Notice

This homework will require you to create functions that output text files in addition to variable outputs. Text files cannot be checked against the solution output using `isequal()`, but there is another function you can use to compare text files called `visdiff()`.

Suppose your output file is called `'textFile1.txt'` and the solution function produces the file `'textFile1_soln.txt'`. From the Command Window, type and run the following command:

```
visdiff('textFile1.txt','textFile1_soln.txt');
```

At this point, a new window will pop up. This is the MATLAB File Comparison Tool. It will not only tell you if the selected files match, but it will also tell you exactly what and where all of the differences are. Use this tool to your advantage. **Please note that sometimes the comparison will say, "No differences to display. The files are not identical, but the only differences are in end-of-line characters." Do not be alarmed if you see this; you will still receive full credit.**

Please keep in mind that your files must be named exactly as specified in the problem descriptions. The solutions will output files with '`_soln`' appended before the extension. Your output filename should be identical to the solution output filename, excluding `'_soln'`. Misspelled filenames will result in a score of 0. You will still need to use `isequal()` to compare non-text-file function outputs.

Also note, you can start the File Comparison Tool by clicking on "Compare" in the Editor ribbon if that is easier for you.

MATLAB has lots of additional functions for reading/manipulating low-level text files, but because we want you to learn the fundamentals of low-level file I/O, we have banned `fileread()` and `textscan()` **for all problems** on this homework. The use of either of those functions on any problem will result in a 0 for that problem.

Happy coding!
~Homework Team

Function Name: `drSeuss`

Inputs:

1. *(char)* The name of a text file, including file extension

Outputs:

1. *(char*) A sentence describing the rhyme scheme

Background:

Imagine you hopped in a time machine and went back to 1950 to help Dr. Seuss write a few of his books. However, before you can help write any books, Dr. Seuss wants you to prove yourself by analyzing the rhyme scheme of his different books. Armed with technology from the future, write a MATLAB function to analyze the rhyme schemes of Dr. Seuss's famous books.

Function Description:

Given a text file, write a function that first checks if the first and second line in the text file rhyme. If they do, output the string:

```
'Wow! Line one and two rhyme, reading this will be a great time.'
```

If line one and two don't rhyme with each other, check if the first and third line in the text file rhyme. If they do, output the string:

```
'Wow! Line one and three rhyme, reading this will be a great time.'
```

If neither the first and second line nor the first and third line rhyme, output the string:

```
'In line one, two, or three, rhymes were not meant to be.'
```

Example:

Text File:

```
Too wet to go out and too cold to play ball
So we sat in the house
We did nothing at all
```

Output:

```
'Wow! Line one and three rhyme, reading this will be a great
time.'
```

Notes:

- To check if lines rhyme, compare the last two letters of the last word on each line. Remember that uppercase and lowercase don't contribute to whether or not words rhyme, so this comparison should be case-insensitive.
- Each text file will contain 3 lines of text.
- None of the lines will end with a punctuation mark.

Function Name: `library`

Inputs:

1. (*char*) A string with a book title to be copied
2. (*double*) The number of lines to print
3. (*double*) The number of times per line the title should appear

Outputs:

(none)

File Outputs:

1. A text file of the book title written

Function Description:

You decide to take inventory of the bookshelf in your childhood room! For some reason, you want to repeat the title of your old favorite book over and over, a specified number of times.

Write a function that takes in an input of a book title along with the total number of lines to write it on and the number of times per line the title should appear. Each line should begin with `'Line <num>. '` and the title should have a space between it and the next title. The output file name should be the first word of the book title you are copying, made all lowercase, followed by the file extension '.txt'.

For example, if the input string is `'Goodnight Moon'` and it was to be written on 3 lines with 2 sentences per line, the output file should be named '`goodnight.txt`' and should look like this:

```
Line 1. Goodnight Moon Goodnight Moon
Line 2. Goodnight Moon Goodnight Moon
Line 3. Goodnight Moon Goodnight Moon
```

Notes:

- There should be no extra spaces at the end of each line.
- Words of the title will be separated by at least one space.
- There should be no extra blank lines in your file (this would occur if your last line has a new line character at the end of it).

Function Name: `magicTreehouse`

Inputs:

1. (*char*) The name of a text file, including file extension
2. (*char*) The first word to look for
3. (*char*) The second word to look for

Outputs:

1. *(double*) A 1x2 vector of the the number of times the first and second word appear in the text file, in order

Background:

In your quest to become a Master Librarian of Time, you can help Jack and Annie look for information in books to help them while they time travel in the Magic Treehouse. Jack and Annie are both looking for different words, so you decide to write a MATLAB function that can search for two words at once in a text file.

Function Description:

Given the name of a text file and two different words to look for, you will write a function to search through the text file and count the number of times both words appear. Then output the number of times the words appear, concatenated as a 1x2 vector.

Example:

Say you were give the name of the text file `'matlab.txt'` and that file contains the following three lines, a second input of `'love'` and a third input of `'MATLAB'.` The output would be `[1,3]`.

```
MATLAB
I love MATLAB!!
MATLAB is actually the best
```

Notes:

- Ignore all punctuation and non-letter characters that aren't spaces.
- Your search for words in the text file should be case sensitive.
- Words will always be separated by at least one space.

Hints:

- The `strtok()` function will be useful for separating sentences into words.

Function Name: `bookmateMatch`

Inputs:

1. *(char)* The name of a text file containing the first person's survey answers
2. *(char)* The name of a text file containing the second person's survey answers

Outputs:

1. *(char)* A sentence describing the compatibility between two potential bookmates

Function Description:

A significant part of reading is sharing the joy with someone else! To ensure that your potential bookmate is top notch and loves to read as much as you do, you will write a function that compares two individuals surveys and outputs a sentence describing the compatibility of the match as roommates who both love to read, or bookmates! The output will be formatted as:

```
'<1st bookmate name> and <2nd bookmate name> have a <percent>%
                bookmate compatibility score.'
```

Both files will be formatted the following way.

```
1     Name: Melanie March
2     Earliest Class: 10:00 AM
3     How late do you stay up:
4        Early
5     X  Late
6        Really Late
7     Favorite book: Harry Potter
8     Tell us about yourself:
9     i am 20 years old and am bringing my cat, fluffy, with me :)  i also
10    really really love to sing in the shower.  I hope my bookmate is
11    musical like mee ;D
12
```

The first 8 lines will always define the same information (Name on first line, earliest class on second, etc). Lines 4-6 will have the strings `'Early'`, `'Late'`, and `'Really Late'`, respectively and there will always be one and only one option selected, denoted by an `'X'` on the appropriate line. The bookmates will always write something on line 9 and may continue writing for an unknown number of lines.

Continued...

The criteria for deciding how compatible two bookmates are includes comparing the time of their first class, how late each stays up, and how talkative they are. Each one of those 3 criteria is worth 20 points for a maximum of 60 points. The points for each criteria should be calculated as follows.

First Class Compatibility:

Divide the difference in hours between the bookmates first classes by 12. Then subtract this number from 1, multiply by 20 and round to the nearest integer. For example, bookmates with their first classes starting at 11:00 AM and 1:00 PM will receive a score of 17 points because the difference in those times is 2 hours and `round((1 - 2 / 12) * 20)` ⇒ `17`

You have been given a helper function `timeDiff()` that will calculate time differences for you. Type `help timeDiff` in the Command Window for information on how to use the helper function.

"Stay Up Late" Compatibility:

Award 20 points to bookmates who stay up to the same time, 10 points to those who stay up to adjacent levels of lateness ("Early" and "Late", or "Late" and "Really Late"), and 0 points to those who stay up "Early" and "Very Late"

Talkativity Compatibility:

You want to make sure you find a bookmate who loves to read as much as you do! This will be based on the total number of characters in each bookmates response is to the last prompt (Tell us about yourself). You should concatenate each bookmates entire response into a single string (not including newline characters or leading/trailing whitespace). The shorter response's total number of characters will be divided by the longer response's total number of characters. That quotient will be multiplied by 20 and rounded to the nearest integer to get the total score for this section.

Once all of these scores have been calculated, the total percentage of compatibility will be the sum of all of the subscores divided by 60, expressed as a percentage between 0 and 100 rounded to the nearest percent.. Output this value in the string output as shown in the beginning of the description.

Notes:

- You can use `'%%'` in `sprintf()` to put a `'%'` in a character vector.
- You can use `strtrim()` to remove leading and trailing whitespace from a character vector.

Function Name: `hungerGames`

Inputs:

1. (*char*) Name of a text file, which contains tribute names and scores

Outputs:

1. *(char*) A description of the most likely tribute to win with their overall score

Background:

It is the time of year again for each of the districts to volunteer their tributes to compete in the annual Hunger Games. In order to distinguish the tributes, The Capital assess the skills of each by having judges assign scores based on how well they perform in a demonstration. In order to speed up the process, The Capital has decided to recruit you to use your MATLAB expertise to decide which tribute is most likely to win!

Function Description:

Given a text file containing the names of each tribute, and the scores they received, determine the average score of each and determine which tribute is most likely to win the Hunger Games. Each line in the text file will be formatted as follows:

```
1     <Tribute Name>:<score1>,<score2>,<score3>...
```

Go through the text file and calculate their average scores. The tribute with the highest score will be considered most likely to win. A string should be outputted in the following format:

```
'<Tribute Name> is most favored to win with a score of <average score>!'
```

If however, Katniss Everdeen is in the running, then regardless of what her score is, she should be most favored to win, and the following string should be outputted:

```
'Katniss Everdeen is most favored to win with a never before seen score of
                                    11!'
```

Notes:

- Round each averages to the nearest integer. There should be no decimals in the output.
- There will never be a tie for the highest score.
- There can be any number of tributes in the text file and there can be any number of scores for each tribute.
- Katniss's name can appear as: `'Katniss'`, `'Katniss Everdeen'`, or `'The Girl on Fire'` (all case insensitive).

Extra Credit

Function Name: `greatGatsby`

Inputs:

1. (*char*) A filename containing a guest list
2. (*char*) A filename containing a list of people that show up
3. *(double)* The total number of people allowed to attend the party

Outputs:

none

File Outputs:

1. A file containing a list of people that attended the party

Background:

You have been hired to be a bouncer for one of Jay Gatsby's epic parties! He has provided you with a guest list, but wants to make sure his party is full of people to get Daisy's attention. So, you need to let all the guests in first, then let more people in until the party is full.

Function Description:

Write a function that will take in two text files, one containing a guest list, and one containing a list of people that arrived to party, and will write a new text file containing the list of people that got into the party. In the output file, the list should contain all of the invited guests that showed up in the order they arrive, followed by the appropriate number of uninvited guests to reach the maximum party limit, in the order that they arrive.

Example:

```
Guest Limit: 4
      Guests:          Arrived:          ---->       Attended:
      Bob              Sally                         Sally
      Joe              Grace                         Joe
      Nick             Joe                           Bob
      Sally            Bob                           Grace
                       Rita
```

Notes:

- The guest list will be titled `<theme of party>_guests.txt`, and the arrived list will be titled `<theme of party>_arrived.txt`; The output file should titled `<theme of party>_attended.txt`.
- The names on the guest list and the arrived list will start on the second line.
- The first line of the output file should be `'Attended:'`
- The number of names on the arrived list will always be greater than or equal to the total number allowed in the party, and all names will be unique. The number of people on the guest list who arrive will always be less than or equal to the guest limit.

Extra Credit

Function Name: `shakespeare`

Inputs:

1. (*char*) The filename of a poem

Outputs:

1. *(char*) The string of the rhyme scheme

Background:

After finding out whether a short poem rhymes or not in a Dr. Seuss book, you want to flex your MATLAB muscle to help you out on your harder English work. You are taking your skills to Shakespeare to find the rhyme scheme of a poem.

Function Description:

Write a function in MATLAB that finds the rhyme scheme of a poem. The first input is a string and is the name of a file to read. The file will be some number of lines, and using the rhyming rules from the first problem (the last two letters of last word must match), you must find the rhyme scheme of the poem in the file. The rhyme scheme will be denoted as a string of capital letters, starting with `'A'`, where lines that rhyme have the same letter. That is, every line that rhymes with the first line will be an `'A'`, every line that rhymes with the next line (that doesn't rhyme with the first) is a `'B'`, and so on. There will not be more than 26 unique line endings (you will never need more than `'Z'`).

Example:

If the poem in the file is as follows:

```
Roses are red,
Violets are blue,
This problem has led,
Me straight to you!
```

The output string would be `'ABAC'`.

Notes:

- Although there is a strict bound on the number of letters, *please* do not try to hard code checking for each letter. Think of how to do it iteratively.
- The lines may or may not end with a punctuation mark of some sort.
- Rhyming should be case insensitive.
- There will be no blank lines in the file.

Hints:

- Consider finding out how to create a temporary string of the right length before you start looking for rhymes.

Extra Credit

Function Name: `theBoyWhoLovedArrays`

Inputs:

1. (*double*) A non-empty array of positive integers
2. (*char*) A filename

Outputs:

none

File Outputs:

1. A text file with a formatted table containing the array values

Background:

From the people who gave you the renowned mathematics book, The Boy Who Loved Math (sorta), CS 1371 now gives you the fun-filled, interactive, mind-bending (unofficial) sequel, The Boy Who Loved Arrays!

Function Description:

Sometimes it is important to share formatted data but you don't have the luxury of using Microsoft Office. In these cases, a plain-ol' text file will have to suffice. But don't worry, your tables can still look good! You will be given an array of data that you need to convert into a formatted table and write that table to a text file. Here's the formatting specifications:

- Each column of the table should be the width of the largest number in that column.
- Each row should be delimited by `-`'s
- Each column should be delimited by `|`'s
- The intersection of the row and column delimiters should be +'s
- All numbers should be left-aligned in their cell and padded with spaces

The formatted array should be written to a file specified by the second input.

Example:

```
array = [  45, 78923, 3; ...
          8923,     9, 8]
fileName = 'example.txt'
```

After the function runs, a file named `example.txt` should be created containing the following:

```
+----+-----+-+
|45  |78923|3|
+----+-----+-+
|8923|9    |8|
+----+-----+-+
```

Notes:

- The file should not have an extra new line at the end.
- You can run the solution function to see the files created if you need more examples.