# AI PYTHON LAB TEST 1

**NAME: MOHAMMED FARNAS ALI MUDABBIR**

**BRANCH: MTECH CSE**

**ROLL NO: 2503B05136**

**1. Q1. Write a Python function to calculate the factorial of a number. Use a coding assistant and document how you refine your prompt to ensure the function handles negative input and zeros. (Each question carries 5 marks)**

**Prompt:**

**Role:** You are an expert Python coding assistant.

**Task:** Write a Python function named factorial(n) that correctly calculates the factorial of a number.

**Requirements:**

Use a clean and efficient approach (recursion or iteration).

Include a detailed docstring describing the function, parameters, return value, and errors.

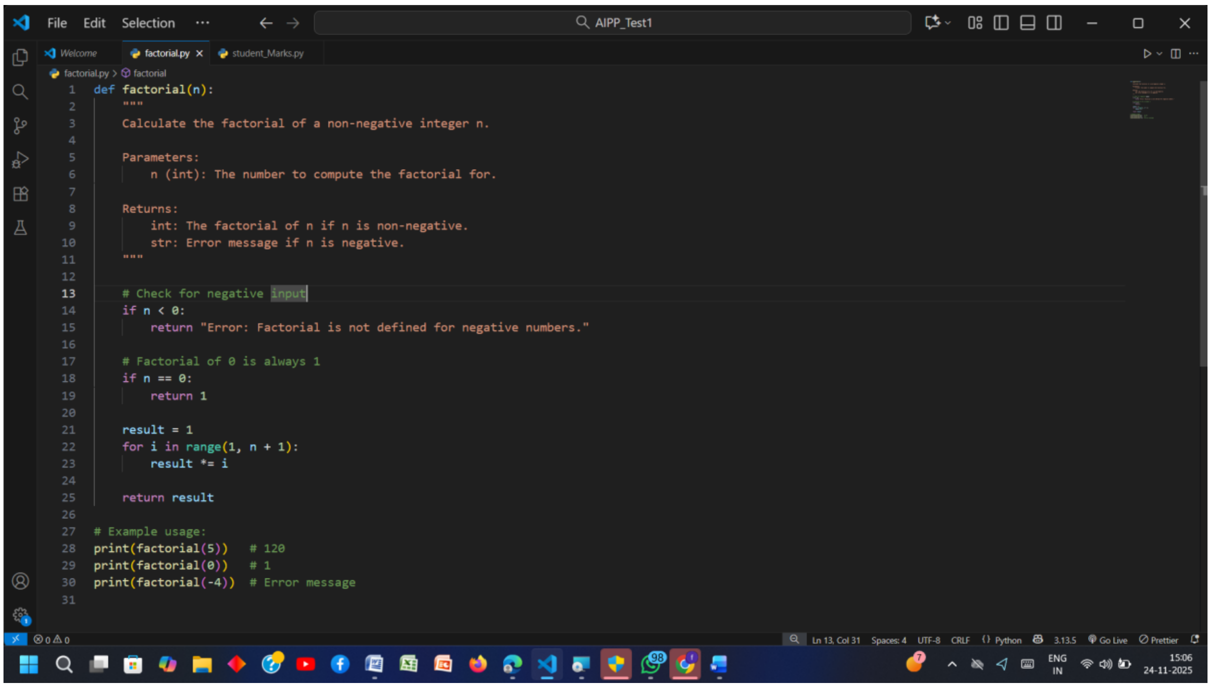
The function must return 1 when the input is 0.

If the input is negative or not an integer, raise a ValueError with a clear message.

Provide example usage after the function.

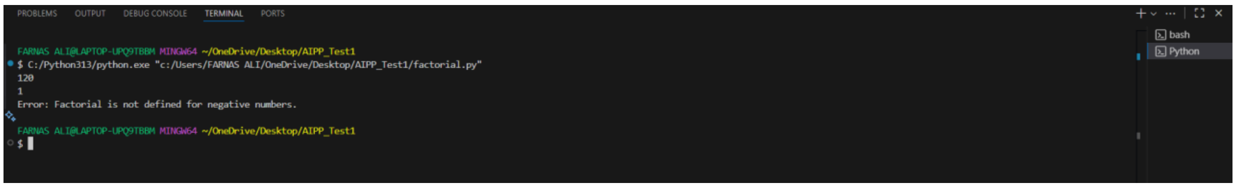
**Output Format**: Return only the final Python code in a code block.

**Code:**



```python
def factorial(n):
    """
    Calculate the factorial of a non-negative integer n.

    Parameters:
        n (int): The number to compute the factorial for.

    Returns:
        int: The factorial of n if n is non-negative.
        str: Error message if n is negative.
    """

    # Check for negative input
    if n < 0:
        return "Error: Factorial is not defined for negative numbers."

    # Factorial of 0 is always 1
    if n == 0:
        return 1

    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(-4))  # Error message
```

**PRACTICAL OUTPUT:**



```
FARNAS ALI@LAPTOP-UPQ9TBBM MINGW64 ~/OneDrive/Desktop/AIPP_Test1
$ C:/Python313/python.exe "c:/Users/FARNAS ALI/OneDrive/Desktop/AIPP_Test1/factorial.py"
120
1
Error: Factorial is not defined for negative numbers.

FARNAS ALI@LAPTOP-UPQ9TBBM MINGW64 ~/OneDrive/Desktop/AIPP_Test1
$ 
```

**2. Q2. Given a list of student marks, compute the mean and list all students above the mean. Demonstrate zero-shot, one-shot, and few-shot prompts to guide an AI tool to solve this. Summarize the differences in approach.**

**(Each question carries 5 marks)**

## 1. ZERO-SHOT PROMPT

**Zero-Shot Prompt**

*"Given a list of student marks, compute the mean and list all students who scored above the mean."*

**AI Output (Expected)**

The AI must infer the structure and steps without any example.

## 2. ONE-SHOT PROMPT

**One-Shot Prompt**

*"Here is an example:
Input: {'A': 50, 'B': 80, 'C': 70}
Output: Mean = 66.67, Above mean = ['B', 'C']

Now solve this: Given a list of student marks, compute the mean and list the students who scored above the mean."*

**AI Output (Expected)**

Uses the single example as a pattern for formatting and logic.

## 3. FEW-SHOT PROMPT

**Few-Shot Prompt**

*"Follow the examples below:*
Example 1:
Input: {'A': 60, 'B': 90, 'C': 40}
Output: Mean = 63.33, Above mean = ['B']

Example 2:
Input: {'John': 72, 'Emma': 88, 'Ryan': 65}
Output: Mean = 75, Above mean = ['Emma']

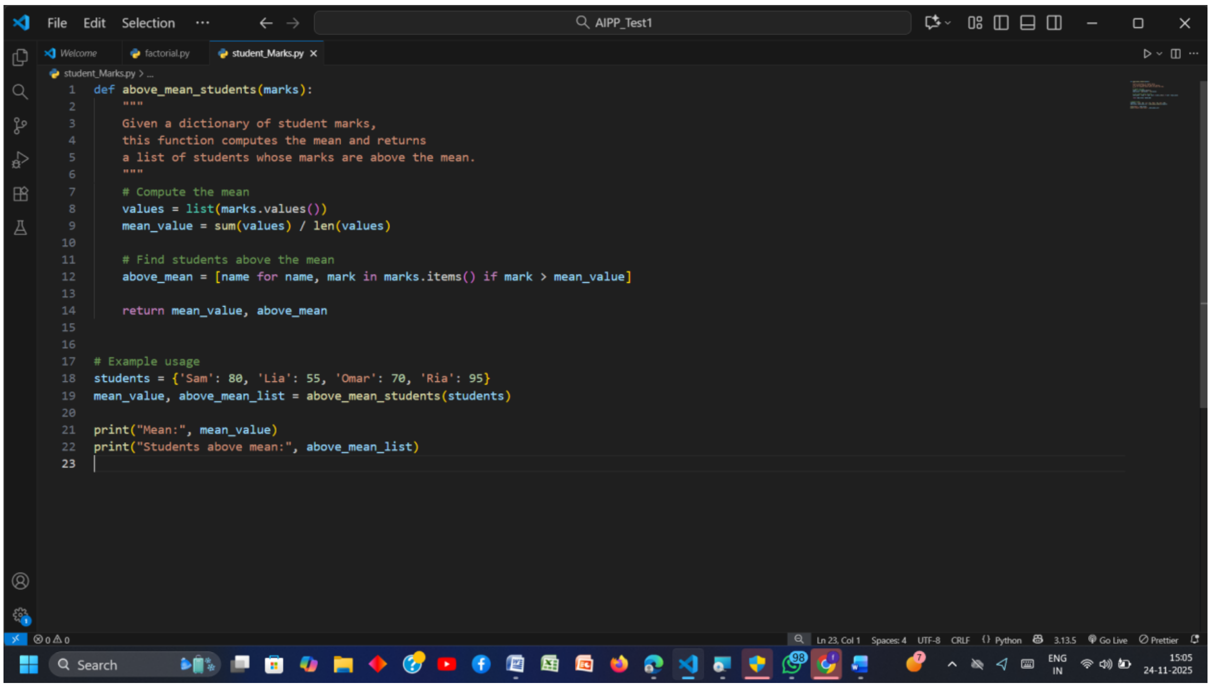
*Now process this new list:
'Sam': 80, 'Lia': 55, 'Omar': 70, 'Ria': 95.
Compute the mean and list all students above the mean."*

**AI Output (Expected)**

AI now imitates the style, structure, and reasoning from multiple examples.

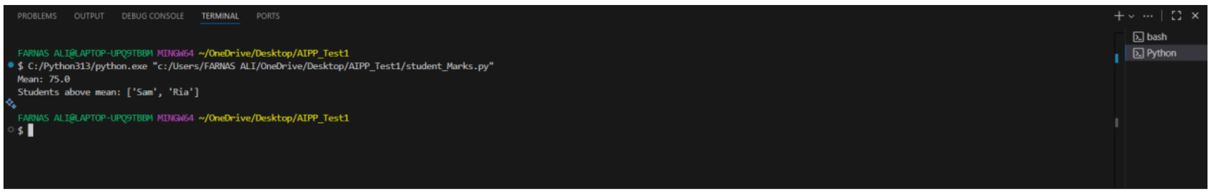
| **Prompt Type** | **Description** | **Example Use** | **AI Behavior** |
|---|---|---|---|
| Zero-Shot | No examples provided. AI must figure out the task from instructions alone. | Quick tasks, general instructions | May produce inconsistent formatting or interpretations. |
| One-Shot | One example is given. | Teaching the expected answer format. | More consistent; AI follows the style of the example. |
| Few-Shot | Multiple examples provided. | Complex tasks requiring structure or reasoning. | Highest accuracy and consistency; AI learns the pattern clearly. |

**Code:**



```python
def above_mean_students(marks):
    """

    Given a dictionary of student marks,
    this function computes the mean and returns
    a list of students whose marks are above the mean.
    """
    # Compute the mean
    values = list(marks.values())
    mean_value = sum(values) / len(values)

    # Find students above the mean
    above_mean = [name for name, mark in marks.items() if mark > mean_value]

    return mean_value, above_mean


# Example usage
students = {'Sam': 80, 'Lia': 55, 'Omar': 70, 'Ria': 95}
mean_value, above_mean_list = above_mean_students(students)

print("Mean:", mean_value)
print("Students above mean:", above_mean_list)
```

**PRACTICAL OUTPUT:**



```
FARNAS ALI@LAPTOP-UPQ9TBBM MINGW64 ~/OneDrive/Desktop/AIPP_Test1
$ C:/Python313/python.exe "c:/Users/FARNAS ALI/OneDrive/Desktop/AIPP_Test1/student_Marks.py"
Mean: 75.0
Students above mean: ['Sam', 'Ria']

FARNAS ALI@LAPTOP-UPQ9TBBM MINGW64 ~/OneDrive/Desktop/AIPP_Test1
$
```