LAB TEST-2 : AI ASSISTED CODING

Name- K.Sidhartha Reddy[2403A51293]

D1 — TDD: bump\_version(name)

1) Prompt Used (AI Assistance)

Write a Python function `bump\_version(name: str) -> str` that:  
• Increments a `\_vNN` suffix (two digits, zero-padded) in the filename before the extension.  
• If none exists, adds `\_v01`.  
• Must preserve the original extension.  
• Examples:  
`report\_v1.csv → report\_v02.csv`,  
`summary.csv → summary\_v01.csv`,  
`log\_v09.txt → log\_v10.txt`.  
Provide regex logic and edge case tests.

2) solution.py

import re  
import os  
  
def bump\_version(name: str) -> str:  
 """  
 Increments or adds a \_vNN version suffix (two digits) before file extension.  
 """  
 base, ext = os.path.splitext(name)  
 match = re.search(r'(?:\_v)(\d+)$', base)  
 if match:  
 num = int(match.group(1)) + 1  
 new\_base = re.sub(r'\_v\d+$', f'\_v{num:02d}', base)  
 else:  
 new\_base = f"{base}\_v01"  
 return new\_base + ext

**Step-by-Step Debugging**

**Step 1 — Splitting filename and extension**

base, ext = os.path.splitext("report\_v1.csv")

**Result:**

* base = "report\_v1"
* ext = ".csv"

**Correct.**

Step 2 — Regex pattern

match = re.search(r'(?:\_v)(\d+)$', "report\_v1")

Captures digits after \_v at the end of the base.

Group(1) = "1"

Correct.

Also works for "log\_v09" → '09', "file\_v99" → '99'.

Step 3 — Increment and format

num = int("1") + 1 # → 2

f'\_v{num:02d}' # → '\_v02'

Zero-pads to 2 digits.

Also works: int('09')+1 → 10 → '\_v10'

Works even for 99 → 100 (becomes \_v100 which is fine)

Correct.

Step 4 — No version present

base = "summary"

match = None

new\_base = f"{base}\_v01" # summary\_v01

Correct.

Step 5 — Extension preservation

It simply returns new\_base + ext

So "summary" + ".csv" → "summary\_v01.csv"

Correct.

Real Test Outputs

| Input | Output |
| --- | --- |
| report\_v1.csv | report\_v02.csv |
| summary.csv | summary\_v01.csv |
| log\_v09.txt | log\_v10.txt |
| file\_v99.csv | file\_v100.csv |
| draft\_v7 | draft\_v08 |
| readme | readme\_v01 |

3) tests.py

import unittest  
from solution import bump\_version  
  
class TestBumpVersion(unittest.TestCase):  
 def test\_increment\_existing(self):  
 self.assertEqual(bump\_version("report\_v1.csv"), "report\_v02.csv")  
 self.assertEqual(bump\_version("log\_v09.txt"), "log\_v10.txt")  
 self.assertEqual(bump\_version("file\_v99.csv"), "file\_v100.csv")  
  
 def test\_add\_if\_missing(self):  
 self.assertEqual(bump\_version("summary.csv"), "summary\_v01.csv")  
 self.assertEqual(bump\_version("data.txt"), "data\_v01.txt")  
  
 def test\_handles\_no\_extension(self):  
 self.assertEqual(bump\_version("readme"), "readme\_v01")  
 self.assertEqual(bump\_version("draft\_v7"), "draft\_v08")  
  
 def test\_preserves\_complex\_names(self):  
 self.assertEqual(bump\_version("my.report\_v3.csv"), "my.report\_v04.csv")  
 self.assertEqual(bump\_version("my.summary.csv"), "my.summary\_v01.csv")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

4) Docstrings & Inline Comments

Included in solution.py — explains logic and function usage.

5) README.md

# bump\_version — [S18D1]  
  
## Overview  
This project implements a function `bump\_version(name)` that increments or appends a `\_vNN` version suffix to filenames before their extension.  
  
## Approach  
- Used regex to detect `\_vNN` at the end of the filename (before extension).  
- If found: increment number and zero-pad to 2 digits.  
- If not found: append `\_v01`.  
- Preserves the original extension.  
  
## Assumptions  
- Filename always has either no extension or a single valid extension.  
- `\_vNN` only considered if it appears right before the extension.  
  
## Complexity  
- Regex search and substitution: \*\*O(n)\*\* on filename length (negligible).  
  
## Running Tests  
```bash  
python -m unittest tests.py  
```

6) Test Output

test\_add\_if\_missing (\_\_main\_\_.TestBumpVersion.test\_add\_if\_missing) ... ok  
test\_handles\_no\_extension (\_\_main\_\_.TestBumpVersion.test\_handles\_no\_extension) ... ok  
test\_increment\_existing (\_\_main\_\_.TestBumpVersion.test\_increment\_existing) ... ok  
test\_preserves\_complex\_names (\_\_main\_\_.TestBumpVersion.test\_preserves\_complex\_names) ... ok  
  
----------------------------------------------------------------------  
Ran 4 tests in 0.001s  
  
OK

D2 -- Normalize Function Deliverables

1. AI Prompt Used

Add a Google-style docstring to this Python function. Ensure it handles edge cases: empty list, uniform values (where max == min). Include examples in the docstring. Also, suggest a safe handling of divide-by-zero when normalizing.

2. solution.py

def normalize(scores):  
 """  
 Normalize a list of numerical scores to the [0, 1] range.  
  
 If all values are the same, returns a list of zeros.  
 If the input list is empty, returns an empty list.  
  
 Args:  
 scores (list of float or int): List of numerical scores.  
  
 Returns:  
 list of float: Normalized scores in the range [0, 1].  
  
 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)  
  
 # Handle divide-by-zero when all elements are equal  
 if m == n:  
 return [0.0 for \_ in scores]  
  
 return [(x - n) / (m - n) for x in scores]

3. tests.py

import unittest  
from solution import normalize  
  
class TestNormalize(unittest.TestCase):  
 def test\_normal\_case(self):  
 self.assertEqual(normalize([10, 20, 30]), [0.0, 0.5, 1.0])  
  
 def test\_empty\_list(self):  
 self.assertEqual(normalize([]), [])  
  
 def test\_uniform\_list(self):  
 self.assertEqual(normalize([5, 5, 5]), [0.0, 0.0, 0.0])  
  
 def test\_single\_element(self):  
 self.assertEqual(normalize([42]), [0.0])  
  
 def test\_negative\_numbers(self):  
 self.assertEqual(normalize([-10, 0, 10]), [0.0, 0.5, 1.0])  
  
 def test\_floats(self):  
 self.assertEqual(normalize([0.1, 0.2, 0.3]), [0.0, 0.5, 1.0])  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

4. Docstrings & Inline Comments

Added Google-style docstring with Args, Returns, Examples.

Inline comment explains divide-by-zero handling for uniform lists.

5. README.md

# Normalize Function  
  
## Approach  
This function normalizes a list of numerical scores to the [0,1] range.  
It handles:  
- Empty lists (returns [])  
- Uniform lists where all values are the same (returns all zeros)  
  
## Assumptions  
- Input is always a list of numbers (int or float)  
- Division by zero is safely handled  
  
## Complexity  
- Time: O(n) for one pass to compute min/max and one pass to normalize  
- Space: O(n) for output list  
  
## Run Tests  
```bash  
python -m unittest tests.py

6. Debugging / Refactor Note

**Before:** Did not handle m == n → divide by zero if all elements equal. **After:** Added check for m == n → returns [0.0, 0.0, ...] safely.