# 6.0001/6.00 Fall 2018

# Problem Set 3

**Handed out:** Thursday, September 20th, 2018
**Due: Friday, September 28th, 2018 at 4:59PM**

This problem set will introduce you to the topic of dictionaries. You can read more about dictionaries in the textbook and in Lecture 6 on MITx. Although this handout is long, most of the information is there to provide you with context, useful examples, and hints, so be sure to read *carefully*.

**Note on Collaboration**:
You may work with other students. However, each student should write up and hand in his or her assignment separately. *Be sure to indicate with whom you have worked in the comments of your submission.*

---

## Problem 0: Getting Started

You will implement a program to detect similarity between two text documents. Don't be intimidated by this problem - we'll give you all the information you need to complete this p-set! We will guide you through the creation of helper functions before you implement the complete program.

## File Set-Up

Download the file 1_ps3.zip and extract all files to the same directory. The files included are: **document_distance.py**, **test_ps3_student.py**, **test1a.txt, test1b.txt**, **test2a.txt, test2b.txt, test3a.txt, test3b.txt, hello_world.txt, hello_friends.txt**.
You will edit ONLY **document_distance.py**.

## Document Distance Overview

Given two documents, you will calculate a score between 0 and 1 that will tell you how similar they are. If the documents are the same, they will get a score of 1. If the documents are completely different, they will get a score of 0. You will calculate the score in two different ways, and observe whether one works better than the other. The first way will use single word frequencies in the two texts. The second way will use bigram (sequence of two adjacent words) frequencies in the two texts.

---

## Problem 1: Word Frequency

We have provided a function called ***load_text*** to read a text file and output all the text in the file into a string. This function takes in a variable called filename, which is a string of the filename you want to load, including the extension. It removes all punctuation, and saves the text as a string. Do not modify this function.

Here's an example usage:

```
>> text = load_text("hello_world.txt")
>> text
'hello world hello'
```

Let's start with calculating the word frequency of each unique word in the text. The goal is to return a dictionary with a distinct word in the text as the key, and how many times the word occurs in the text as the value. For example, in `hello_world.txt`, you would get the following output:

```
>> {'hello': 2, 'world': 1}
```

Implement ***get_frequencies*** in **document_distance.py** using the above instructions and the docstring provided. *One important point to keep in mind:* this function takes one parameter, `words`, whose type can be a string or a list. If words is a string, your code should convert the string to a list of words before creating the frequency dictionary.

Note: You can assume that the text documents we provide will not have extra whitespaces (i.e. there are no tabs or newlines or extra spaces between words).

## Problem 2: Find Bigrams

Now, instead of counting how many times single words occur in documents, we will look at the frequency of bigrams in the two texts. A bigram is a sequence of two adjacent words in a text. For example, if the text is "problem set number three", the bigrams would be "problem set", "set number", and "number three". You will be implementing a series of functions that allows you to scan through a text. You'll be using a two-word window that moves from the beginning to the end of the text.

You will extract bigrams from the input text. Implement ***find_bigrams*** as follows and fill in the function in document_distance.py. Hint: Create a list of individual words and merge adjacent words to make a bigram string.

## Problem 3: Similarity

Now it's time to calculate similarity! Complete the function ***calculate_similarity*** in **document_distance.py** with the following conditions. This function can be used with the word frequency dictionary or the bigram frequency dictionary created by the function ***get_frequencies***.

The similarity will be a calculated by taking the *difference in text frequencies* divided by the *total frequencies*. The procedure below is for the word frequency dictionary. The same procedure is applied for the bigram frequency dictionary. Assume you have two frequency dictionaries, dict1 and dict2, one for each of the texts.

1. The *difference in text frequencies* = **DIFF** is the sum of the values from each of the following three scenarios:
    a. If a word occurs in dict1 and dict2 then get the absolute value of the difference in frequencies
    b. If a word occurs only in dict1 then take the frequency from dict1
    c. If a word occurs only in dict2 then take the frequency from dict2
2. The total frequencies = **ALL** is calculated by summing all frequencies in both dict1 and dict2.
3. Return **1- DIFF/ALL** rounded to **2 decimal places**.
   For example, if you run **document_distance.py**, the similarity between the files **hello_world.txt** and **hello_friends.txt** is
    - 0.4 using words
    - 0.0 using bigrams

## Problem 4: Most Frequent Words

In this final part, you will find out which words occur the most. You'll count how many times every words occurs, combined across both texts. You'll create and return a list of the most frequent word. If more than one word is tied to occur the most times, return an alphabetically ordered list of all these words. Implement the function **get_most_frequent_words** in **document_distance.py**.

For example, given:
`freq1 = {"hello":5, "world":1}` and
`freq2 = {"hello":1, "world":5}` then
`get_most_frequent_words(freq1, freq2)` returns `["hello", "world"]`.

When you are done, run the tester file **test_ps3_student.py** to check our test cases.

---

## Hand-in Procedure

### 1. Naming Files

Save your solutions with the original file name: document_distance.py. **Do not ignore this step or save your files with a different name!**

### 2. Time and Collaboration Info

At the start of your file, in a comment, write down the number of hours (roughly) you spent on the problems in that part, and the names of the people with whom you collaborated.

For example:

```
    # Problem Set 3
    # Name: Jane Lee
    # Collaborators: John Doe
    # Time Spent: 3:30
    # Late Days Used: 1 (only if you are using any)
    # … your code goes here …
```

## 3. Submit

To submit **document_distance.py**, upload it to the problem set website linked from Stellar. You may upload new versions of each file until the 4:59 PM deadline, but anything uploaded after that time will be counted towards your late days, if you have any remaining. If you have no remaining late days, you will receive no credit for a late submission.

After you submit, please be sure to view your submitted file and double-check you submitted the right thing.