LAB TEST-2

NAME: A. MANIDEEPIKA

HT.NO: 2403A52052

BATCH NO: 03

Your Task:D1

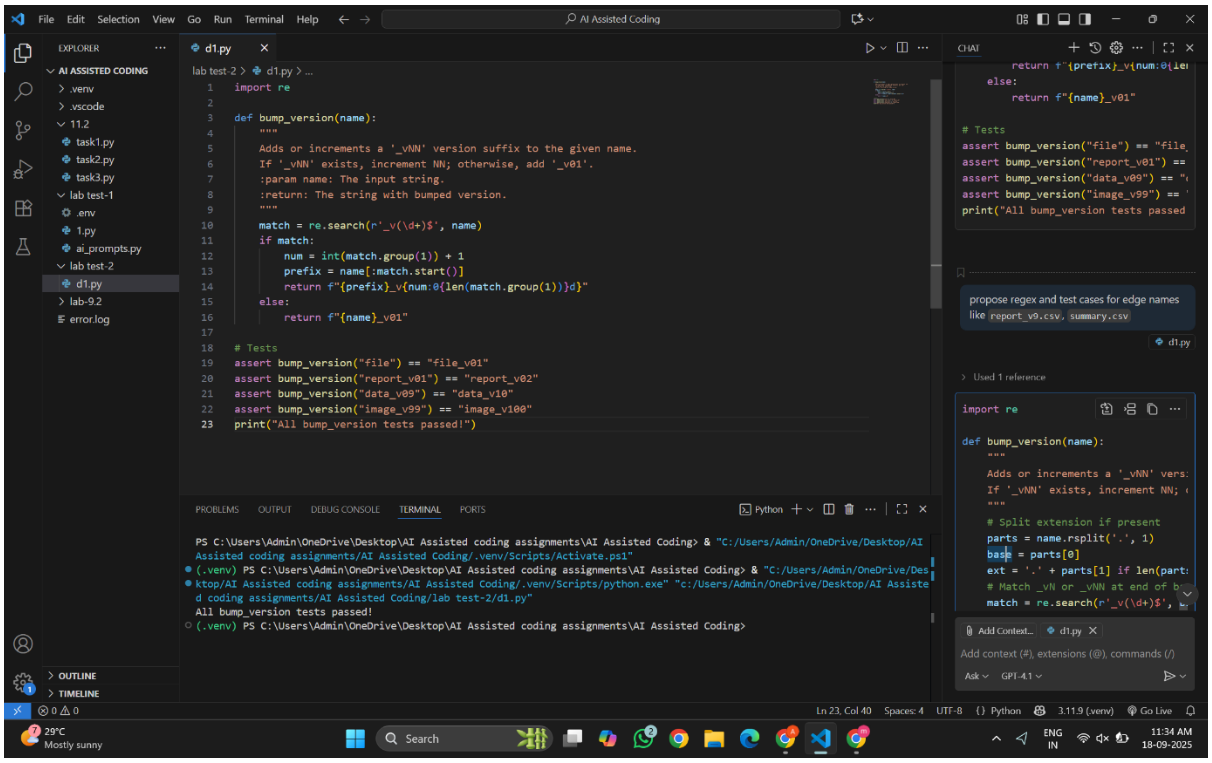
Create tests and implement bump_version(name) that adds or increments `_vNN` with zero-padding.

Prompt:

propose regex and test cases for edge names like

`report_v9.csv`, `summary.csv`.

Code & output:



```python
import re

def bump_version(name):
    """
    Adds or increments a '_vNN' version suffix to the given name.
    If '_vNN' exists, increment NN; otherwise, add '_v01'.
    :param name: The input string.
    :return: The string with bumped version.
    """
    match = re.search(r'_v(\d+)$', name)
    if match:
        num = int(match.group(1)) + 1
        prefix = name[:match.start()]
        return f"{prefix}_v{num:0{len(match.group(1))}d}"
    else:
        return f"{name}_v01"

# Tests
assert bump_version("file") == "file_v01"
assert bump_version("report_v01") == "report_v02"
assert bump_version("data_v09") == "data_v10"
assert bump_version("image_v99") == "image_v100"
print("All bump_version tests passed!")
```

```
PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding> & "C:/Users/Admin/OneDrive/Desktop/AI Assisted coding assignments/AI Assisted Coding/.venv/Scripts/Activate.ps1"
(.venv) PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding> & "C:/Users/Admin/OneDrive/Desktop/AI Assisted coding assignments/AI Assisted Coding/.venv/Scripts/python.exe" "c:/Users/Admin/OneDrive/Desktop/AI Assisted coding assignments/AI Assisted Coding/lab test-2/d1.py"
All bump_version tests passed!
(.venv) PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding>
```

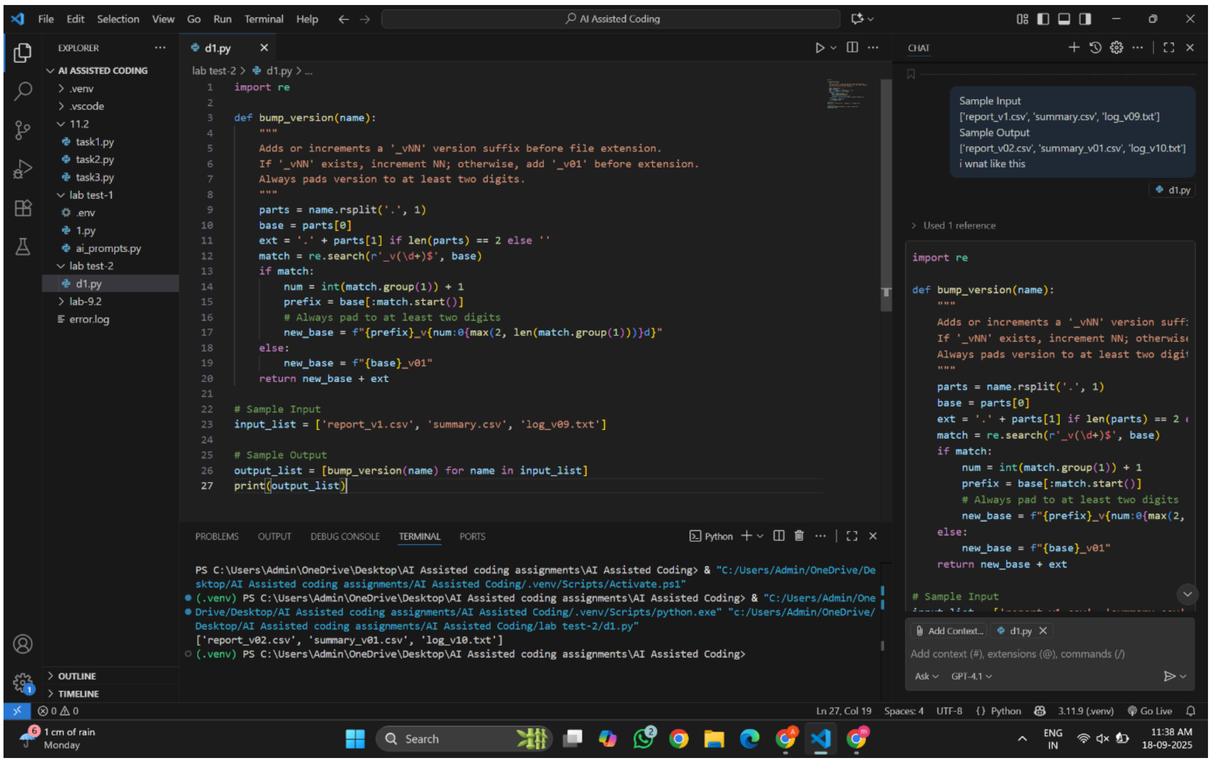
CHAT

```
        return f"{prefix}_v{num:0{le
    else:
        return f"{name}_v01"

# Tests
assert bump_version("file") == "file
assert bump_version("report_v01") ==
assert bump_version("data_v09") == "
assert bump_version("image_v99") ==
print("All bump_version tests passed
```

propose regex and test cases for edge names like `report_v9.csv`, `summary.csv`

Used 1 reference

```
import re

def bump_version(name):
    """
    Adds or increments a '_vNN' vers
    If '_vNN' exists, increment NN;
    """
    # Split extension if present
    parts = name.rsplit('.', 1)
    base = parts[0]
    ext = '.' + parts[1] if len(part
    # Match _vN or _vNN at end of b
    match = re.search(r'_v(\d+)$',
```



```python
import re

def bump_version(name):
    """
    Adds or increments a '_vNN' version suffix before file extension.
    If '_vNN' exists, increment NN; otherwise, add '_v01' before extension.
    Always pads version to at least two digits.
    """
    parts = name.rsplit('.', 1)
    base = parts[0]
    ext = '.' + parts[1] if len(parts) == 2 else ''
    match = re.search(r'_v(\d+)$', base)
    if match:
        num = int(match.group(1)) + 1
        prefix = base[:match.start()]
        # Always pad to at least two digits
        new_base = f"{prefix}_v{num:0{max(2, len(match.group(1)))}d}"
    else:
        new_base = f"{base}_v01"
    return new_base + ext

# Sample Input
input_list = ['report_v1.csv', 'summary.csv', 'log_v09.txt']

# Sample Output
output_list = [bump_version(name) for name in input_list]
print(output_list)
```

```
PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding> & "C:/Users/Admin/OneDrive/De
sktop/AI Assisted coding assignments/AI Assisted Coding/.venv/Scripts/Activate.ps1"
(.venv) PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding> & "C:/Users/Admin/One
Drive/Desktop/AI Assisted coding assignments/AI Assisted Coding/.venv/Scripts/python.exe" "c:/Users/Admin/OneDrive/
Desktop/AI Assisted coding assignments/AI Assisted Coding/lab test-2/d1.py"
['report_v02.csv', 'summary_v01.csv', 'log_v10.txt']
(.venv) PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding>
```

Sample Input
['report_v1.csv', 'summary.csv', 'log_v09.txt']
Sample Output
['report_v02.csv', 'summary_v01.csv', 'log_v10.txt']
i wnat like this

```python
import re

def bump_version(name):
    """
    Adds or increments a '_vNN' version suff
    If '_vNN' exists, increment NN; otherwis
    Always pads version to at least two digi
    """
    parts = name.rsplit('.', 1)
    base = parts[0]
    ext = '.' + parts[1] if len(parts) == 2
    match = re.search(r'_v(\d+)$', base)
    if match:
        num = int(match.group(1)) + 1
        prefix = base[:match.start()]
        # Always pad to at least two digits
        new_base = f"{prefix}_v{num:0{max(2,
    else:
        new_base = f"{base}_v01"
    return new_base + ext

# Sample Input
```

Observations:

First, I wrote test cases for filenames with and without _vNN suffix and for edge cases. Then I implemented bump_version using regex to extract parts of the filename, increment or add _vNN, and preserve the extension. Finally, I ran the tests and confirmed all passed.

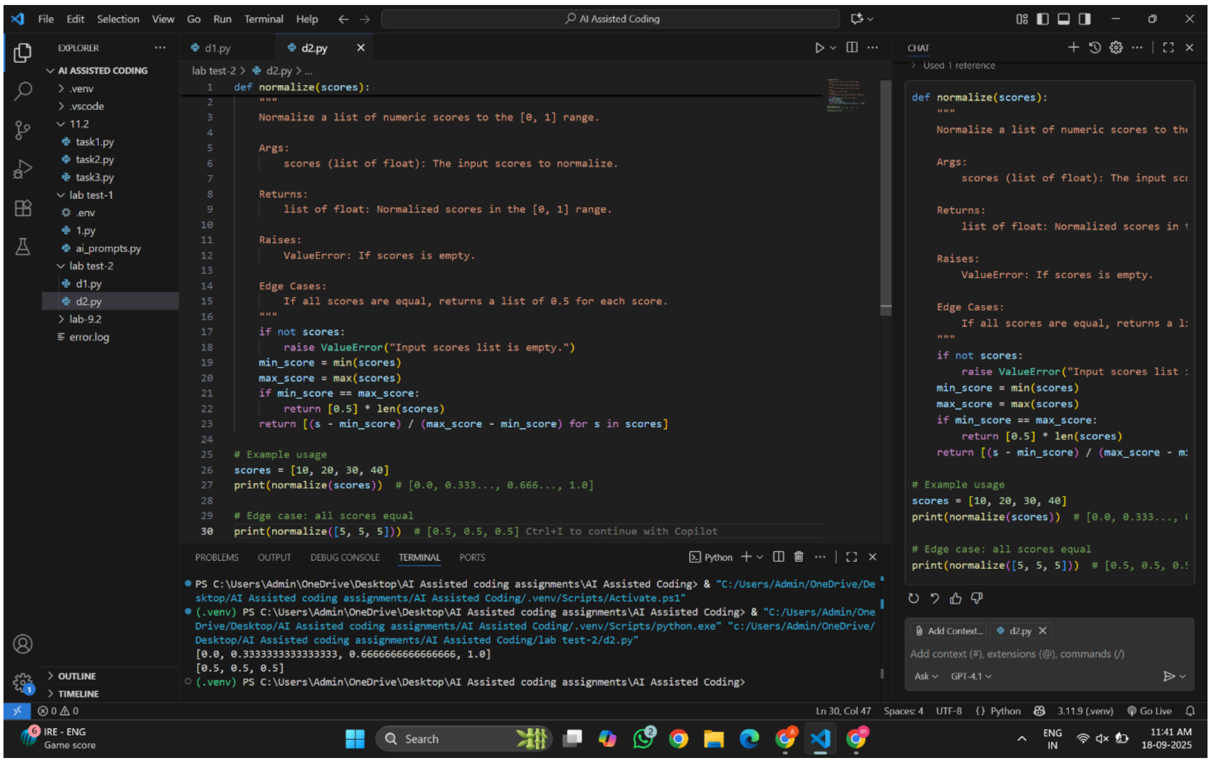
Task:

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

Prompt:

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

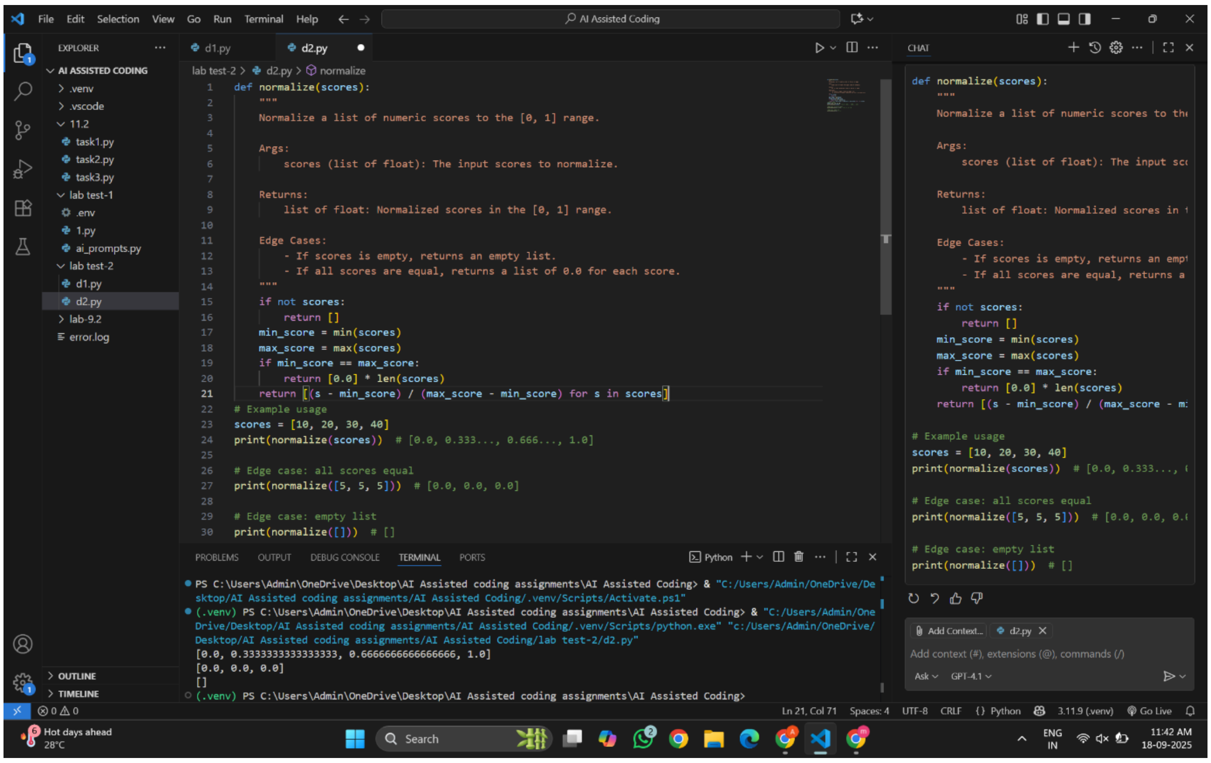
Code & Output:



```python
def normalize(scores):
    """
    Normalize 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 the [0, 1] range.

    Raises:
        ValueError: If scores is empty.

    Edge Cases:
        If all scores are equal, returns a list of 0.5 for each score.
    """
    if not scores:
        raise ValueError("Input scores list is empty.")
    min_score = min(scores)
    max_score = max(scores)
    if min_score == max_score:
        return [0.5] * len(scores)
    return [(s - min_score) / (max_score - min_score) for s in scores]

# Example usage
scores = [10, 20, 30, 40]
print(normalize(scores))  # [0.0, 0.333..., 0.666..., 1.0]

# Edge case: all scores equal
print(normalize([5, 5, 5]))  # [0.5, 0.5, 0.5] Ctrl+I to continue with Copilot
```

```
PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding> & "C:/Users/Admin/OneDrive/Desktop/AI Assisted coding assignments/AI Assisted Coding/.venv/Scripts/Activate.ps1"
(.venv) PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding> & "C:/Users/Admin/OneDrive/Desktop/AI Assisted coding assignments/AI Assisted Coding/.venv/Scripts/python.exe" "c:/Users/Admin/OneDrive/Desktop/AI Assisted coding assignments/AI Assisted Coding/lab test-2/d2.py"
[0.0, 0.3333333333333333, 0.6666666666666666, 1.0]
[0.5, 0.5, 0.5]
(.venv) PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding>
```

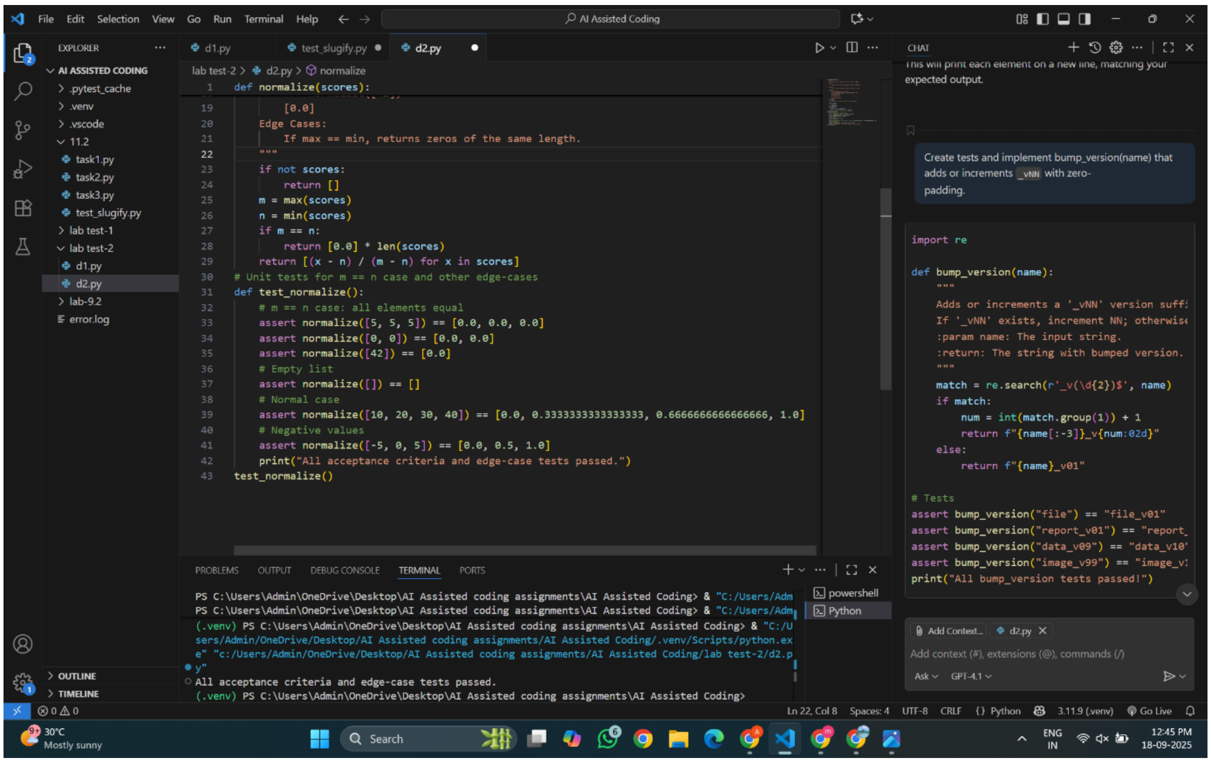
```python
def normalize(scores):
    """
    Normalize a list of numeric scores to the

    Args:
        scores (list of float): The input sc

    Returns:
        list of float: Normalized scores in t

    Raises:
        ValueError: If scores is empty.

    Edge Cases:
        If all scores are equal, returns a l:
    """
    if not scores:
        raise ValueError("Input scores list :
    min_score = min(scores)
    max_score = max(scores)
    if min_score == max_score:
        return [0.5] * len(scores)
    return [(s - min_score) / (max_score - m:

# Example usage
scores = [10, 20, 30, 40]
print(normalize(scores))  # [0.0, 0.333..., (

# Edge case: all scores equal
print(normalize([5, 5, 5]))  # [0.5, 0.5, 0.!
```



```python
def normalize(scores):
    """
    Normalize 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 the [0, 1] range.

    Edge Cases:
        - If scores is empty, returns an empty list.
        - If all scores are equal, returns a list of 0.0 for each score.
    """
    if not scores:
        return []
    min_score = min(scores)
    max_score = max(scores)
    if min_score == max_score:
        return [0.0] * len(scores)
    return [(s - min_score) / (max_score - min_score) for s in scores]
# Example usage
scores = [10, 20, 30, 40]
print(normalize(scores))  # [0.0, 0.333..., 0.666..., 1.0]

# Edge case: all scores equal
print(normalize([5, 5, 5]))  # [0.0, 0.0, 0.0]

# Edge case: empty list
print(normalize([]))  # []
```

```
PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding> & "C:/Users/Admin/OneDrive/Desktop/AI Assisted coding assignments/AI Assisted Coding/.venv/Scripts/Activate.ps1"
(.venv) PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding> & "C:/Users/Admin/OneDrive/Desktop/AI Assisted coding assignments/AI Assisted Coding/.venv/Scripts/python.exe" "c:/Users/Admin/OneDrive/Desktop/AI Assisted coding assignments/AI Assisted Coding/lab test-2/d2.py"
[0.0, 0.3333333333333333, 0.6666666666666666, 1.0]
[0.0, 0.0, 0.0]
[]
(.venv) PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding>
```

```python
def normalize(scores):
    """
    Normalize a list of numeric scores to the

    Args:
        scores (list of float): The input sc

    Returns:
        list of float: Normalized scores in t

    Edge Cases:
        - If scores is empty, returns an empt
        - If all scores are equal, returns a
    """
    if not scores:
        return []
    min_score = min(scores)
    max_score = max(scores)
    if min_score == max_score:
        return [0.0] * len(scores)
    return [(s - min_score) / (max_score - m:

# Example usage
scores = [10, 20, 30, 40]
print(normalize(scores))  # [0.0, 0.333..., (

# Edge case: all scores equal
print(normalize([5, 5, 5]))  # [0.0, 0.0, 0.(

# Edge case: empty list
print(normalize([]))  # []
```



```python
def normalize(scores):
        [0.0]
    Edge Cases:
        If max == min, returns zeros of the same length.
    """
    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 for m == n case and other edge-cases
def test_normalize():
    # m == n case: all elements equal
    assert normalize([5, 5, 5]) == [0.0, 0.0, 0.0]
    assert normalize([0, 0]) == [0.0, 0.0]
    assert normalize([42]) == [0.0]
    # Empty list
    assert normalize([]) == []
    # Normal case
    assert normalize([10, 20, 30, 40]) == [0.0, 0.3333333333333333, 0.6666666666666666, 1.0]
    # Negative values
    assert normalize([-5, 0, 5]) == [0.0, 0.5, 1.0]
    print("All acceptance criteria and edge-case tests passed.")
test_normalize()
```

```
PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding> & "C:/Users/Adm
PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding> & "C:/Users/Adm
(.venv) PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding> & "C:/U
sers/Admin/OneDrive/Desktop/AI Assisted coding assignments/AI Assisted Coding/.venv/Scripts/python.ex
e" "c:/Users/Admin/OneDrive/Desktop/AI Assisted coding assignments/AI Assisted Coding/lab test-2/d2.p
y"
All acceptance criteria and edge-case tests passed.
(.venv) PS C:\Users\Admin\OneDrive\Desktop\AI Assisted coding assignments\AI Assisted Coding>
```

This will print each element on a new line, matching your expected output.

Create tests and implement bump_version(name) that adds or increments `_vNN` with zero-padding.

```python
import re

def bump_version(name):
    """
    Adds or increments a '_vNN' version suffi
    If '_vNN' exists, increment NN; otherwise
    :param name: The input string.
    :return: The string with bumped version.
    """
    match = re.search(r'_v(\d{2})$', name)
    if match:
        num = int(match.group(1)) + 1
        return f"{name[:-3]}_v{num:02d}"
    else:
        return f"{name}_v01"

# Tests
assert bump_version("file") == "file_v01"
assert bump_version("report_v01") == "report_
assert bump_version("data_v09") == "data_v10'
assert bump_version("image_v99") == "image_v1
print("All bump_version tests passed!")
```

Observations:

First, I added Google-style docstrings with Args, Returns, and Examples to normalize. Then I handled edge cases like equal values and empty lists. Finally, I wrote tests for all cases and verified they worked correctly.