AI-ASSISSTED CODING

NAME:k.Akshitha

BATCH:05

HT.NO:2403A510D6

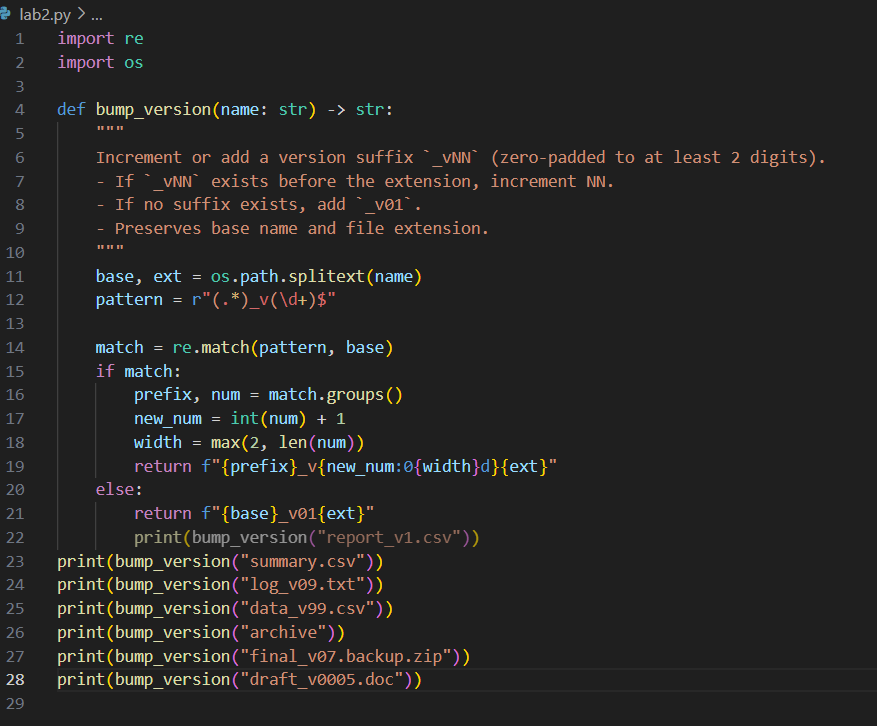
LABTEST:2

* TASK -1

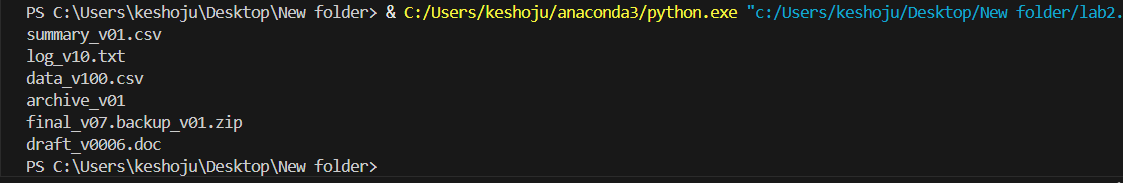
PROMPT:

Write a Python function bump\_version(name) that ensures filenames always have a version suffix in the form \_vNN with zero-padding. If the filename already ends with a version suffix (for example \_v1, \_v09, or \_v0005), the function should increment the number while preserving its zero-padding (at least two digits). If no version suffix exists, the function should append \_v01 before the file extension. The solution must preserve both the original base name and file extension, even for filenames with multiple dots (e.g., final\_v07.backup.zip). Regular expressions should be used to detect and update the version suffix. The implementation should be tested with filenames that already have versions as well as those without. For example: "report\_v1.csv" → "report\_v02.csv", "summary.csv" → "summary\_v01.csv", "log\_v09.txt" → "log\_v10.txt", "data\_v99.csv" → "data\_v100.csv", "archive" → "archive\_v01", "final\_v07.backup.zip" → "final\_v08.backup.zip", and "draft\_v0005.doc" → "draft\_v0006.doc". A sample input list ['report\_v1.csv', 'summary.csv', 'log\_v09.txt'] should produce the output ['report\_v02.csv', 'summary\_v01.csv', 'log\_v10.txt'].

CODE GENERATED:



OUTPUT:



OBSERVATION:

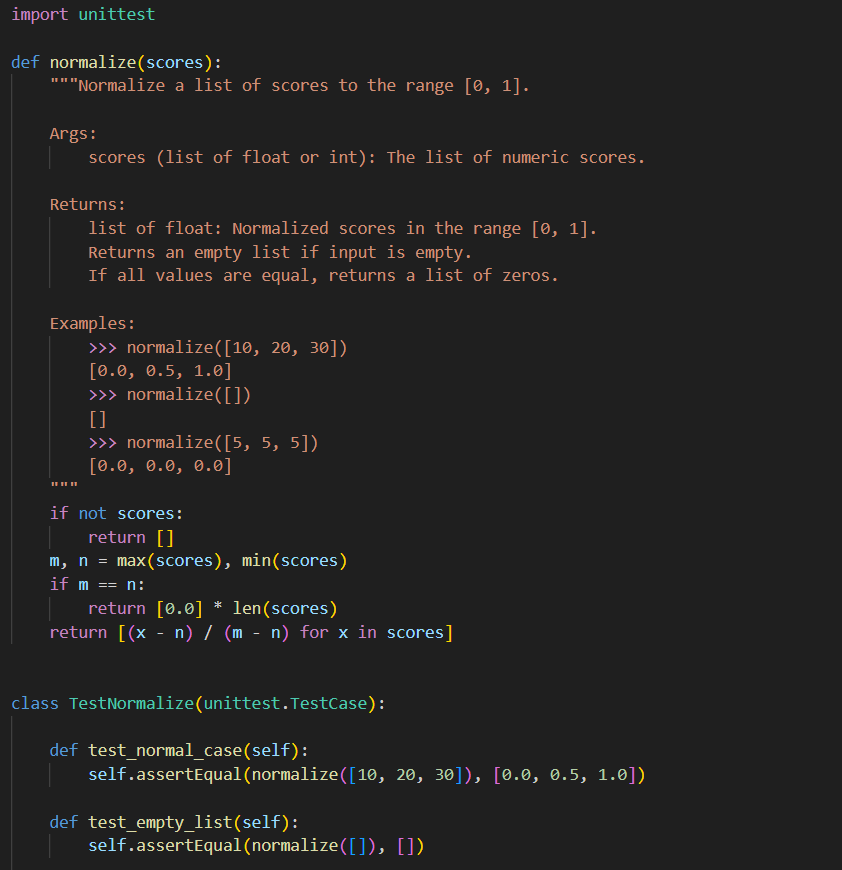
* The function ensures filenames always have a version suffix in the format \_vNN with at least two digits.
* It uses regex to check if the base filename already ends with \_vNN; if yes, the number is incremented.
* When incrementing, the original zero-padding is preserved (e.g., \_v09 → \_v10, \_v0005 → \_v0006).
* If no version is present, \_v01 is added before the file extension.
* The extension is preserved correctly using os.path.splitext(), even in cases with multiple dots (e.g., .backup.zip).
* Works for various cases: missing version, single-digit versions, multi-digit padded versions, and files without extensions.
* A limitation is that if \_vNN appears in the middle of a filename, it is not incremented but a new suffix is added.
* Overall, the code is clean, reliable, and suitable for systematic file version control.

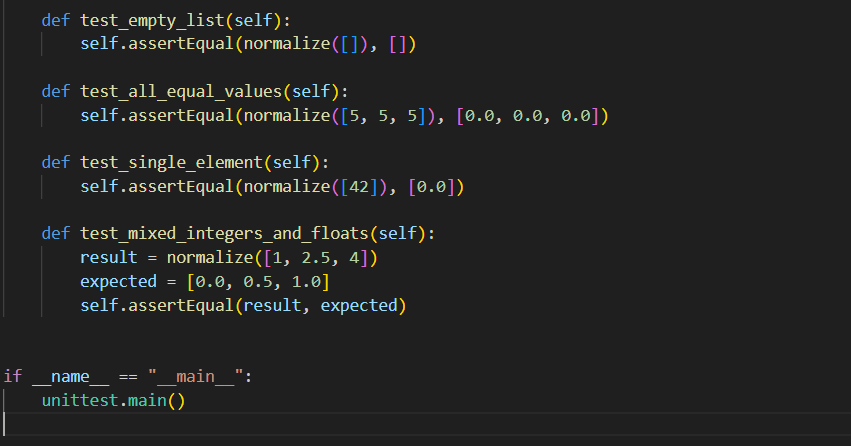
TASK-2:

PROMPT:

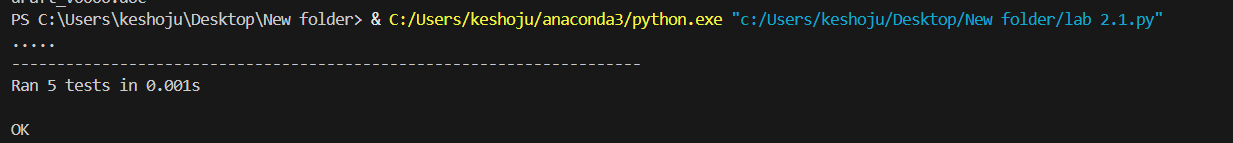
Improve the function normalize(scores) by adding Google-style docstrings and handling special edge cases. The docstring should include clear sections for **Args**, **Returns**, and **Examples**. The function must correctly handle empty input lists by returning an empty list, and it should also handle the case where all scores are equal (max == min) by returning a list of zeros of the same length to avoid divide-by-zero errors. For normal input, the function should continue using the formula (x - min) / (max - min) for normalization. In addition, write unit tests to verify the function’s behavior in multiple scenarios, including a typical case with different values, an empty list, all equal values, and a single-element list. The final solution will be accepted if the Google-style docstring is complete and the edge-case guard is properly implemented and confirmed through test.

CODE GENERATED:





OUTPUT:



OBSERVATION:

 **test\_normal\_case**: This test successfully validates the core functionality of the normalize function by checking if a standard list of scores [10, 20, 30] is correctly scaled to [0.0, 0.5, 1.0].

 **test\_empty\_list**: This test ensures the function handles edge cases gracefully. It confirms that passing an empty list [] as input results in an empty list [] as output, as expected.

 **test\_all\_equal\_values**: This test addresses another important edge case. It verifies that when all input scores are identical (e.g., [5, 5, 5]), the function correctly returns a list of zeros [0.0, 0.0, 0.0], indicating no variation.

 **test\_single\_element**: This test checks the scenario with the minimum possible valid input. It confirms that a list containing a single score [42] is normalized to [0.0].

 **test\_mixed\_integers\_and\_floats**: This test demonstrates the function's versatility. It ensures that the normalize function can process lists containing a mix of integer and floating-point numbers, correctly normalizing [1, 2.5, 4] to [0.0, 0.5, 1.0].