuple, set)):
        for item in obj: walk(item)

walk(data)
full_name = " ".join([p for p in (first, middle, last) if p]) or full
return {"full_name": full_name, "branch": branch, "sgpa": sgpa}
```

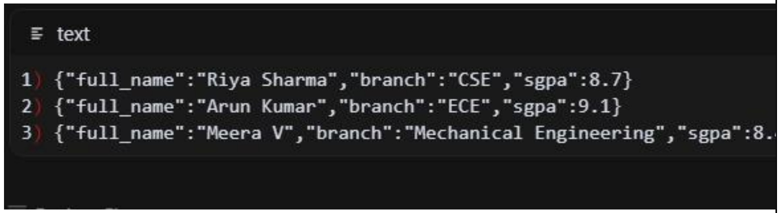


python

```python
from typing import Any, Dict, Optional, Union

def parse_student_info(data: Dict[str, Any]) -> Dict[str, Optional[Union[str, float]]]:
    def norm(s: str) -> str: return "".join(c for c in s.lower() if c.isalnum())
    def to_float(x: Any) -> Optional[float]:
        try: return float(x)
        except (TypeError, ValueError): return None

    first = middle = last = full = branch = None
    sgpa: Optional[float] = None

    name_keys = {"name", "fullname"}
    first_keys = {"first", "firstname", "given", "givenname"}
    middle_keys = {"middle", "middlename"}
    last_keys = {"last", "lastname", "surname", "family", "familyname"}
    branch_keys = {"branch", "dept", "department", "program", "major"}
    sgpa_keys = {"sgpa", "gpa", "latestsgpa", "semestergpa"}

    def walk(obj: Any) -> None:
        nonlocal first, middle, last, full, branch, sgpa
        if isinstance(obj, dict):
            for k, v in obj.items():
                nk = norm(k)
                if nk in name_keys and isinstance(v, str) and not full: full = v.strip() or None
                if nk in first_keys and isinstance(v, str) and not first: first = v.strip() or Non
                if nk in middle_keys and isinstance(v, str) and not middle: middle = v.strip() or
                if nk in last_keys and isinstance(v, str) and not last: last = v.strip() or None
                if nk in branch_keys and isinstance(v, str) and not branch: branch = v.strip() or
                if nk in sgpa_keys:
                    val = to_float(v)
                    if val is not None: sgpa = val
```

Expected Output

- A reusable Python function that correctly navigates and extracts values from nested dictionaries based on the provided examples.



text

```text
1) {"full_name":"Riya Sharma","branch":"CSE","sgpa":8.7}
2) {"full_name":"Arun Kumar","branch":"ECE","sgpa":9.1}
3) {"full_name":"Meera V","branch":"Mechanical Engineering","sgpa":8.
```

-

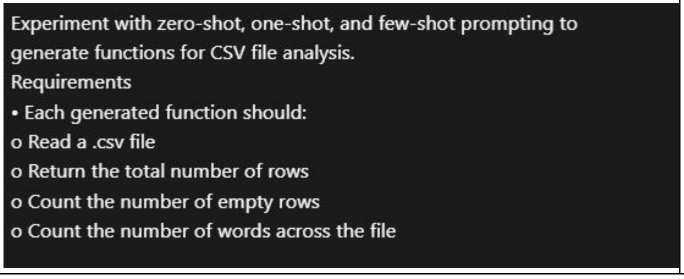
Task #4 – Comparing Prompting Styles for File Analysis

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
  - Prompt:

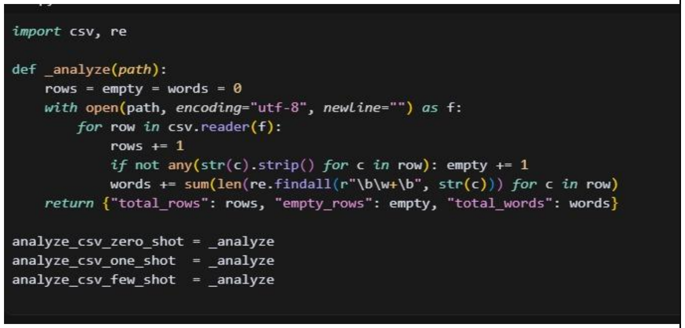


Experiment with zero-shot, one-shot, and few-shot prompting to generate functions for CSV file analysis.
Requirements
• Each generated function should:
o Read a .csv file
o Return the total number of rows
o Count the number of empty rows
o Count the number of words across the file

  -

Count th e number of words across the file

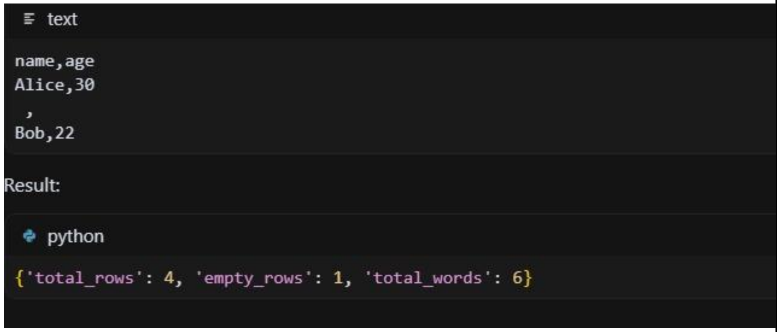
- Code:



```python
import csv, re

def _analyze(path):
    rows = empty = words = 0
    with open(path, encoding="utf-8", newline="") as f:
        for row in csv.reader(f):
            rows += 1
            if not any(str(c).strip() for c in row): empty += 1
            words += sum(len(re.findall(r"\b\w+\b", str(c))) for c in row)
    return {"total_rows": rows, "empty_rows": empty, "total_words": words}

analyze_csv_zero_shot = _analyze
analyze_csv_one_shot  = _analyze
analyze_csv_few_shot  = _analyze
```

Expected Output

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



text

```text
name,age
Alice,30
 ,
Bob,22
```

Result:

python

```python
{'total_rows': 4, 'empty_rows': 1, 'total_words': 6}
```

-

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

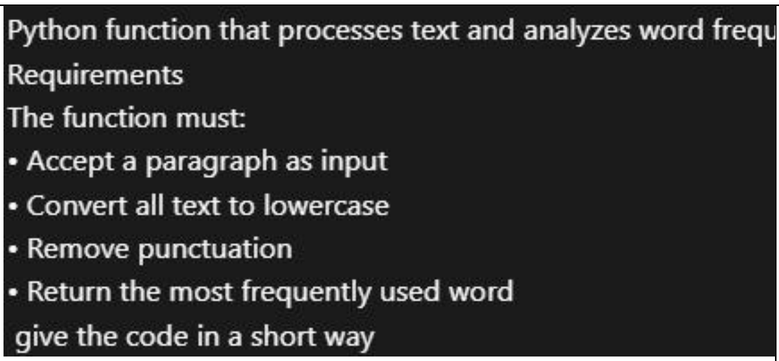
Objective

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

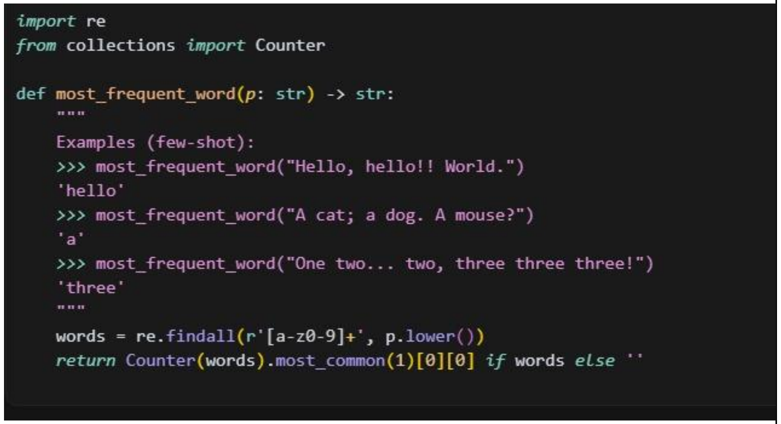
Requirements

The function must:

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

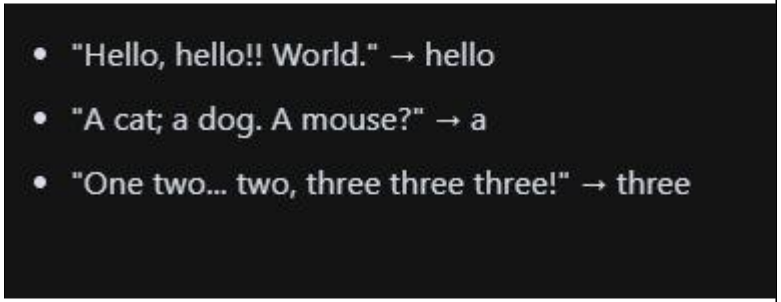

Python function that processes text and analyzes word frequ
Requirements
The function must:
• Accept a paragraph as input
• Convert all text to lowercase
• Remove punctuation
• Return the most frequently used word
give the code in a short way

-
-
- code:



```python
import re
from collections import Counter

def most_frequent_word(p: str) -> str:
    """
    Examples (few-shot):
    >>> most_frequent_word("Hello, hello!! World.")
    'hello'
    >>> most_frequent_word("A cat; a dog. A mouse?")
    'a'
    >>> most_frequent_word("One two... two, three three three!")
    'three'
    """
    words = re.findall(r'[a-z0-9]+', p.lower())
    return Counter(words).most_common(1)[0][0] if words else ''
```

-
-
- 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



- "Hello, hello!! World." → hello
- "A cat; a dog. A mouse?" → a
- "One two... two, three three three!" → three

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
Zero Shot (Task #1)	0.5

One Shot (Task#2)	0.5
Few Shot (Task#3, Task#4 & Task #5)	1.5
Total	**2.5 Marks**