Milestone 2: the index

In this milestone, you'll fill out the function create\_index(filenames, index, file\_titles) to read in a list of filenames and update an index of which terms show up in which files, as well as a file titles dictionary that maps the filename to its title.

Most search engines are built on top of an "inverted index", or "index" for short -- an index is a structure where, for each term, we have a list of documents that this term appears in. Consider the following examples of terms and documents they appear in:

* "burrito" appears in the text document "recipes.txt", "greatest eats.txt", "top 10 foods.txt", and "favs.txt"
* "sushi" appears in the text document "favs.txt" and "Japanese foods.txt"
* "samosa" appears in the text document "appetizers.txt"

The most natural structure to use in Python to represent such a mapping from terms to a list of documents is a dictionary. The above index could be represented as the following dictionary with keys that are strings and values that are lists of strings:

index = {

'burrito': ['recipes.txt', 'greatest eats.txt', 'top 10 foods.txt', 'favs.txt'],

'sushi': ['favs.txt', 'Japanese foods.txt'],

'samosa': ['appetizers.txt']

}

create\_index(filenames, index, file\_titles is passed the following information:

* filenames is a list of file names (strings) that you'll use to build an index.
* index is a dictionary representing the index that you'll build up. When your function is called, it will be passed an empty dictionary ({}) for index. Since dictionaries are mutable types, any changes your function makes to index will persist after the function completes; there's no need to return anything.
* file\_titles is a dictionary where the keys are file names (strings) and the values are the titles of the articles in each file (also strings). We'll explain the details of this parameter later in this description. When your function is called, it will be passed an empty dictionary ({}) for file\_titles and your function will add entries to this dictionary as appropriate.

You're responsible for populating the index and file\_titles dictionaries. First, we'll cover how to fill out the index dictionary, and then we'll discuss the file\_titles dictionary.

Milestone 2a: building an index

You'll build index based on the set of files specified in the parameter filenames. For each file in filenames, you should parse out all the **terms** in the file and add appropriate entries to the index you're building. Terms are defined as follows:

* Terms are *separated in text by spaces or newline (return) characters*.
* Terms should have all their letters converted to *lowercase*.
* Terms should have *all punctuation symbols stripped* off from their beginning and end. *Punctuation characters in the middle of a term are fine* and should not be removed.
* If a term *contains only punctuation marks* (in other words, if the term is an empty string after stripping punctuation and whitespace), it *should not be added to the index*.

The Python string library, which is already imported in the starter code (import string) provides a constant called string.punctuation which contains all the punctuation marks in Python. You can use this in conjunction with the .strip() function (which, with no parameters, just removes whitespace and newlines, but given a parameter will remove those characters) to remove punctuation from just the beginning and end of a string. Here's an example:

>>> raw = '$$j.lo!'

>>> term = raw.strip(string.punctuation)

>>> term

'j.lo'

Notice that the period in the middle of 'j.lo' isn't removed by strip() -- strip() doesn't remove punctuation in the middle of a string.

To make this all more concrete, let's say we have 2 files to index, doc1.txt and doc2.txt. First, let's look at doc1.txt:

\*We\* are 100,000

STRONG! $$

In parsing this file, we should produce the following terms:

* The string '\*We\*' should be converted to the term 'We'
* The string 'are' should be converted to the term 'are'
* The string '100,000' should be converted to the term '100,000' (note the comma inside 100,000 is fine and should not be removed)
* The string 'STRONG!' should be converted to the term 'strong'
* The string '$$' should be ignored (not included in index) since a string of only punctuation is not a term.

As you produce these terms, you should add them appropriately into the index. If we started with an empty index before processing the file doc1.txt, after processing doc1.txt the index should look like this:

{

'we': ['doc1.txt'],

'are': ['doc1.txt'],

'100,000': ['doc1.txt'],

'strong': ['doc1.txt']

}

Let's say that after processing doc1.txt, we process another file, doc2.txt, which is shown below:

Strong, you are!

--Yoda--

We should appropriately update the index we had before with the terms in doc2.txt to result in index as below:

{

'we': ['doc1.txt'],

'are': ['doc1.txt', 'doc2.txt'],

'100,000': ['doc1.txt'],

'strong': ['doc1.txt', 'doc2.txt'],

'you': ['doc2.txt'],

'yoda': ['doc2.txt']

}

Notice that in cases where a term in doc2.txt was already in the index (such as 'are' and 'strong), the list of documents that contain the term was *expanded* to include the file doc2.txt. In cases where a term in doc2.txt wasn't previously in the index (such as 'you' and 'yoda'), a new entry is added to indicate that the given term appeared in doc2.txt.

In order to test that you're building index correctly, you can comment out the lines of the doctests that check the dictionary file\_titles for now, and press the Check button.

Milestone 2b: building file\_titles

Your code should also populate a dictionary called file\_titles which maps a filename to the title of the article. In the news article data we provide in this assignment, the first line of each file is a title for the article in that file. Here's an example of the first portion of two of the actual files (001.txt and 002.txt , respectively) in the BBC News article data that we provide for you in the assignment:

**001.txt**

Broadband steams ahead in the US

More and more Americans are joining the internet's fast lane, according to official figures.

...

**002.txt**

EA to take on film and TV giants

Video game giant Electronic Arts (EA) says it wants to become the biggest entertainment firm in the world.

...

If file\_titles starts as an empty dictionary (which is the case when create\_index() is called), after we process 001.txt and 002.txt, the file\_titles dictionary should look like this:

{

'001.txt': 'Broadband steams ahead in the US',

'002.txt': 'EA to take on film and TV giants',

}

Important notes:

* Make sure to remove the "newline" character at the end of the title.
* The terms in the title line for each file *should still be included in the index*, along with the rest of the document!

Doctests for the create\_index() function are provided to check that you've populated both dictionaries correctly; press Check to run them.

You can also run your create\_index function by running the searchengine.py program and specifying the directory name for a set of text files you'd like to index. We provide two datasets for you to test your code -- the first is in a directory named small, which includes 3 very short text files, making the size of the resulting index small enough to inspect manually. You can run your program by pressing Run or running python searchengine.py small in your terminal. We've provided a main() function that calls your create\_index() function and prints the resulting index and file\_titles (note that the output below is produced on a Mac, where file paths use the forward-slash character '/'; on a PC you might see a double backslash '\\' in the file paths instead). Here's what the result of running your code on small should be (don't worry if things aren't in the same order; also, we've added line breaks to make the output more readable):

Index:

{'file3': ['small/3.txt'],

'title': ['small/3.txt', 'small/2.txt', 'small/1.txt'],

'apple': ['small/3.txt', 'small/2.txt', 'small/1.txt'],

'ball': ['small/3.txt', 'small/1.txt'],

'carrot': ['small/3.txt', 'small/2.txt', 'small/2.txt', 'small/2.txt'],

'gerbil': ['small/3.txt'],

'hamster': ['small/3.txt'],

'iguana': ['small/3.txt'],

'lizard': ['small/3.txt'],

'file2': ['small/2.txt'],

'dog': ['small/2.txt', 'small/1.txt'],

'file1': ['small/1.txt'],

'elephant': ['small/1.txt'],

'frog': ['small/1.txt']}

File names -> document titles:

{'small/3.txt': '\*\* File3 title \*\*',

'small/2.txt': '\*\* File2 title \*\*',

'small/1.txt': '\*\* File1 title \*\*'}

When you think you've gotten your program working well on the small dataset, try it out on the BBC News article dataset with python searchengine.py bbcnews -- the output produced here will be too large to verify manually, but running on such a large dataset is a good way to see if your program crashes on any cases that might have been missed with the small dataset.