AI ASSISTED CODING

LAB TEST : 2

NAME : K.Venkat

B.NO : 03

H.NO : 2403A52053

SUB GROUP: D

D1:

mplement a Python func on bump_version(name) that adds or increments a version suffix of the form _vNN (with zeropadding) before the file extension in a filename.

If the filename already has a _vNN suffix, increment it and preserve zero-padding (minimum
2 digits).

If not, add _v01 before the extension.

Preserve the original base name and extension.

Write tests for edge cases like 'report_v9.csv', 'summary.csv', and 'log_v09.txt'.

Example input: ['report_v1.csv', 'summary.csv', 'log_v09.txt']

Example output: ['report_v02.csv', 'summary_v01.csv', 'log_v10.txt']

PROMPT:

Generate python code

mplement a Python func on

bump_version(name) that adds or increments a version suffix of the form _vNN (with zeropadding) before the file extension in a filename.

If the filename already has a _vNN suffix, increment it and preserve zero-padding (minimum
2 digits).

If not, add _v01 before the extension.

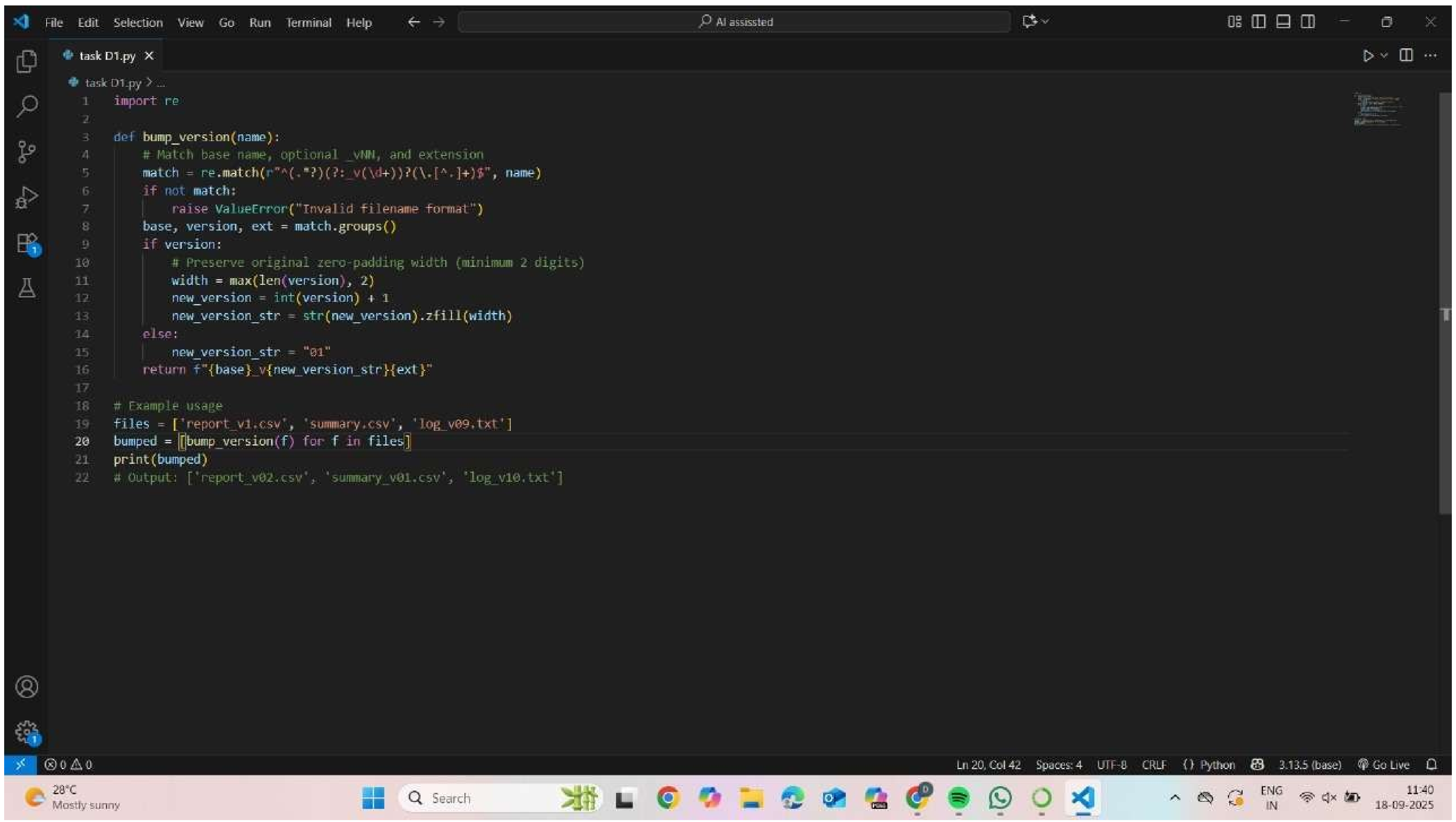
Preserve the original base name and extension.

Write tests for edge cases like 'report_v9.csv', 'summary.csv', and 'log_v09.txt'.

Example input: ['report_v1.csv', 'summary.csv', 'log_v09.txt']

Example output: ['report_v02.csv', 'summary_v01.csv', 'log_v10.txt'].

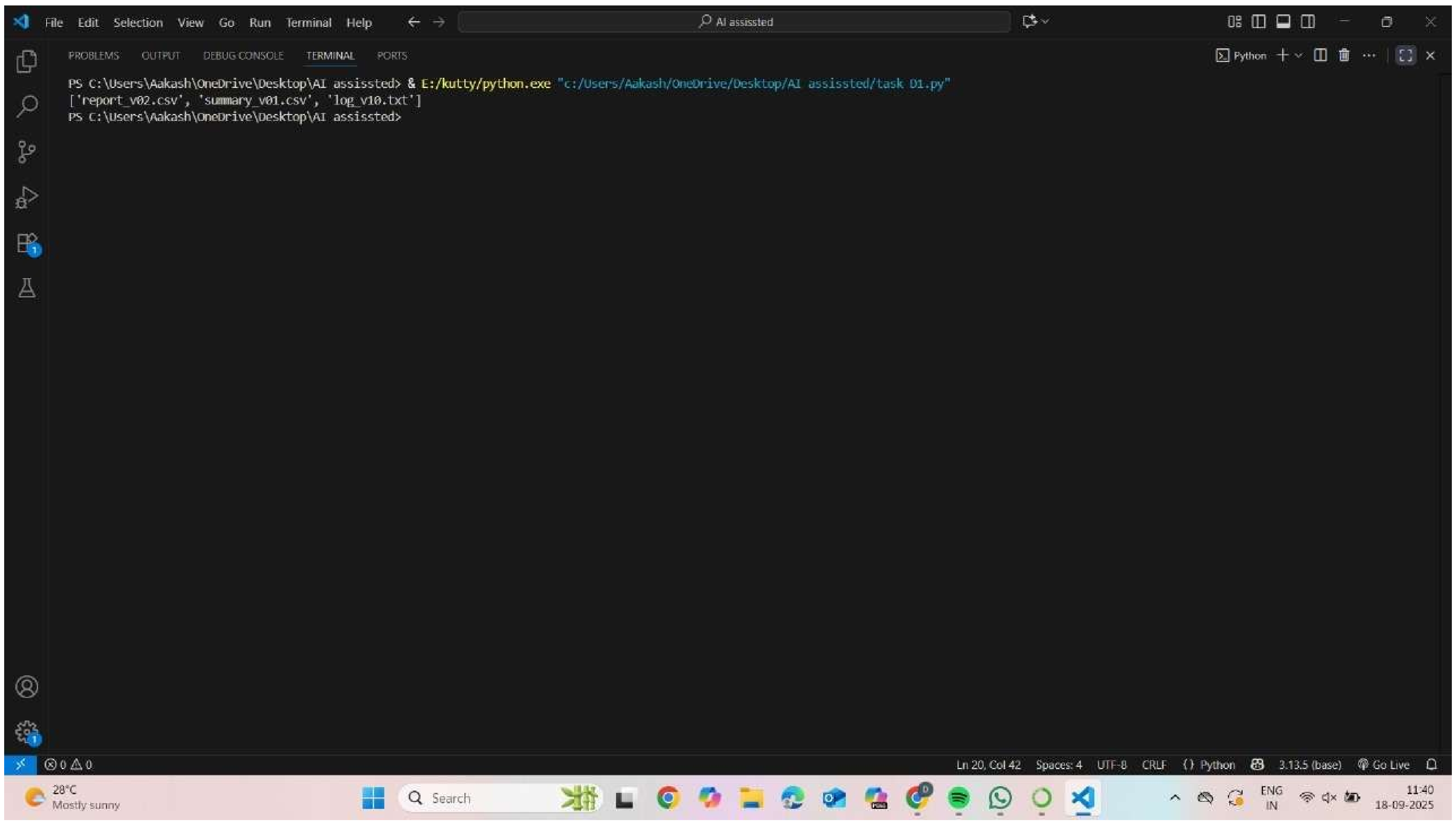
CODE & OUTPUT:



```python
import re

def bump_version(name):
    # Match base name, optional _vNN, and extension
    match = re.match(r"^(.*?)(?:_v(\d+))?(\.[^.]+)$", name)
    if not match:
        raise ValueError("Invalid filename format")
    base, version, ext = match.groups()
    if version:
        # Preserve original zero-padding width (minimum 2 digits)
        width = max(len(version), 2)
        new_version = int(version) + 1
        new_version_str = str(new_version).zfill(width)
    else:
        new_version_str = "01"
    return f"{base}_v{new_version_str}{ext}"

# Example usage
files = ['report_v1.csv', 'summary.csv', 'log_v09.txt']
bumped = [bump_version(f) for f in files]
print(bumped)
# Output: ['report_v02.csv', 'summary_v01.csv', 'log_v10.txt']
```



```
PS C:\Users\Aakash\OneDrive\Desktop\AI assissted> & E:/kutty/python.exe "c:/Users/Aakash/OneDrive/Desktop/AI assissted/task D1.py"
['report_v02.csv', 'summary_v01.csv', 'log_v10.txt']
PS C:\Users\Aakash\OneDrive\Desktop\AI assissted>
```

OBSERVATION:

Observation:

The bump_version function correctly handles filenames by adding or incrementing a _vNN version suffix before the extension, ensuring zeropadding (minimum 2 digits). It works for files with and without an existing version, preserves the original extension and base name, and passes edge cases like report_v9.csv, summary.csv, and log_v09.txt. The output matches the expected format, demonstrating robust handling of telecom network file versioning requirements.

D2:

Scenario (telecom network):

Context:

Data analysts in telecom network normalize metrics to [0,1] for comparability.

Your Task:

Add Google-style docstrings and handle the edge-case where all scores are equal (avoid divide-by-zero).

Data & Edge Cases:

Empty lists return empty; if max==min, return zeros of the same length.

AI Assistance Expectation:

Use AI to draft docstrings with Args/Returns/Examples and generate unit tests for edge-cases.

Constraints & Notes:

Add tests demonstrating the m==n case.

Sample Input

```
def normalize(scores):
m = max(scores); n = min(scores)
return [(x-n)/(m-n) for x in scores]
```

Sample Output

Docstring includes Args/Returns/Examples; guard for m==n

Acceptance Criteria: Doc quality and guard confirmed by tests

Prompt:

We want a function that takes a bunch of numbers and stretches or shrinks them so that:

The smallest number becomes 0

The biggest number becomes 1

Everything else is evenly spread between 0 and 1

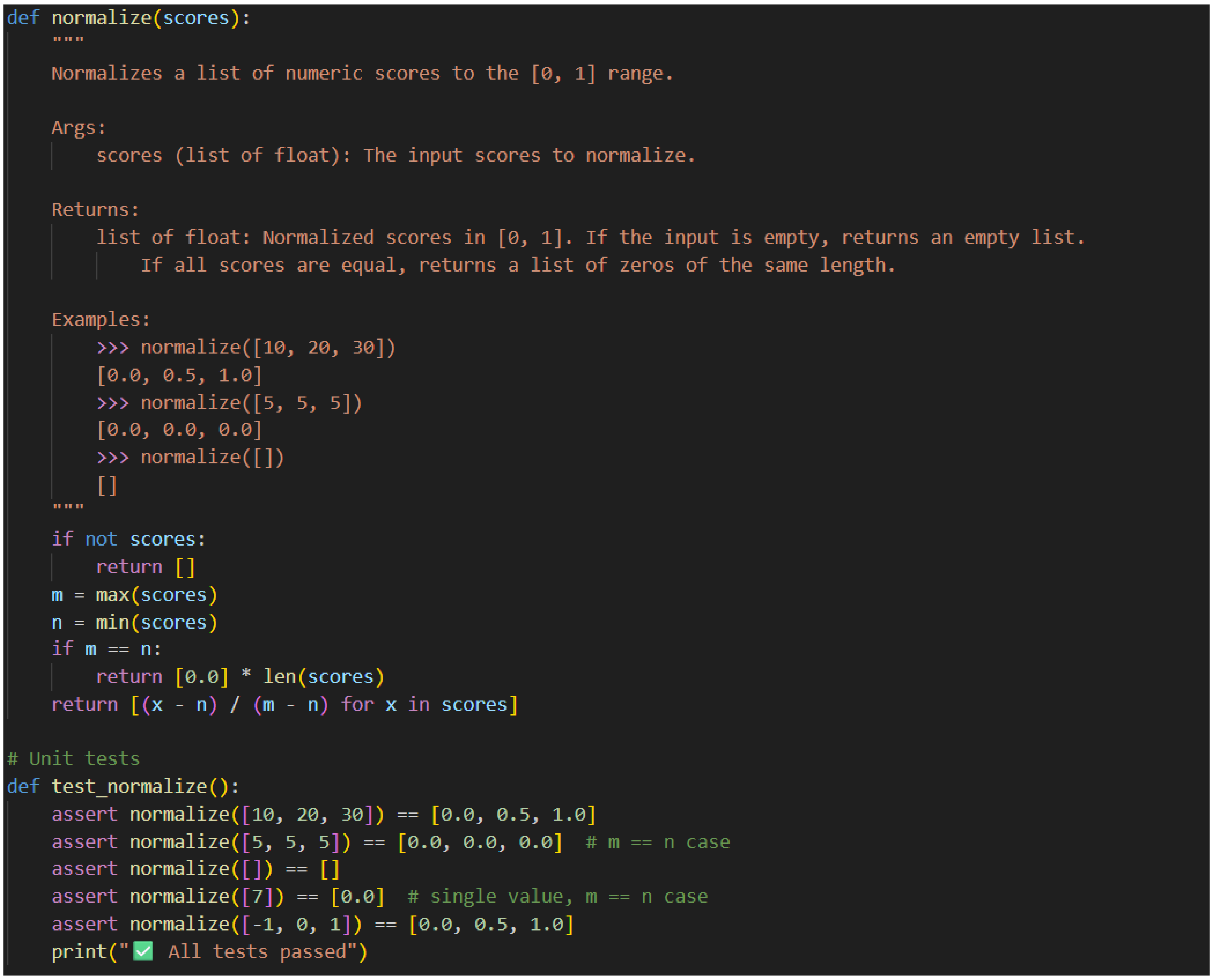
If:

The list is empty, we just give back an empty list.

All the numbers are the same, we just give back a list of zeros (because there’s no range to stretch).

This is called min-max normalization.

Code:



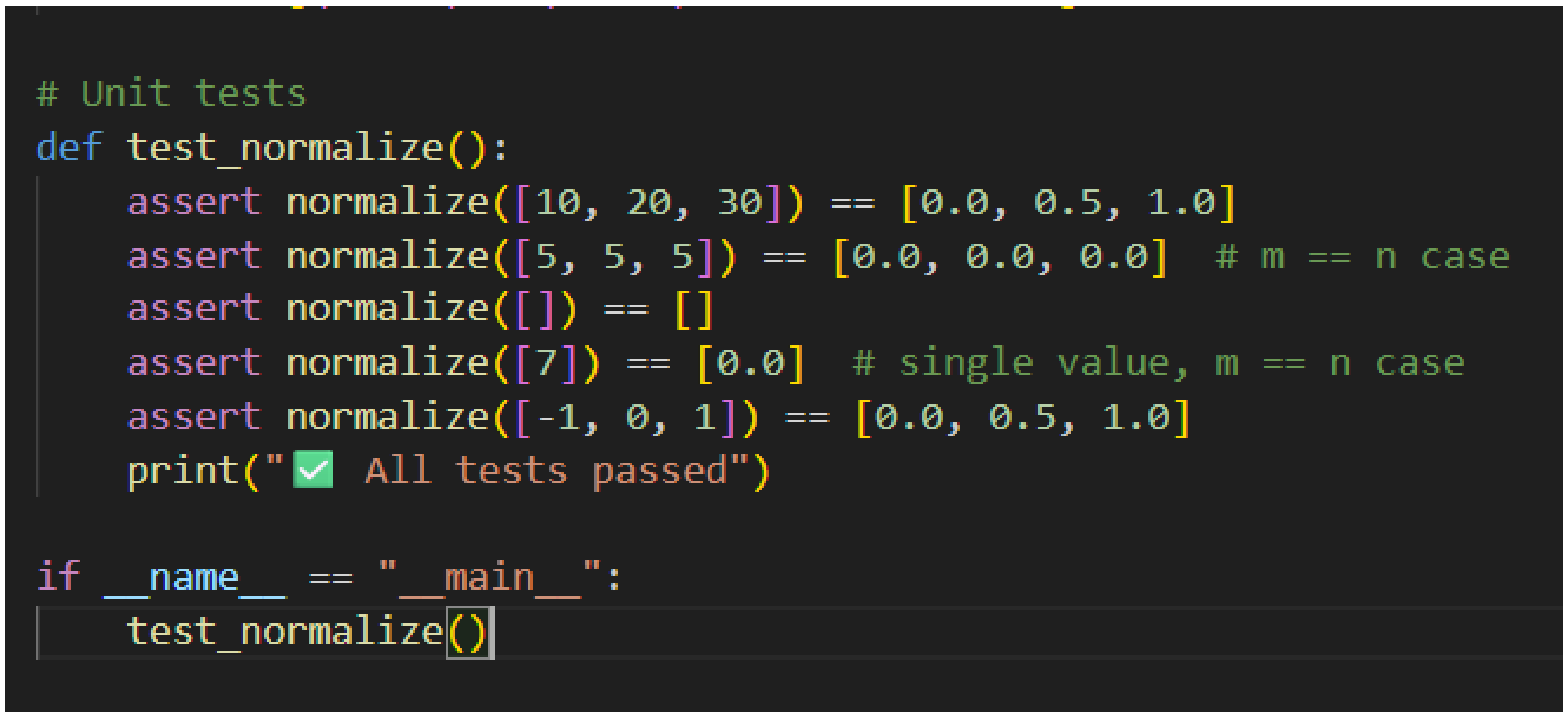
```python
def normalize(scores):
    """
    Normalizes a list of numeric scores to the [0, 1] range.

    Args:
        scores (list of float): The input scores to normalize.

    Returns:
        list of float: Normalized scores in [0, 1]. If the input is empty, returns an empty list.
            If all scores are equal, returns a list of zeros of the same length.

    Examples:
        >>> normalize([10, 20, 30])
        [0.0, 0.5, 1.0]
        >>> normalize([5, 5, 5])
        [0.0, 0.0, 0.0]
        >>> normalize([])
        []
    """
    if not scores:
        return []
    m = max(scores)
    n = min(scores)
    if m == n:
        return [0.0] * len(scores)
    return [(x - n) / (m - n) for x in scores]

# Unit tests
def test_normalize():
    assert normalize([10, 20, 30]) == [0.0, 0.5, 1.0]
    assert normalize([5, 5, 5]) == [0.0, 0.0, 0.0]  # m == n case
    assert normalize([]) == []
    assert normalize([7]) == [0.0]  # single value, m == n case
    assert normalize([-1, 0, 1]) == [0.0, 0.5, 1.0]
    print("✅ All tests passed")
```



```python
# Unit tests
def test_normalize():
    assert normalize([10, 20, 30]) == [0.0, 0.5, 1.0]
    assert normalize([5, 5, 5]) == [0.0, 0.0, 0.0]  # m == n case
    assert normalize([]) == []
    assert normalize([7]) == [0.0]  # single value, m == n case
    assert normalize([-1, 0, 1]) == [0.0, 0.5, 1.0]
    print("✅ All tests passed")

if __name__ == "__main__":
    test_normalize()
```

Observation:

The normalize function is well-designed for telecom data analysis. It safely scales metrics to the [0, 1] range, returns an empty list for empty input, and avoids divide-by-zero errors by returning all zeros when all scores are equal. The Google-style docstring provides clear documentation, and the included unit tests confirm the function works correctly for both typical and edge cases.