

## AI LAB ASSIGNMENT-4.1

NAME : PULI.ALA

ROLL NO.: 2403A52394

BATCH NO.:14

COURSE NAME: AI ASSISTED CODING

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

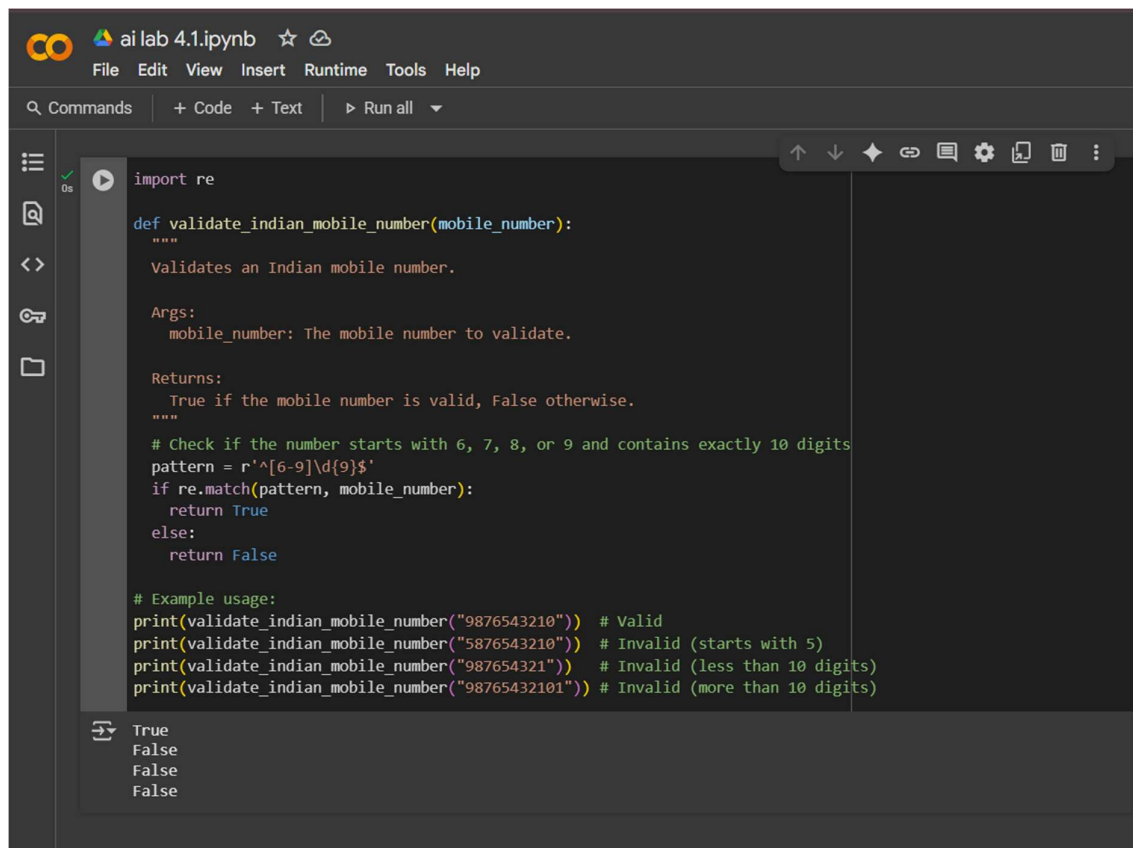
#### Objective

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:
  - Starts with 6, 7, 8, or 9
  - Contains exactly 10 digits

Code:



The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar includes the 'ai lab 4.1.ipynb' title and a menu with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu is a 'Commands' bar with '+ Code', '+ Text', and 'Run all' buttons. The left sidebar contains icons for file management and search. The main area displays a Python function `validate_indian_mobile_number` with its docstring and example usage. The output of the function is shown in a separate cell below the code.

```
import re

def validate_indian_mobile_number(mobile_number):
    """
    Validates an Indian mobile number.

    Args:
        mobile_number: The mobile number to validate.

    Returns:
        True if the mobile number is valid, False otherwise.
    """
    # Check if the number starts with 6, 7, 8, or 9 and contains exactly 10 digits
    pattern = r'^[6-9]\d{9}$'
    if re.match(pattern, mobile_number):
        return True
    else:
        return False

# Example usage:
print(validate_indian_mobile_number("9876543210")) # Valid
print(validate_indian_mobile_number("5876543210")) # Invalid (starts with 5)
print(validate_indian_mobile_number("987654321")) # Invalid (less than 10 digits)
print(validate_indian_mobile_number("98765432101")) # Invalid (more than 10 digits)
```

```
True
False
False
False
```

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

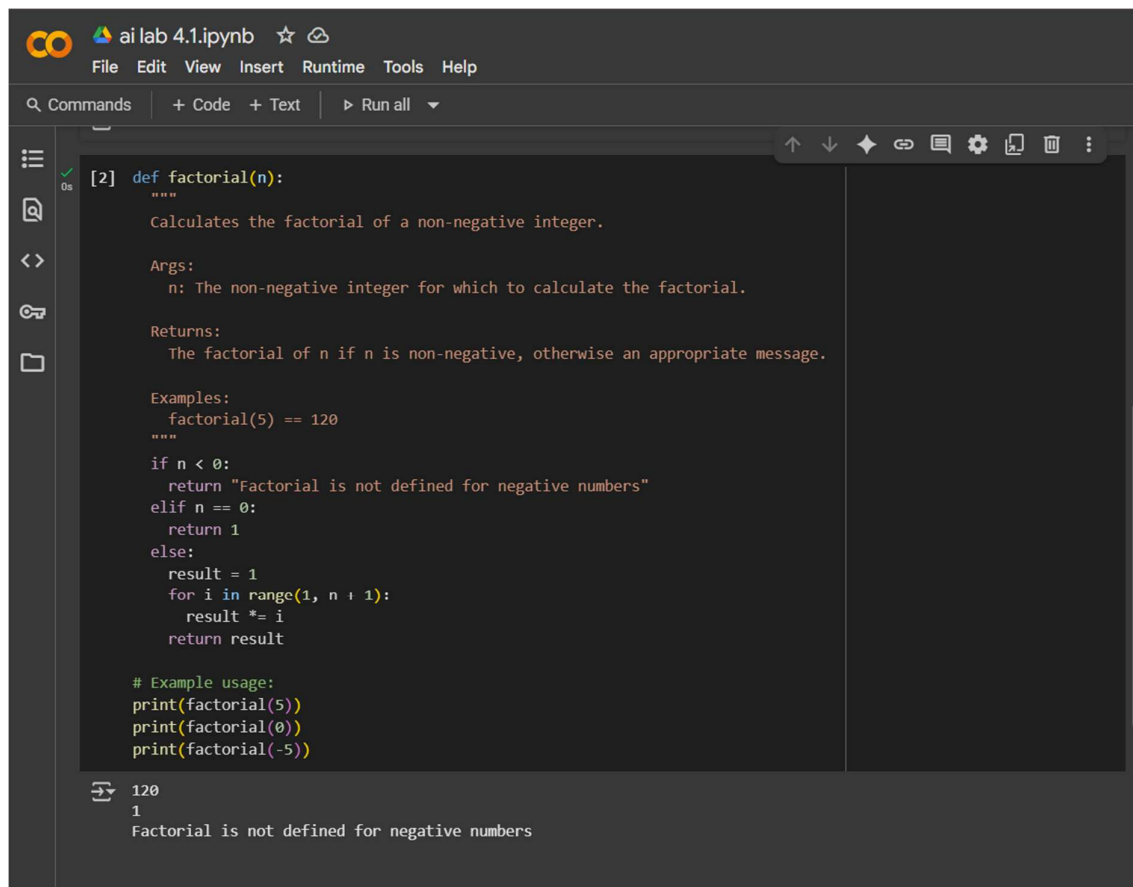
### Objective

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

Code:



The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar includes the 'ai lab 4.1.ipynb' title and standard menu items (File, Edit, View, Insert, Runtime, Tools, Help). Below the menu is a toolbar with icons for commands, code, text, and running. The main area contains a code cell with a Python function definition for a factorial. The function includes docstrings, argument descriptions, return value descriptions, and example usage. The output of the cell shows the results of calling the function with 5, 0, and -5.

```
[2] def factorial(n):  
    """  
    Calculates the factorial of a non-negative integer.  
  
    Args:  
        n: The non-negative integer for which to calculate the factorial.  
  
    Returns:  
        The factorial of n if n is non-negative, otherwise an appropriate message.  
  
    Examples:  
        factorial(5) == 120  
    """  
    if n < 0:  
        return "Factorial is 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))  
print(factorial(0))  
print(factorial(-5))
```

120  
1  
Factorial is 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:
  - Full Name
  - Branch
  - SGPA

CODE:

```
ai lab 4.1.ipynb ☆
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text ▶ Run all ▼

def extract_student_info(student_data):
    """
    Extracts Full Name, Branch, and SGPA from a nested dictionary of student information.

    Args:
        student_data: A dictionary containing student information.

    Returns:
        A dictionary containing the extracted Full Name, Branch, and SGPA.

    Examples:
        >>> student1 = {
        ...     "student_id": "S101",
        ...     "personal_info": {
        ...         "name": {
        ...             "first": "Alice",
        ...             "last": "Smith"
        ...         },
        ...         "contact": {
        ...             "email": "alice.smith@example.com",
        ...             "phone": "123-456-7890"
        ...         }
        ...     },
        ...     "academic_info": {
        ...         "branch": "Computer Science",
        ...         "sgpa": 8.5,
        ...         "courses": ["CS101", "MA101"]
        ...     }
        ... }
        >>> extract_student_info(student1)
        {'Full Name': 'Alice Smith', 'Branch': 'Computer Science', 'SGPA': 8.5}
```

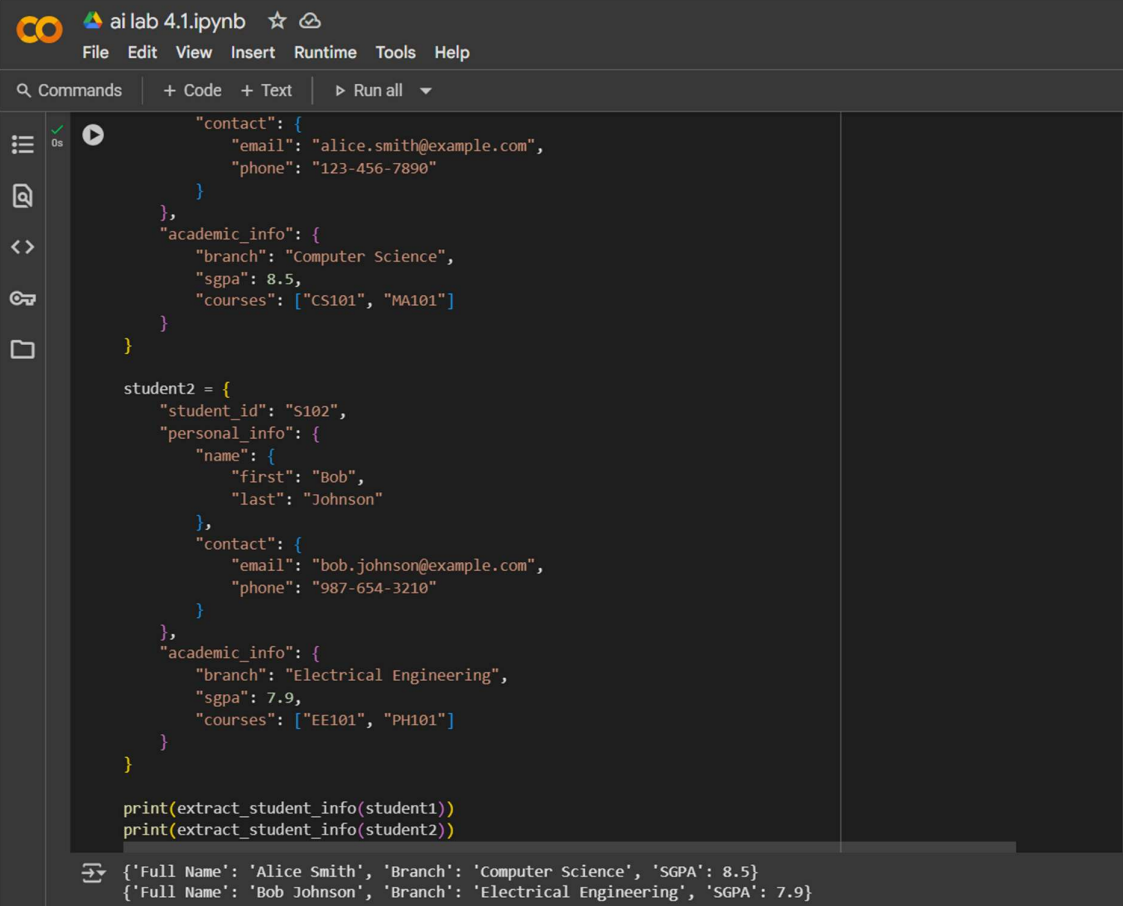
```
Q Commands + Code + Text ▶ Run all ▼

>>> student2 = {
...     "student_id": "S102",
...     "personal_info": {
...         "name": {
...             "first": "Bob",
...             "last": "Johnson"
...         },
...         "contact": {
...             "email": "bob.johnson@example.com",
...             "phone": "987-654-3210"
...         }
...     },
...     "academic_info": {
...         "branch": "Electrical Engineering",
...         "sgpa": 7.9,
...         "courses": ["EE101", "PH101"]
...     }
... }
>>> extract_student_info(student2)
{'Full Name': 'Bob Johnson', 'Branch': 'Electrical Engineering', 'SGPA': 7.9}

full_name = student_data["personal_info"]["name"]["first"] + " " + student_data["personal_info"]["name"]["last"]
branch = student_data["academic_info"]["branch"]
sgpa = student_data["academic_info"]["sgpa"]

return {"Full Name": full_name, "Branch": branch, "SGPA": sgpa}

# Example usage:
student1 = {
    "student_id": "S101",
    "personal_info": {
        "name": {
            "first": "Alice",
            "last": "Smith"
        },
        "contact": {
            "email": "alice.smith@example.com",
            "phone": "123-456-7890"
        }
    },
    "academic_info": {
        "branch": "Computer Science",
        "sgpa": 8.5,
        "courses": ["CS101", "MA101"]
    }
}
```



The screenshot shows a JupyterLab environment with a dark theme. The top bar includes the 'ai lab 4.1.ipynb' title and a menu with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu is a 'Commands' search bar and buttons for '+ Code', '+ Text', and 'Run all'. The main editor area contains Python code defining two student objects, 'student1' and 'student2', each with 'contact', 'academic\_info', and 'personal\_info' fields. At the bottom, two print statements call 'extract\_student\_info' for each student. The output area at the very bottom shows the results of these calls as dictionaries.

```
"contact": {
    "email": "alice.smith@example.com",
    "phone": "123-456-7890"
},
"academic_info": {
    "branch": "Computer Science",
    "sgpa": 8.5,
    "courses": ["CS101", "MA101"]
}

student2 = {
    "student_id": "S102",
    "personal_info": {
        "name": {
            "first": "Bob",
            "last": "Johnson"
        },
        "contact": {
            "email": "bob.johnson@example.com",
            "phone": "987-654-3210"
        },
        "academic_info": {
            "branch": "Electrical Engineering",
            "sgpa": 7.9,
            "courses": ["EE101", "PH101"]
        }
    }

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

```
{'Full Name': 'Alice Smith', 'Branch': 'Computer Science', 'SGPA': 8.5}
{'Full Name': 'Bob Johnson', 'Branch': 'Electrical Engineering', 'SGPA': 7.9}
```

## 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

Code:

```
Untitled2.ipynb ☆
File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

import re
from collections import Counter

def most_frequent_word(paragraph):
    """
    Processes text and returns the most frequently used word.

    Args:
        paragraph: The input text paragraph.

    Returns:
        The most frequently used word in the paragraph.

    Examples:
    >>> most_frequent_word("This is a sample paragraph. This paragraph is a sample.")
    'this'

    >>> most_frequent_word("The quick brown fox jumps over the lazy dog. The dog is lazy.")
    'the'

    >>> most_frequent_word("A a A b b c")
    'a'
    """
    # Convert to lowercase
    paragraph = paragraph.lower()

    # Remove punctuation
    paragraph = re.sub(r'[^\w\s]', '', paragraph)

    # Split into words and count frequency
    words = paragraph.split()
    word_counts = Counter(words)
```

```
Untitled2.ipynb ☆
File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

# Return the most frequent word
if word_counts:
    return word_counts.most_common(1)[0][0]
else:
    return None

# Example usage:
print(most_frequent_word("This is a sample paragraph. This paragraph is a sample. "))
print(most_frequent_word("The quick brown fox jumps over the lazy dog. The dog is lazy. "))
print(most_frequent_word("A a A b b c"))
print(most_frequent_word(""))

this
the
a
None
```