Python Week 3

Due by Sunday at the end of Week 3 (11:59 pm Central)

1 Directions and Policies

1.1 Collaboration Policy

You are welcome to ask your instructor for help, but this project must be completed **individually**. It is expected that whatever work you submit to Gradescope is your own.

1.2 What to submit

You will submit one file to Gradescope with the filename `pweek3.py`. The rest of this document will explain what should be in that file.

2 What is this assignment all about?

Google's search function is one of the best search tools, and it is based on the PageRank algorithm which is one of the most innovative algorithms ever developed. To get an idea behind the PageRank algorithm, and the basis of this assignment, watch the following video: PageRank Algorithm Explained.
In this assignment, you will perform the matrix calculations behind the PageRank algorithm for a general **Adjacency Matrix**.

3 Functions to write

3.1 `Normalize` function

Define a function called `Normalize` which will take as input a `n x n` NumPy array `A` representing an adjancency matrix containing only `0`'s and `1`'s (and all `0`'s on the main diagonal). Then `Normalize` will return a NumPy array where each column of `A` is divided by the sum of its elements. Note: the matrix `A` will be entered as integers `0` and `1`, but you should convert these entries to floats so the quotients are also floats.

For example, if $\texttt{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$, then `Normalize(A)` would return $\begin{bmatrix} 0. & 0.5 & 0. & 0. \\ 0.33333333 & 0. & 0. & 0.5 \\ 0.33333333 & 0. & 0. & 0.5 \\ 0.33333333 & 0.5 & 1. & 0. \end{bmatrix}$.

3.2 `PageRank` function

Define a function called `PageRank` which takes as input a "normalized" matrix (i.e. a `n x n` NumPy array which comes from the `Normalize` function) and an optional argument `iter` which is initially set to `100`.

Then the function will create a list `L` of "rank vectors" where the first rank vector (i.e. the initial rank vector) is a `n x 1` NumPy array of all `1/n` (as a float). For the example `A` above, the initial rank vector would be $\begin{bmatrix}0.25\\0.25\\0.25\\0.25\end{bmatrix}$.
The initial rank vector will be called `r0`. Then the `PageRank` function should return the list `L = [r0, r1, r2, ..., rT]` where `r(i+1)` $=$ `L` $\cdot$ `r(i)` for `i`≥ 0 and `len(L) = iter`.
For the example `A` above with `iter = 100`, the final element of `L` would be the rank vector $\begin{bmatrix}0.1199886212\\0.2399822739\\0.2399819140\\0.3999623047\end{bmatrix}$.

3.3 `SearchResults` function

Now that we have the results of the `PageRank` function, based on the final rank vector it yields, we need to decide which website is most likely to be visited. The `SearchResults` function will take as input a `n x 1` NumPy array of floats `r`. In practice, this would be the final element of `L` (i.e. the final rank vector) form the `PageRank` function.
Each entry of `r` will be assigned it's row number (using `0`-indexing) and a list of 2-tuples will be created with the row number and the associated float. For example, if `r` $= \begin{bmatrix}0.1199886212\\0.2399822739\\0.2399819140\\0.3999623047\end{bmatrix}$,
then the list of 2-tuples would be
`[(0, 0.1199886212), (1, 0.2399822739), (2, 0.2399819140), (3, 0.3999623047)]`
Then `SearchResults` will return this list, sorted by percentage from greatest to least. For this example, it would return the list
`[(3, 0.3999623047), (1, 0.2399822739), (2, 0.2399819140), (0, 0.1199886212)]`.
Note: You may want to look up the `argsort` function as part of the NumPy module for this function.

3.4 IMPORTANT NOTE

You should test your code by printing and checking the results. However, you should suppress all print statements, both inside and outside the functions, when you're submitting to the autograder. If you get a "parsing" error when submitting to the autograder, that could be one of the issues.
If you are ever unsure of an autograder error, don't hesitate to ask your instructor.
Finally, you may only use native Python modules or Python modules mentioned in this document. You may not use or import any other Python modules.

4 How will the assignment be graded?

4.1 Autograder: 90 points

The autograder tests your script and grades it based on execution and sample inputs. The following tests will be run:

1. Was the correct file submitted? (5 points)
2. Do all functions exist? (5 points)
3. Does `pweek3.py` contain proper Python syntax? (5 points)
4. Sample tests will be run on the functions. (75 points)

4.2 Manual review: 10 points

Your code will be checked manually for readability, proper comments, and anything the autograder cannot check. A good rule of thumb is one comment should be used for every 4 to 5 lines of code. If someone (or yourself a few years from now!) reads this code, they should be able to read your comments to get a feel for what you are doing.

4.3 Autograder workarounds

If we see that your program does not do what was requested in this document, but the error was not detected in the automated testing, a deduction may be given at this point. The scores assigned by the autograder will not be changed during manual review unless we discover some kind of intentional wrongdoing (such as an attempt to circumvent or reverse-engineer the autograder's operation).

4.4 Final word

It is **really** important to test locally
As you write your project, test it locally on your own computer. Do not try to test it with only the autograder. Make sure your programs run on your own computer before uploading them to Gradescope. It is much harder to debug a broken program based solely on reports you get from the autograder compared to working with your local Python interpreter.