

## I210: Information Infrastructure I – Mastery Check 4

*When you have completed this Mastery Check, you should:*

- 1. Submit your solution to Canvas under "Mastery Check 4"*
- 2. Upload a copy of your work to OneDrive as a backup in case something goes wrong with Canvas.*

Make sure you click through to the final "Submit" button on Canvas. You are responsible for making sure your work is submitted **by the end of the lab**, so it is strongly advised that you verify the submission before leaving. **We CANNOT accept late submissions.**

As a reminder, you may use anything on the Canvas section for **I210 only**, as well as the course textbook. You may use any notes (physical or online) taken for **I210 only**. You may work on an STC machine or on your personal laptop.

**You may NOT use the Internet except:**

- the Canvas sections for I210
- any group coding or note spaces you've set up for I210.

*Using code you found online from outside of this class or code that you or your group did not write that is not from the book or slides is likely to constitute academic misconduct.*

## INFO I210 Mastery Check 4: Python Word Frequency Analysis

### Objective:

To develop a Python program that reads a text file, calculates the frequency of each word, and outputs the results in a meaningful way. Through this project, you will gain hands-on experience with concepts including file handling, dictionaries, loops, and functions.

### Project Description:

The project will be divided into several stages, each designed to reinforce key programming concepts while contributing to the overall goal of creating a word frequency analysis tool.

### Stage 1: File Handling in Python

Objective: Complete the **read\_file(file\_name)** function

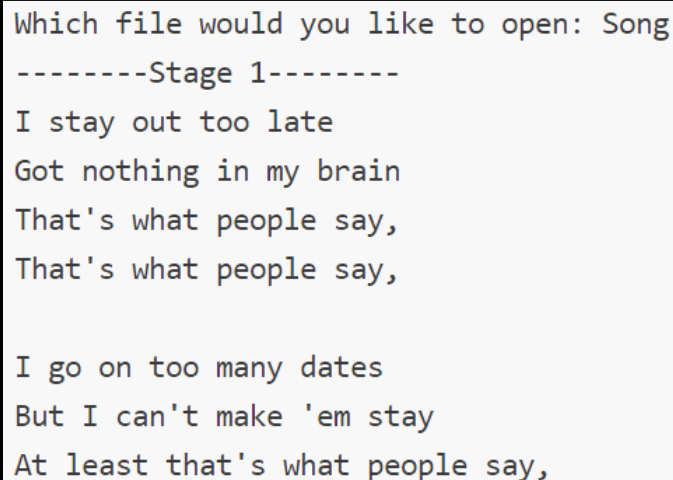
Open and read the file whose name is provided in the file\_name parameter.

Return the contents of the file as a single STRING. When the text is displayed, you should see the contents of the original text, with spacing and formatting preserved.

Optional bonus point: Allow the user to enter only the file name only (no .txt extension) and have the program automatically account for the .txt in the filename.

<Please note, if you prefer to have the filename hardcoded and not utilize a user input for the filename, no points will be deducted for this modification>

When you run the program after finishing Stage 1, it should look something like this:

A screenshot of a terminal window showing the output of a Python program. The text is as follows:

```
Which file would you like to open: Song
-----Stage 1-----
I stay out too late
Got nothing in my brain
That's what people say,
That's what people say,

I go on too many dates
But I can't make 'em stay
At least that's what people say,
```

\*This image was truncated/cropped. Additional text will display in your output.

## Stage 2: Processing Text Data

Objective: Complete the **get\_words(text)** function

The “text” parameter provides the string you created in Stage 1. Your first job is to split that string into a list of individual words. If you get stuck on how to do this, think about what separates individual words in the text.

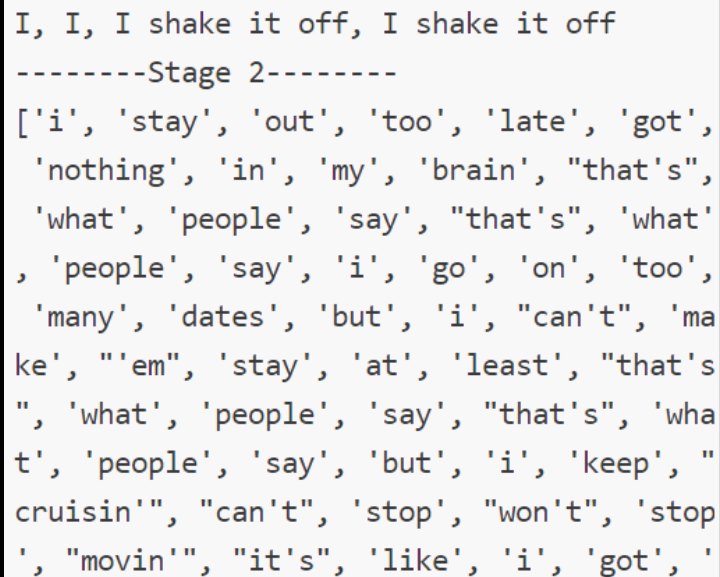
The list of words should then be “cleaned” to remove items not suitable for our use. We need you to remove punctuation from the word. We also need you to convert the words to lowercase.

The starter file contains more detailed instructions for working through this stage.

```
def get_words(text):
    punctuation = "!#$%&\\()*+,-./:;<=>?@[\\]^_`{|}~" + '\n'
    # split the text into words
    # create a list to store the cleaned words
    # for each word in the text that was split into words,
    # create a blank string called "clean_word"
    # loop over each character of the original word
    #     # if that character is not contained in the "punctuation" list defined above,
    #     # add it to clean_word
    # append the lower cases version of the clean word to the list of cleaned words
    # return the list of cleaned words
```

This should return the cleaned list of words.

When you run the program after finishing Stage 2, it should look something like this:



```
I, I, I shake it off, I shake it off
-----Stage 2-----
['i', 'stay', 'out', 'too', 'late', 'got',
 'nothing', 'in', 'my', 'brain', "that's",
 'what', 'people', 'say', "that's", 'what',
 'people', 'say', 'i', 'go', 'on', 'too',
 'many', 'dates', 'but', 'i', "can't", 'ma
ke', "'em", 'stay', 'at', 'least', "that's",
 'what', 'people', 'say', "that's", 'wha
t', 'people', 'say', 'but', 'i', 'keep', "
cruisin'", "can't", 'stop', "won't", 'stop
', "movin'", "it's", 'like', 'i', 'got', '
```

\*This image was truncated/cropped. Additional text should display in your output.

### Stage 3: Counting Words with Dictionaries

Objective: Complete the **count\_words(words)** function

The “words” parameter is a list of words.

Create and use a dictionary to tally up the number of times each word appears in the text. If you get stuck on how to do this, think about our previous coding example “Counting Coins”. The keys will be the individual words, and the values will be the number of times the word appears in the text.

Iterate over words in the text to update the dictionary with word counts.

Return the dictionary with the word tallies.

When you run the program after finishing Stage 3, it should look something like this:

```
ff', 'i', 'shake', 'it', 'off', '']  
-----Stage 3-----  
{ 'i': 62, 'stay': 2, 'out': 2, 'too': 2, 'late': 1, 'got': 4, 'nothing': 1, 'in': 3, 'my': 8, 'brain': 1, "that's": 8, 'what': 8, 'people': 4, 'say': 4, 'go': 3, 'on': 5, 'many': 1, 'dates': 1, 'but': 4, "can't": 3, 'make': 2, "'em": 1, 'at': 1, 'least': 1, 'keep': 2, "cruisin'": 2, 'stop': 4, "won't": 3, "movin'": 1, "it's": 4, 'like': 3, 'this': 3, 'music': 2, 'mind': 2, "sayin'": 2, 'gonna': 22, 'be': 2, 'alright': 2, "'cause": 3, 'the': 15, 'players': 3, 'play': 15, 'and': 14, 'haters': 4, 'hate': 16, 'baby': 7, "i'm": 9, 'just': 8, 'shake': 70, 'it': 36, 'off': 36, 'hoo': 24,
```

\*This image was truncated/cropped. Additional text should display in your output.

## Stage 4: Sorting and Displaying Results

Objective: Complete the **display\_results(word\_counts)** function

The “word\_counts” parameter is a dictionary containing frequencies for each word.

Turn the dictionary into a list containing the keys and values from the dictionary. If you get stuck on how to do this, you may want to reference the slidedeck for Lesson 17.

If you returned results now, you might see something similar to this:

```
-----Stage 4-----
[('i', 62), ('stay', 2), ('out', 2), ('too', 2), ('late', 1), ('got', 4), ('nothing', 1), ('in', 3),
 ('my', 8), ('brain', 1), ('that's', 8), ('what', 8), ('people', 4), ('say', 4), ('go', 3), ('on', 5),
 ('many', 1), ('dates', 1), ('but', 4), ('can't', 3), ('make', 2), ('em', 1), ('at', 1), ('least',
 1), ('keep', 2), ('cruisin'', 2), ('stop', 4), ('won't', 3), ('movin'', 1), ('it's', 4), ('like',
 3), ('this', 3), ('music', 2), ('mind', 2), ('sayin'', 2), ('gonna', 22), ('be', 2), ('alright', 2),
```

\*This image was truncated/cropped. Additional text should display in your output.

Next, Sort the list by word frequency (the values in the dictionary) in descending order. For the sorting, you can use Selection Sort, Insertion Sort, key sorting, or lambda sorting.

If you returned results now, you might see something similar to this:

```
[('shake', 70), ('i', 62), ('it', 36), ('off', 36), ('hoo', 24), ('gonna', 22), ('fake', 18), ('hat
e', 16), ('the', 15), ('play', 15), ('break', 15), ('and', 14), ('i'm', 9), ('my', 8), ('that's', 8),
 ('what', 8), ('just', 8), ('baby', 7), ('on', 5), ('got', 4), ('people', 4), ('say', 4), ('but',
 4), ('stop', 4), ('it's', 4), ('haters', 4), ('they', 4), ('don't', 4), ('in', 3), ('go', 3), ('ca
n't', 3), ('won't', 3), ('like', 3), ('this', 3), ('cause', 3), ('players', 3), ('heartbreakers',
 3), ('fakers', 3), ('hey', 3), ('you', 3), ('to', 3), ('yeah', 3), ('stay', 2), ('out', 2), ('too',
```

\*This image was truncated/cropped. Additional text should display in your output.

If you aren't getting your output to be in descending order, you may need to reverse your list items.

Finally, Print the 20 most common words and their frequencies.

If you printed, you might see something similar to this:

\*This image was truncated/cropped.

Additional text should display in your output.

```
-----Stage 4-----
('shake', 70)
('i', 62)
('it', 36)
('off', 36)
('fake', 18)
('hate', 16)
('play', 15)
('break', 15)
('the', 15)
('and', 14)
('i'm', 9)
('what', 8)
```

Optional bonus point: Alter the output of the top 20 words to look better. Can you get it to look like this? Or output with spacing similar to columns? Output into a table?

```
-----Stage 4-----  
Top 20 Words  
shake: 70  
i: 62  
it: 36  
off: 36  
hoo: 24  
gonna: 22  
fake: 18  
hate: 16
```

```
-----Stage 4-----  
Rank  Word      Frequency  
-----  
1     shake      70  
2     i          62  
3     it         36  
4     off        36  
5     hoo        24  
6     gonna      22  
7     fake       18  
8     hate       16
```

↓ BONUS below ↓

Part 0: Check out optional bonus points in Stage 1 and Stage 4 above.

Part 1: Write code to write the original lyrics from Part 1 into a new txt file called 'my\_text.txt'. Include the code in your py file and also turn in 'my\_text.txt' file with your submission in Canvas.

Part 2: Append a new ending/verse to the original lyrics from Part 1 into the Song.txt file. Include the code in your py file and also turn in 'Song.txt' containing your appended lyrics with your submission in Canvas.

Part 3: Create a **generate\_random\_text()** function:

1. The function should take a list of words and a number of words as input
  - a. The number of words *could* be hardcoded, or for additional points, allow the user to set the number of words for the randomly generated text
2. Randomize the words in the list when selecting them for this step.
3. It should choose words from the list of words from the project and combine them into one string.
4. The number of words provided is the input to the function. Reminder: the input may or may not be stored as a numeric value.
5. Return the string with the randomly generated text and print it out for the user to see.
6. You can stop here or go on...
7. Create a **save\_user\_text()** function that takes the parameter of the random string output from "generate\_random\_text" and writes the user-created random text to a file called "User\_Created\_Text.txt"

Reminder: Maximum Score: 110pts