**School of Computer Science and Artificial Intelligence**

**Lab Assignment # 4.1**

**Program : B. Tech (CSE)**

**Specialization : CSE**

**Course Title : AI ASSISTED CODING**

**Course Code : 24CS101PC214**

**Semester : III**

**Academic Session : 2025-2026**

**Name of Student : E.KARTHIK PATEL**

**Enrollment No. : 2403A51416**

**Batch No. : 16**

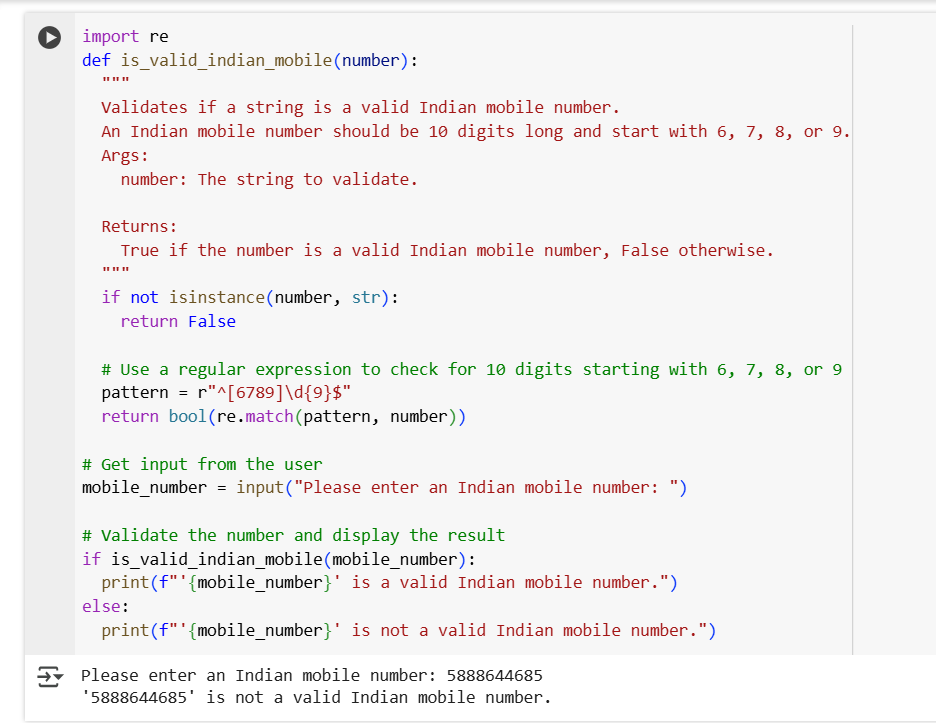
**Date :22/08/2025**

TASK 1

QUESTION :

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

CODE AND OUTPUT:



EXPLANATION :

* import re: This line imports the re module, which provides support for regular expressions. Regular expressions are used here to define the pattern for a valid mobile number.
* def is\_valid\_indian\_mobile(number):: This defines a function named is\_valid\_indian\_mobile that takes one argument, number.
* if not isinstance(number, str): return False: This checks if the input number is not a string. If it's not, the function immediately returns False because a mobile number is expected to be a string.
* pattern = r"^[6789]\d{9}$": This line defines a regular expression pattern.
  + ^: Matches the beginning of the string.
  + [6789]: Matches a single digit that is either 6, 7, 8, or 9.
  + \d{9}: Matches exactly 9 digits (0-9).
  + $: Matches the end of the string.
  + r"": The r indicates a raw string, which is helpful for regular expressions to avoid issues with backslashes.
* return bool(re.match(pattern, number)): This is the core validation step.
  + re.match(pattern, number): Attempts to match the pattern at the beginning of the number string. If a match is found, it returns a match object; otherwise, it returns None.
  + bool(...): Converts the result of re.match to a boolean. If a match object is returned (meaning the pattern matched), bool() returns True. If None is returned (meaning the pattern didn't match), bool() returns False.
* mobile\_number = input("Please enter an Indian mobile number: "): This line prompts the user to enter a mobile number using the input() function and stores the entered value in the mobile\_number variable.
* if is\_valid\_indian\_mobile(mobile\_number): ... else: ...: This is an if-else block that calls the is\_valid\_indian\_mobile function with the user's input.
  + If the function returns True (the number is valid), it prints a message indicating that the number is valid.
  + If the function returns False (the number is not valid), it prints a message indicating that the number is not valid.

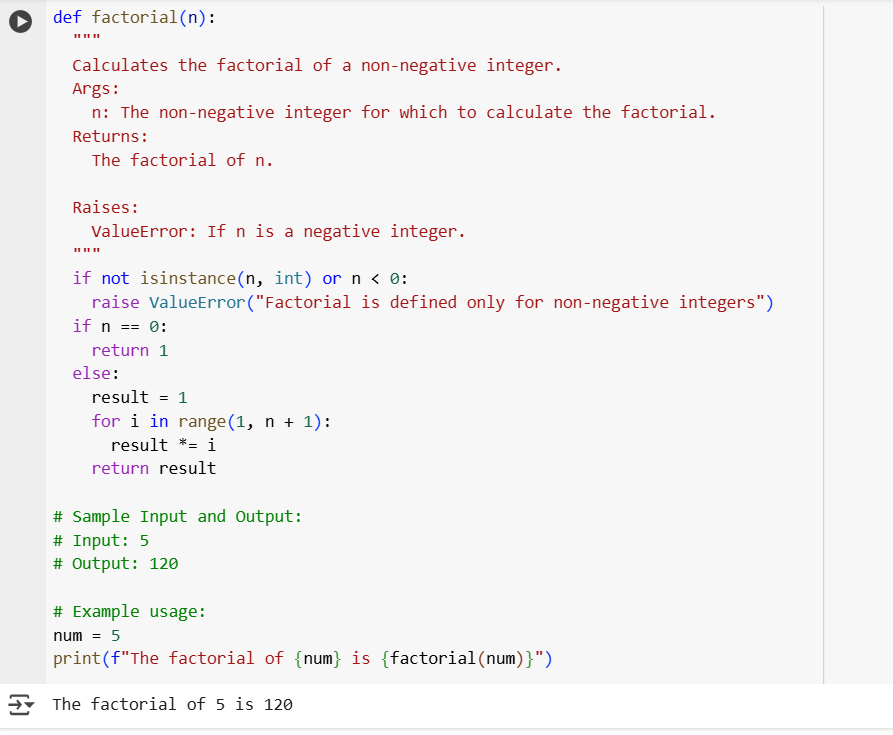
In summary, the code defines a function to check if a string is a valid Indian mobile number based on a specific format using regular expressions, then takes user input and uses the function to tell the user if their input is a valid number.

TASK 2

QUESTION :

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

CODE AND OUTPUT:



EXPLANATION :

This code defines a function to calculate the factorial of a non-negative integer and demonstrates its usage.

Here's a breakdown:

* def factorial(n):: This defines a function named factorial that takes one argument, n.
* """Docstring explaining the function""": This is a docstring, which explains what the function does, its arguments (Args), what it returns (Returns), and any exceptions it might raise (Raises).
* if not isinstance(n, int) or n < 0:: This is an input validation step. It checks if the input n is not an integer or if it is a negative integer.
  + isinstance(n, int): Checks if n is an integer.
  + n < 0: Checks if n is less than 0.
  + not ... or ...: The condition is true if n is not an integer OR if n is negative.
* raise ValueError("Factorial is defined only for non-negative integers"): If the input validation fails (i.e., n is not a non-negative integer), this line raises a ValueError with a descriptive message. This stops the function execution and signals that the input was invalid.
* if n == 0:: This checks if n is equal to 0.
* return 1: If n is 0, the factorial is defined as 1, so the function returns 1.
* else:: This block is executed if n is a positive integer.
* result = 1: Initializes a variable result to 1. This will store the calculated factorial.
* for i in range(1, n + 1):: This starts a for loop that iterates from 1 up to and including n.
* result \*= i: In each iteration, the current value of result is multiplied by the loop variable i. This performs the cumulative multiplication needed for the factorial.
* return result: After the loop finishes, the function returns the final calculated result, which is the factorial of n.
* # Sample Input and Output:: These lines are comments providing an example of what input and output to expect.
* # Input: 5: Sample input value.
* # Output: 120: Expected output for the sample input.
* # Example usage:: A comment indicating the start of the example usage.
* num = 5: Assigns the value 5 to the variable num.
* print(f"The factorial of {num} is {factorial(num)}"): This line demonstrates how to call the factorial function and print the result.
  + factorial(num): Calls the factorial function with num (which is 5) as the argument.
  + f"...": This is an f-string, which allows embedding the value of variables directly within a string.
  + print(...): Prints the formatted string to the console.

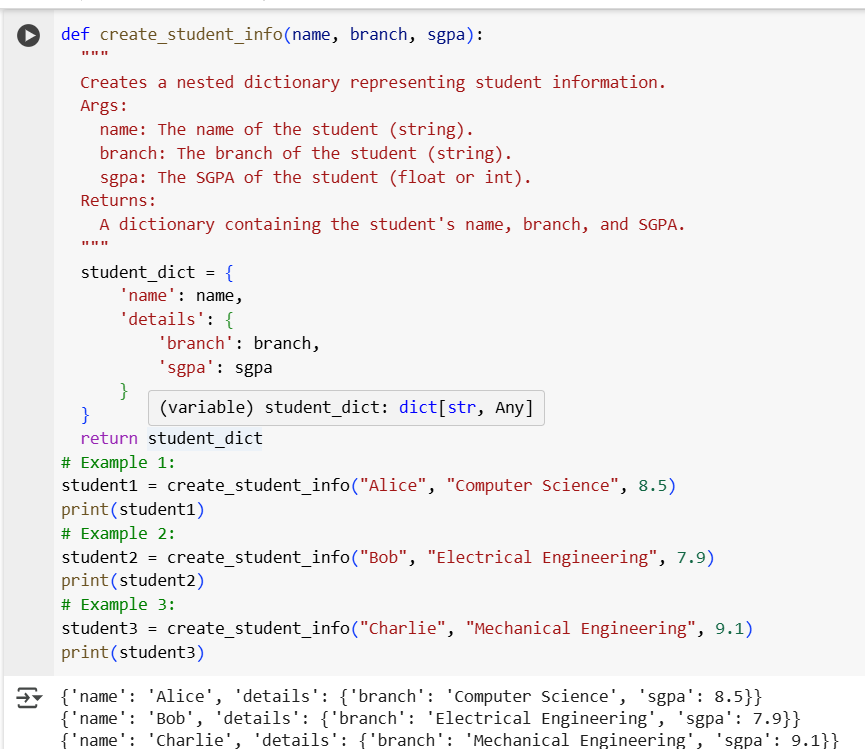
In summary, the code defines a function that correctly calculates the factorial for non-negative integers, includes error handling for invalid inputs, and provides a clear example of how to use the function.

TASK 3

QUESTION :

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 AND OUTPUT :



EXPLANATION:

This code defines a Python function that creates a nested dictionary to store student information (name, branch, and SGPA) and provides examples of how to use this function.

Here's a breakdown:

* def create\_student\_info(name, branch, sgpa):: This defines a function named create\_student\_info that takes three arguments: name, branch, and sgpa.
* """Docstring explaining the function""": This is a docstring that explains what the function does, its arguments (Args), and what it returns (Returns).
* student\_dict = { ... }: This line creates a dictionary named student\_dict.
  + 'name': name: This key-value pair stores the student's name using the name argument passed to the function.
  + 'details': { ... }: This key-value pair creates a nested dictionary under the key 'details'. This nested dictionary contains additional information about the student.
  + 'branch': branch: Inside the nested dictionary, this key-value pair stores the student's branch using the branch argument.
  + 'sgpa': sgpa: Also inside the nested dictionary, this key-value pair stores the student's SGPA using the sgpa argument.
* return student\_dict: After creating the dictionary, the function returns the student\_dict.
* # Example 1:**,**# Example 2:**,**# Example 3:: These are comments indicating the start of example usage.
* student1 = create\_student\_info("Alice", "Computer Science", 8.5): This line calls the create\_student\_info function with the arguments "Alice", "Computer Science", and 8.5. The returned dictionary is assigned to the variable student1.
* print(student1): This line prints the contents of the student1 dictionary.
* The subsequent example blocks (student2, student3) follow the same pattern, creating and printing dictionaries for different students with their respective information.

In summary, the code provides a reusable function to structure student data into a clear, nested dictionary format, making it easy to store and access related information for each student. The examples demonstrate how to use the function to create and view these dictionaries.

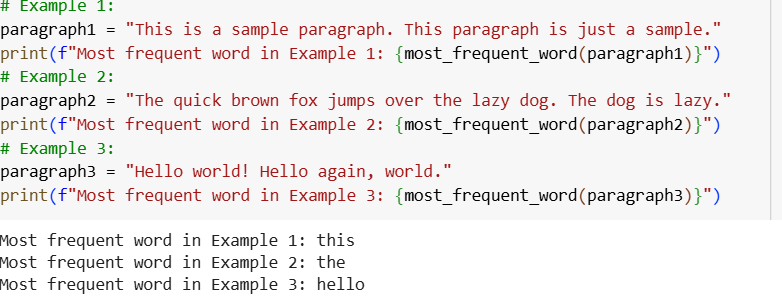
TASK 4

QUESTION:

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

CODE AND OUTPUT:





EXPLANATION :

This code defines a Python function called most\_frequent\_word. This function is designed to find the most frequently used word within a given text paragraph. It first converts the entire input paragraph to lowercase to ensure case-insensitivity in word counting. Next, it removes all punctuation marks from the text. The cleaned text is then split into individual words. The code uses the Counter object from the collections module to efficiently count the occurrences of each word. Finally, it identifies and returns the word with the highest frequency count. The included examples demonstrate the function's usage with different sample paragraphs.

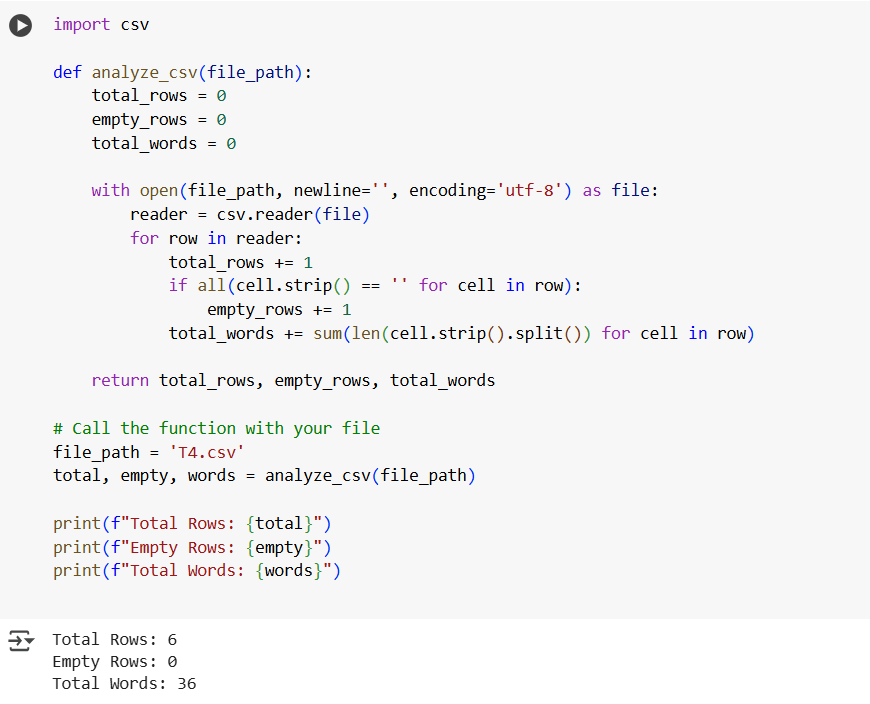
TASK 5

QUESTION :

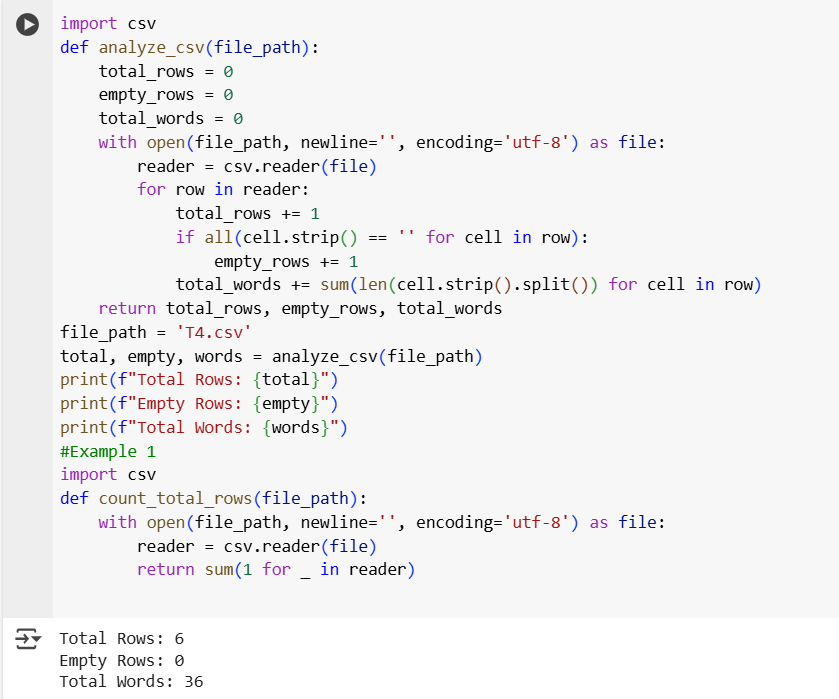
Experiment with zero-shot, one-shot, and few-shot prompting to  
generate functions for CSV file analysis.  
Requirements  
• Each generated function should:  
o Read a .csv file  
o Return the total number of rows  
o Count the number of empty rows  
o Count the number of words across the file

CODE AND OUTPUT:

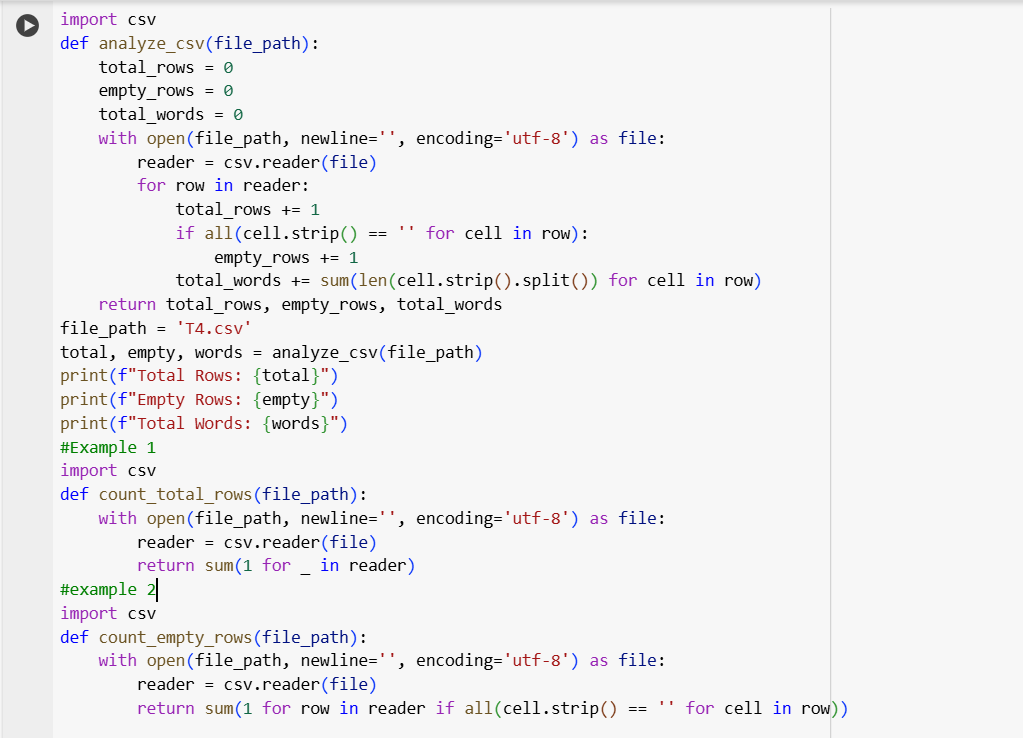
zero-shot

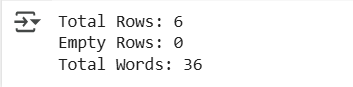


one-shot



few-shot





**Explanation :**

All three approaches aim to generate a Python function that reads a CSV file (T4.csv) and returns:

The total number of rows,

The number of completely empty rows,

The total number of words across all cells.

The core logic of the function is the same in each case:

It opens the file using csv.reader

Iterates over each row to:

Count it as a total row

Check if all cells are empty (for empty row count)

Split and count words in each cell (for word count)

Then returns the three values and prints them.

The difference lies in how the model is prompted:

Zero-shot gives no example—just instructions.

One-shot gives one simple related example (e.g., just counting rows).

Few-shot gives multiple related examples (e.g., counting total rows and empty rows), helping the model better understand and combine tasks.