Drill Problem #1

Function Name: tooManyTLAs

Inputs:

1. (*char*) A string containing the name of the file

Outputs:

1. (*char*) A phrase made from the acronym of the input file

Function Description:

During a long, boring, not-CS-1371 lecture you and your best friend realize you'd much rather be chatting, but your professor is guaranteed to call you out for talking if you try! Passing notes is too risky, since the class size is small and you two sit in the front row. Even worse, you can't just send Facebook messages to each other since you have to turn in your typed notes at the end of class, and your professor goes way too fast for you to do both separately! Instead, you and your friend decide to hide your messages as an acronym made out of the first letter of every line in your notes. To speed up the process, you decide to write a MATLAB function that can go through your friend's notes to pick out the message and display it as a string.

Write a function that takes in an input ".txt" file and pulls out the first letter of every line and combines these letters into a single string. The first two lines will be a title and line containing a single space that should be excluded from the final string.

Notes:

- There will be multiple words in each output phrase. Each line that marks word breaks in the input text file will contain a single space (so that there are no empty lines).
- Remember to not include the first two lines before the message starts.

Drill Problem #2

Function Name: detention

Inputs:

1. (*char*) A string of the name of a .txt file
2. (*char*) A string of the exact sentence that needs to be written

Outputs:

1. (*char*) A string of a sentence indicating whether each line of the .txt file matches the exact sentence from the second input

Function Description:

"Writing Lines" is a classic punishment for naughty students that wind up in detention at school. The teacher writes a line on the board, usually describing what the student did wrong, and the misbehaving student is expected to write this line over and over again a certain number of times until the student may be dismissed.

Your task is to write a MATLAB function that can check all the lines, which are recorded in a ".txt" file, that the student turns in and determine whether the student wrote every single line properly or if they messed up their punishment and need to repeat it.

You will need to iterate through every single line of the input ".txt" file and compare each line to the exact sentence given as the second input. The output of your function should be a sentence stored in a string that indicates the results.

If every single line matches the second input exactly, your result should read:

`'Good job. You are free to go!'`

If one or more of the lines does not match the second input exactly, your result should read:

`'Not so fast… Start all over!'`

Notes:

- The number of lines contained inside the ".txt" file is irrelevant.
- Your string output must match the solution file exactly in order to receive credit.

Hints:

- You may find the `fgetl()` and `all()` functions useful.

Drill Problem #3

Function Name: phoneNumbers

Inputs:

1. (*char*) A string containing the name of the file

Outputs:

1. (*double*) The total count of phone numbers collected

Output Files:

1. A ".txt" file containing the names and valid phone numbers obtained

Function Description:

Yesterday was your 21st birthday. Naturally, your friends took you to dinner and with that came a few challenges for the night, one of which was to collect 21 phone numbers. Now that it is the next morning, you have a long list of numbers; however, some of them seem a little off, so you want to go through and see which ones are actually valid. Who has the time to look through 21 numbers, though? Being the MATLAB genius that you are, you decided to have MATLAB sort through every phone number you obtained, decide if it is a valid phone number, and then output a new file containing only the names and numbers of the valid phone numbers.

You will be given an input ".txt" file with an unknown number of lines, because you may not have collected all 21 numbers. Each line is guaranteed to be in the format <name>; (<number>), including parentheses. For the number to be valid, it must be in the format XXX-XXX-XXXX. If there are symbols or letters within the number, then it is invalid. Further, if the digits are not separated by hyphens then the number is invalid as well. If you determine that the number is valid, it should be printed to an output file named the same as the input file but with '_updated' appended to the end of the filename (for example, `'phoneList90210.txt'` should become `'phoneList90210_updated.txt'`). The line printed to the output file should be exactly the same as the line from the input file, as in <name>; (<number>). Finally, your code should keep track of the total number of valid numbers collected and output that number.

Notes:

- There should be no extraneous lines or spaces in your output file.
- You will **not** have an extra space at the end of your file.
- You are guaranteed to have at least one correct number in the input ".txt" file.
- Do not forget to close your files in your code.

Drill Problem #4

Function Name: fauxHipster

Inputs:

1. (*char*) A string containing the filename of a conversation's ".txt" file
2. (*char*) A string containing the filename of a ".txt" file with hip words

Outputs:

1. (*double*) The level of hipster-ness present in the file

Function Description:

The number of phony hipsters is on the rise, and you don't want that to encroach on your genuine creativity and frugal means of clothing yourself. So, like any true hipster would, you write a MATLAB function to determine how much of a fake hipster someone is. The function will input the name of a ".txt" file containing the text of a conversation they had and a second ".txt" filename containing words to check for in the conversation. Your function should award one faux-hipster point for each occurrence of the "hip" words in the text conversation.

You should also award one faux-hipster point for every hashtag used in the conversation. A hashtag is defined as the pound (#) followed immediately by any non-space character.

Finally, you are able to have overlapping points. For example, if the word `'denim'` were in the hip words file, and `'#denim'` appeared in the conversation file, you would need to award 2 points because it is a hipster word and a hashtag. Finding word matches should NOT be case sensitive (`'scarf'` is the same as `'SCARF'`).

Notes:

- Be sure to close your files as you are coding.
- You are not guaranteed to have any "hip" words in the conversation text file.

Drill Problem #5

Function Name: ottendorf

Inputs:

1. (*char*) The beginning of the filenames of the ".txt" files being used as reference passages
2. (*char*) The name of a ".txt" file containing the cipher

Outputs:

1. (*char*) The secret message decoded using the cipher and reference passages

Function Description:

An Ottendorf—or book—cipher is a type of encoding method (referenced in the movie National Treasure) in which sets of numbers correspond to letters in some set of reference materials. In this variation, each letter will be encoded in the following manner:

```
<reference #>-<line #>-<word #>-<letter #>
```

For example, the cipher line 3-23-8-4 corresponds to the fourth letter of the eighth word in the twenty-third line of the third reference text file. The first number would usually represent a page number in a book, so you can think of each reference file as a single page of a whole unified reference.

Write a function that takes a file of encoded letters (one letter's sequence per line) and outputs the message decoded using a given set of reference documents.

Notes:

- The reference filenames will always be formatted as '<input1>_#.txt'. For example, if the first input of the function were 'hpLastPage' and the current cipher line were 3-23-8-4, you would be looking in the file called 'hpLastPage_3.txt'.
- The end of the cipher file will be marked with a line containing 0-0-0-0; blank (empty) lines should be represented as spaces in the decoded message output, and lines of punctuation in the cipher should be carried over to the output string as well.
- You do not need to worry about finding blank lines or punctuation in the reference files; no words with quotes, hyphens, etc. will be encoded.

Hints:

- Don't forget to close files as you finish using them.
- You should only tokenize using spaces.
- Remember that lines read in from a file are strings (the `str2num()` function may be useful).