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	Venkataramana 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.	Week2 - Monday

- To understand how zero-shot, one-shot, and few-shot 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.

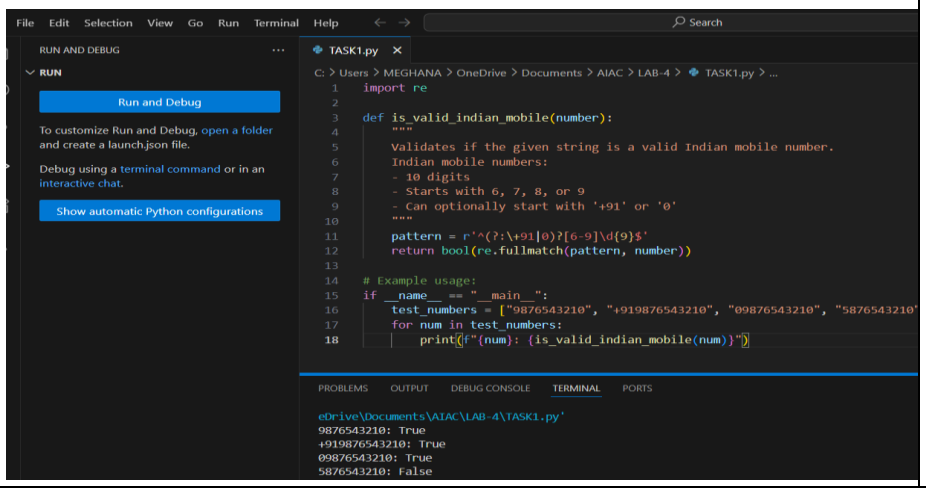
Requirements

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

Expected Output

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

PROMPT: Write a python code to generate a python function that validates an Indian mobile number



Code explanation:

This code checks if a given input is a valid Indian mobile number.
Here's how it works:

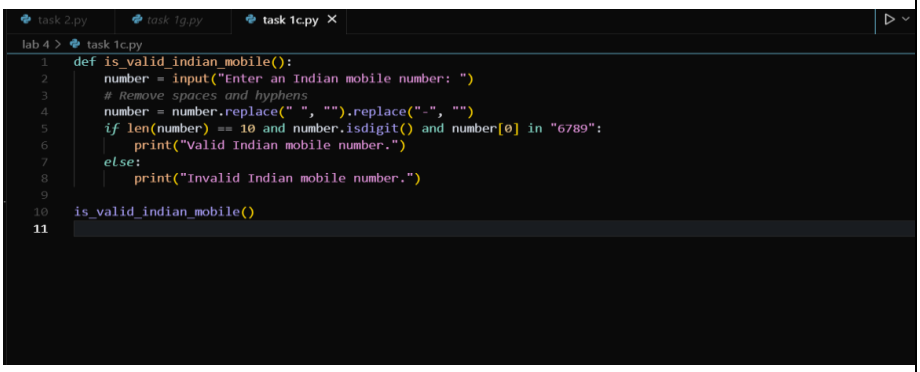
- It imports the re module for regular expressions.
- The function is_valid_indian_mobile(number):
  - Removes spaces and hyphens from the input.
  - Uses a regex pattern to check if the number has exactly 10 digits and starts with a digit from 6 to 9 (the valid starting digits for Indian mobile numbers).
  - Returns True if the number matches the pattern, otherwise False.
- The script asks the user to enter a mobile number, checks its validity using the function, and prints whether it is valid or not.

In cursor:

Prompt:

write a function in python which checks if indian mobile number is valid or not and the function must ensure the mobile number Starts with 6, 7, 8, or 9 and Contains exactly 10 digits and takes input and prints output

code:



Output:

```
Enter an Indian mobile number: 9874445552
Valid Indian mobile number.
```

Code explanation:

The function is_valid_indian_mobile() checks if a user-input mobile number follows the standard Indian mobile number format.Removes any spaces and hyphens from the input to handle formatted numbers like "98765-43210" or "987 654 3210".

The number is valid if it meets ALL three conditions:

- **Length**: Exactly 10 digits
- **Digits only**: Contains only numeric characters (0-9)
- **Starting digit**: First digit must be 6, 7, 8, or 9

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

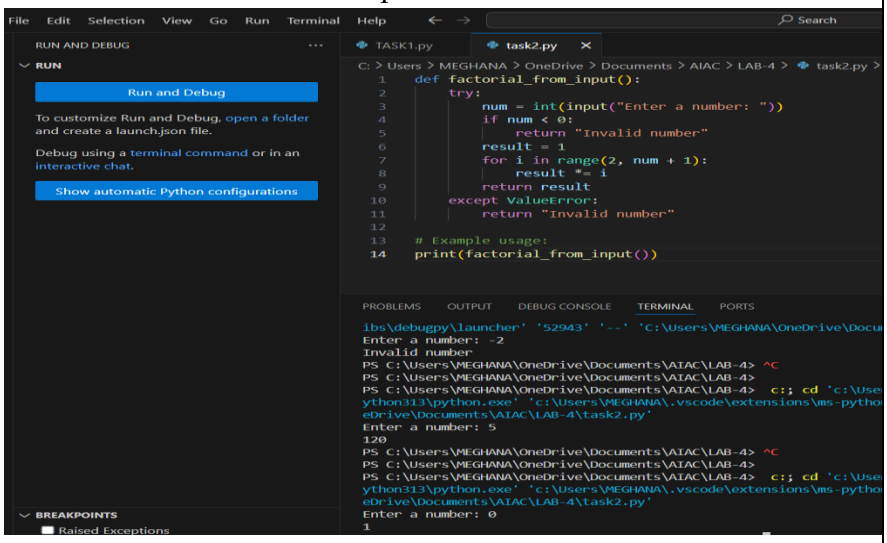
- Provide one sample input-output pair in the prompt to guide the AI.
- The function should handle:
  - 0! correctly
  - Negative input by returning an appropriate message

Expected Output

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

PROMPT: Write a python function that calculates the factorial of a number that takes input from user and return negative input by invalid number example: Input: Enter a non-negative integer:5

Output:120



Code explanation:

This code calculates the factorial of a given number entered by the user. Here's how it works:

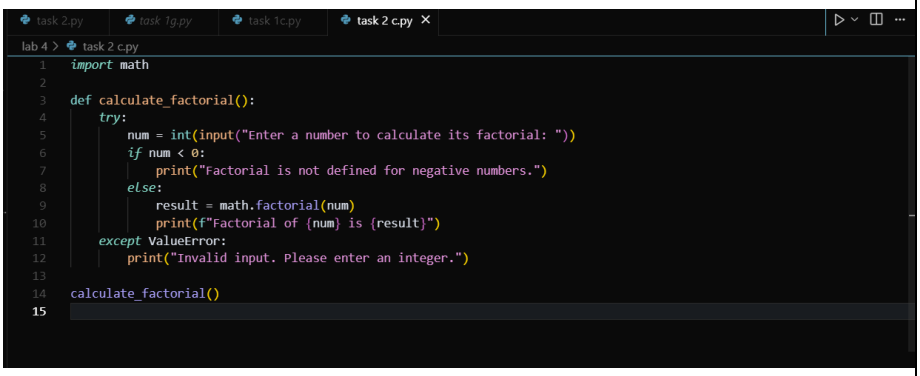
- The factorial(n) function:
  - Returns an error message if the input is negative (since factorial is not defined for negative numbers).
  - Returns 1 if the input is 0 (since 0! = 1).
  - Otherwise, it calculates the factorial by multiplying all integers from 1 to n.

- The script asks the user to enter a number.
- It tries to convert the input to an integer and calls the factorial function.
- If the input is not a valid integer, it prints an error message.
- The result (or error message) is printed as output = ...

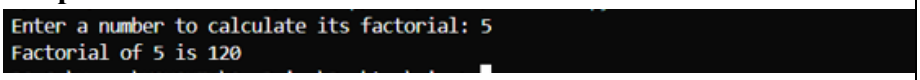
In cursor:

Prompt:
generate a python function that calculates the factorial of a number The function should handle 0! correctly and Negative input by returning an appropriate message and it should take an input and print the output, example: input = 3 output =6 and import math for fast execution

code:



Output:



Code explanation:
The function calculate_factorial() takes user input and calculates the factorial of that number using Python's built-in math module

---

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

Expected Output

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

Prompt:

create a function that parses a nested dictionary representing student information. The function should extract and return Full Name, Branch, SGPA

example:

Input:

Full Name': 'Arjun Mehta',

'Branch': 'Computer Science',

'SGPA': 8.7

Input 2:

Full Name': 'Sneha Reddy',

'Branch': 'Electronics',

'SGPA': 9.2

when printing the details it should just print them or take an input this function should take multiple inputs and give output

CODE:

```
def parse_student_info(students):
    """
    Parses a list of nested dictionaries representing student information.
    Extracts and returns Full Name, Branch, and SGPA for each student.

    Args:
        students (list): List of dictionaries, each representing a student.

    Returns:
        list: List of tuples (Full Name, Branch, SGPA)
    """
    extracted = []
    for student in students:
        name = student.get('Full Name')
        branch = student.get('Branch')
        sgpa = student.get('SGPA')
        extracted.append((name, branch, sgpa))
    return extracted

# Example usage:
if __name__ == "__main__":
    n = int(input("Enter number of students: "))
    students = []
    for i in range(n):
        print(f"\nEnter details for student {i+1}:")
        name = input("Full Name: ")
        branch = input("Branch: ")
        sgpa = float(input("SGPA: "))
        students.append({'Full Name': name, 'Branch': branch, 'SGPA': sgpa})

    details = parse_student_info(students)
    print("\nStudent Details:")
    for name, branch, sgpa in details:
        print(f"Full Name: {name}, Branch: {branch}, SGPA: {sgpa}")
```

OUTPUT:

```
Enter number of students: 3

Enter details for student 1:
Full Name: megha
Branch: cse
SGPA: 9

Enter details for student 2:
Full Name: srikar
Branch: cse
SGPA: 10

Enter details for student 3:
Full Name: sweety
Enter details for student 3:
Full Name: sweety
Full Name: sweety
Branch: cse
SGPA: 10

Student Details:
Full Name: megha, Branch: cse, SGPA: 9.0
Full Name: megha, Branch: cse, SGPA: 9.0
Full Name: srikar, Branch: cse, SGPA: 10.0
Full Name: sweety, Branch: cse, SGPA: 10.0
```

Explanation:

This code calculates the factorial of a given number entered by the user. Here's how it works:

- The factorial(n) function:
  - Returns an error message if the input is negative (since factorial is not defined for negative numbers).
  - Returns 1 if the input is 0 (since 0! = 1).
  - Otherwise, it calculates the factorial by multiplying all integers from 1 to n.
- The script asks the user to enter a number.
- It tries to convert the input to an integer and calls the factorial function.
- If the input is not a valid integer, it prints an error message.
- The result (or error message) is printed as output = ...

In cursor:

Prompt:

create a function that parses a nested dictionary representing student information. The function should extract and return Full Name, Branch, SGPA

example:

Input:

Full Name': 'Arjun Mehta',

'Branch': 'Computer Science',

'SGPA': 8.7

Input 2:

Full Name': 'Sneha Reddy',

'Branch': 'Electronics',
'SGPA': 9.2
when printing the details it should just print them or take an input
this function should take multiple inputs and give output
code:

```python
def extract_student_details(student_dict):

    def flatten(d, parent_key='', sep='_'):
        items = []
        for k, v in d.items():
            new_key = f"{parent_key}{sep}{k}" if parent_key else k
            if isinstance(v, dict):
                items.extend(flatten(v, new_key, sep=sep).items())
            else:
                items.append((new_key, v))
        return dict(items)

    flat_dict = flatten(student_dict)
    full_name = flat_dict.get('Full Name') or flat_dict.get('Full_Name')
    branch = flat_dict.get('Branch')
    sgpa = flat_dict.get('SGPA')
    return (full_name, branch, sgpa)
def get_multiple_students():
    students = []
    n = int(input("Enter number of students: "))
    for i in range(n):
        print(f"\nEnter details for student {i+1}:")
        full_name = input("Full Name: ")
        branch = input("Branch: ")
        sgpa = input("SGPA: ")
        try:
            sgpa = float(sgpa)
        except ValueError:
            print("Invalid SGPA. Setting SGPA to 0.0")
            sgpa = 0.0
        # You can nest this dictionary further if needed  TAB to jump here
        student_dict = {
            'Full Name': full_name,
```

lab 4 > task 3c.py

```python
    def get_multiple_students():
        for i in range(n):
            try:
            except ValueError:
                print("Invalid SGPA. Setting SGPA to 0.0")
                sgpa = 0.0
            # You can nest this dictionary further if needed
            student_dict = {
                'Full Name': full_name,
                'Branch': branch,
                'SGPA': sgpa
            }
            students.append(student_dict)
        return students

    def print_student_details(students):
        print("\nStudent Details:")
        for student in students:
            full_name, branch, sgpa = extract_student_details(student)
            print(f"Full Name: {full_name}, Branch: {branch}, SGPA: {sgpa}")

    if __name__ == "__main__":
        students = get_multiple_students()
        print_student_details(students)
```

Output:

```
Enter number of students: 2

Enter details for student 1:
Full Name: we
Branch: rt
SGPA: 3.6

Enter details for student 2:
Full Name: kg
Branch: sg
SGPA: 9.9

Student Details:
Full Name: we, Branch: rt, SGPA: 3.6
Full Name: kg, Branch: sg, SGPA: 9.9
```

Explanation:

What this script does

- Collects details for multiple students from user input
- Extracts each student's full name, branch, and SGPA (even if nested inside sub-dictionaries)
- Prints the details in a uniform 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

Expected Output

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

zero-shot:

Prompt:

generate functions for CSV file analysis. 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 ,it should read the file task4.txt and print the output

Code:

task 1g.py | task 2g.py | task 3g.py | task 4_1.py | task4.txt

lab 4 > task 4_1.py > count_words

```python
import csv

def read_csv_file(filename):
    with open(filename, 'r', newline='', encoding='utf-8') as file:
        reader = list(csv.reader(file))
    return reader

def total_rows(data):
    return len(data)

def count_empty_rows(data):
    return sum(1 for row in data if all(cell.strip() == '' for cell in row))

def count_words(data):
    return sum(len(' '.join(row).split()) for row in data)

if __name__ == "__main__":
    filename = 'task4.txt'
    data = read_csv_file(filename)
    print("Total number of rows:", total_rows(data))
    print("Number of empty rows:", count_empty_rows(data))
    print("Total number of words:", count_words(data))
```

Output:

```
Total number of rows: 6
Number of empty rows: 3
Total number of words: 13
```

one-shot:

Prompt:

generate functions for CSV file analysis. 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 ,it should read the file task4.txt and print the output Example: For a file with rows [“hello world”] and [“ ”], the output is total_rows=2, empty_rows=1, total_words=2.

Code:

lab 4 > task 4_2.py > ...

```python
import csv

def read_csv_file(filename):
    with open(filename, 'r', newline='', encoding='utf-8') as f:
        reader = csv.reader(f)
        rows = [row for row in reader]
    return rows

def count_total_rows(rows):
    return len(rows)

def count_empty_rows(rows):
    empty_count = 0
    for row in rows:
        # Consider a row empty if all fields are empty or whitespace
        if all(cell.strip() == '' for cell in row):
            empty_count += 1
    return empty_count

def count_total_words(rows):
    word_count = 0
    for row in rows:
        for cell in row:
            word_count += len(cell.strip().split())
    return word_count

if __name__ == "__main__":
    filename = 'task4.txt'
    rows = read_csv_file(filename)
    total_rows = count_total_rows(rows)
    empty_rows = count_empty_rows(rows)
    total_words = count_total_words(rows)
    print(f"total_rows={total_rows}, empty_rows={empty_rows}, total_words={total_wo
```

Output:

```
total_rows=6, empty_rows=3, total_words=13
```

few-shot prompting:

Prompt:

generate functions for CSV file analysis. 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 ,it should read the file task4.txt and print the output ,
Example 1: Input [“Hi there”] → rows=1, empty=0, words=2.
Example 2: Input [“”] → rows=1, empty=1, words=0.
Example 3: Input [“Good morning”, “ ”] → rows=2, empty=1, words=2.
Now generalize.

Code:

lab 4 > task 4_3.py > ...

```python
import csv

def read_csv_file(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        reader = csv.reader(f)
        rows = [','.join(row) for row in reader]
    return rows

def total_rows(rows):
    return len(rows)

def count_empty_rows(rows):
    return sum(1 for row in rows if not row.strip())

def count_words(rows):
    return sum(len(row.strip().split()) for row in rows if row.strip())

def analyze_csv_file(filename):
    rows = read_csv_file(filename)
    total = total_rows(rows)
    empty = count_empty_rows(rows)
    words = count_words(rows)
    print(f"rows={total}, empty={empty}, words={words}")

if __name__ == "__main__":
    analyze_csv_file('task4.txt')
```

Output:

```
rows=6, empty=0, words=13
```

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

Prompt:

generate a Python function that processes text and analyzes word frequency. The function must Accept a paragraph as input, Convert all text to lowercase, Remove punctuation, Return the most frequently used word

Example 1:

Input: "Hello hello world!"

Output: "hello" (since "hello" appears 2 times, more than "world").

Example 2:

Input: "Python, python; PYTHON is great."

Output: "python" (case-insensitive, punctuation removed).

Example 3:

Input: "I love AI. AI is the future, and AI is fun!"

Output: "ai" (appears 3 times).

Now write a Python function that accepts a paragraph, converts text to lowercase, removes punctuation, and returns the most frequent word.that takes input and prints output

Code:

lab 4 > task 5g.py > ...

```python
import string
from collections import Counter

def most_frequent_word(paragraph):
    # Convert to lowercase
    text = paragraph.lower()
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Split into words
    words = text.split()
    # Count word frequencies
    freq = Counter(words)
    # Find the most common word
    if freq:
        most_common_word, _ = freq.most_common(1)[0]
        return most_common_word
    else:
        return ""

# Accept input and print output
if __name__ == "__main__":
    paragraph = input("Enter a paragraph: ")
    print(most_frequent_word(paragraph))
```

Output:

```
Enter a paragraph: hi hi hi hi i am a cat
hi
```

In Cursor:

Prompt:

generate a Python function that processes text and analyzes word frequency. The function must Accept a paragraph as input, Convert all text to lowercase, Remove punctuation, Return the most frequently used word

Example 1:

Input: "Hello hello world!"

Output: "hello" (since "hello" appears 2 times, more than "world").

Example 2:

Input: "Python, python; PYTHON is great."

Output: "python" (case-insensitive, punctuation removed).

Example 3:

Input: "I love AI. AI is the future, and AI is fun!"

Output: "ai" (appears 3 times).

Now write a Python function that accepts a paragraph, converts text to lowercase, removes punctuation, and returns the most frequent word.that takes input and prints output and call the function

Code:

```python
import string

def most_frequent_word(paragraph):
    # Convert to lowercase
    paragraph = paragraph.lower()
    # Remove punctuation
    translator = str.maketrans('', '', string.punctuation)
    paragraph = paragraph.translate(translator)
    # Split into words
    words = paragraph.split()
    # Count frequency
    freq = {}
    for word in words:
        freq[word] = freq.get(word, 0) + 1
    # Find the most frequent word
    if not freq:
        return None
    most_freq_word = max(freq, key=freq.get)
    return most_freq_word

if __name__ == "__main__":
    paragraph = input("Enter a paragraph: ")
    result = most_frequent_word(paragraph)
    print(result)
```

Output:

```
Enter a paragraph: i like like like apples
like
```

Differences:

Git:

- Uses **collections.Counter**, a built-in tool specialized for counting hashable items.
- freq.most_common(1) directly gives you the most frequent word and its count.
- More **concise, efficient, and Pythonic**
- Returns an **empty string ""** if no words exist.

Cursor:

- Manually builds the frequency dictionary using .get().
- Uses max(freq, key=freq.get) to find the word with the highest count.
- More **explicit and beginner-friendly**, but slightly longer.
- Returns **None** if no words exist.

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**