

### 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.

Coding!

~Homework Team

**Function Name:** musicAwards

**Inputs:**

1. (*char*) Filename of a text file containing the winners of a music awards show
2. (*char*) Category of award you need to change
3. (*char*) Artist and/or song you should substitute as the winner

**Outputs:**

None

**File Outputs:**

1. An updated text file containing the new winner under the appropriate category

**Background:**

You are a high-ranking official working for a certain music awards show. You have been trusted with the responsibility of getting the proper list of categories and winners on stage to be announced. However, you've discovered that someone has been changing the results of certain categories to the performers that they think should win. It is up to you to keep an eye on those categories that are potentially altered, and make sure that the correct winner gets announced on stage.

**Function Description:**

Write a function that will read through a text file to find the given category (2nd input), and change the winner to the one specified (3rd input). Then write the results to a new file, titled `<originalFileName>_updated.txt`.

**Example:**

results.txt:

```
Album of the year: 24K Magic - Bruno Mars
Song of the year: Despacito - Luis Fonsi
Best Rap Song: HUMBLE. - Kendrick Lamar
```

```
>> musicAwards('results.txt', 'Song of the year', 'That's What I Like - Bruno Mars')
```

results\_updated.txt:

```
Album of the year: 24K Magic - Bruno Mars
Song of the year: That's What I Like - Bruno Mars
Best Rap Song: HUMBLE. - Kendrick Lamar
```

**Notes:**

- The categories will always be separated from the songs/artist with a colon.
- The category that you will have to change will never be on the last line of the text file.

**Function Name:** catchyCounter

**Inputs:**

1. (*char*) The filename of a text file containing lyrics to a song
2. (*char*) A word/phrase to count

**Outputs:**

1. (*double*) The number of times the word/phrase appears

**Background:**

You are starting a job as a songwriter for some of the music industry's biggest stars. But before you can write your first hit single, you need to figure out what makes a song catchy. You find that repetition can be very effective, so you decide to count the number of times a specific word/phrase appears in notoriously catchy songs.

**Function Description:**

Write a function in MATLAB that counts the number of times a certain word or phrase appears in a song's lyrics. The first input will be the filename of a .txt file that contains the lyrics to a song. The second input will be the word/phrase you are counting in those lyrics. Your function should output the total number of times that word/phrase appears in the .txt file, regardless of case.

**Example:**

beat\_it.txt:

```
Just beat it, beat it, beat it, beat it
No one wants to be defeated
Showin' how funky and strong is your fight
It doesn't matter who's wrong or right
Just beat it (beat it)
```

```
>> num = catchyCounter('beat_it.txt','beat it')
```

```
num = 6
```

**Notes:**

- Because the phrase may be longer than one word, you should use `strfind()`. Also, the word or phrase that you are looking for might be a part of a larger word or phrase.
- You do not need to check if a phrase occurs across multiple lines.

**Function Name:** albumArt

**Inputs:**

1. (*char*) The name of a text file containing ASCII album art

**Outputs:**

None

**File Outputs:**

1. A text file containing the review of the album art

**Background:**

You have just been hired to rate all of the art in a old shop that still sells this ancient technology: "CDs". You want to get out of this musty old place as quickly as possible, so you decide to use your handy MATLAB knowledge to rate all of the albums for you so you can go grab a tall, nonfat latte with caramel drizzle from Starbucks.

**Function Description:**

Given a text file, you must read the average value of all the characters in the image using the table below:

Character	'@'	'%'	'?'	'!'	'*'	'~'	'.'	' '
Darkness	1	2	3	4	5	6	7	8

Once you have found the average value of the all the characters in the text file based on darkness, you must determine a descriptive statement. If the average darkness is greater than 5, the descriptive statement is 'Very light cover, probably good music for kids!', if the average darkness is greater than 3, the descriptive statement is 'Cover of medium darkness, seems like it could be a bit too edgy...', and for all other possibilities the descriptive statement is 'Cover is very dark, looks like something quite inappropriate for children.'. Then, print out a text file named '<name of the file>\_review.txt' with the following form:

```
Review of <filename without the extension>'s album art:
The cover has a darkness score of <average of all characters>.
<The descriptive statement>
```

**Notes:**

- All your output files should be 3 lines long.
- Your darkness score should be rounded to 2 decimal places.

**Function Name:** formation

**Inputs:**

1. (*double*) 1xN vector of the number of dancers in each line
2. (*char*) A file name, including file extension, for the output file group number

**Outputs:**

None

**File Outputs:**

1. A text file showing the dancers in formations

**Background:**

It's the 2018 Grammys. Beyoncé needs help. The dance coordinator lost the sheet showing the formation of the dancers. Luckily, the coordinator kept a backup sheet that showed the dancers formation in vectors, but the dancers need their actual positions on stage. They've heard of your amazing MATLAB skills and have enlisted you to save Beyoncé's show.

**Function Description:**

Write a function that takes in the vector of dancers in each line and outputs a file showing the dancers' formations. To find the dancers' formations and create the output file, follow these rules:

1. Find the maximum value in the input vector. Each line of the formation will have this many places.
2. For each line of the formation:
  - a. Put the specified number of dancers from the input vector in the line. For example, if the first number in the vector is 3, there are 3 dancers in the first line.
  - b. Align the dancers along the center. If there are 3 dancers and 5 spaces, there will be one empty space at each end of the line.
  - c. Use a 1 to represent a dancer and a 0 for a vacant position.
3. The output file should print the lines in the following format:

Line <num> has the following line up: <formation>.

- a. <num> refers to the current line number, which should start at 1 and increment by 1 for each line, and <formation> refers to the dancers' formation.
  - b. There should be two spaces in between the numbers representing the positions.
4. Use the second input to title the output file.

*Continued...*

**Example:**

```
>> formation([3 5 1], 'beyonce.txt')
```

beyonce.txt:

Line 1 has the following line up: 0 1 1 1 0

Line 2 has the following line up: 1 1 1 1 1

Line 3 has the following line up: 0 0 1 0 0

**Notes:**

- You are guaranteed that all numbers will be odd.
- There should not be an empty new line at the end of the file.

**Hints:**

- Think about how you can find the center of each line and index from there.
- The `zeros()` function may be helpful.
- What happens when you call `num2str()` on a vector of doubles?

**Function Name:** mashUp

**Inputs:**

1. (char) The name of a text file containing song 1
2. (char) The name of a text file containing song 2
3. (char) The name of a text file containing instructions on what song lines to take


**Outputs:**

None

**File Outputs:**

1. A text file containing the mashed up song

**Background:**

Since originality in the popular music industry is lost forever, you decide to mash up already existing songs to make some  new singles.

**Function Description:**

Given the filenames of text files that contain two different songs and a third file of instructions, you will combine different lines together in a new text file. The instructions will always follow this format:

<song # (1 or 2)>-<line #>

The name of your new text file should be <song1>\_<song2>\_mashUp.txt, where <song1> and <song2> are the filenames from the input, without the extension.

**Example:**

song1.txt:

```
We're soaring, flying
There's not a star in heaven that we can't reach
If we're trying
So we're breaking free
```

song2.txt:

```
You get the best of both worlds
Chill it out, take it slow
Then you rock out the show
You get the best of both worlds
Mix it all together
And you know that it's the best of both worlds
```

*Continued...*

instructions.txt:

1-1  
2-2  
2-3  
1-4  
2-1

```
>> mashUp('song1.txt', 'song2.txt', 'instructions.txt')
```

song1\_song2\_mashUp.txt:

We're soaring, flying  
Chill it out, take it slow  
Then you rock out the show  
So we're breaking free  
You get the best of both worlds

**Notes:**

- Each line of lyrics should be on its own line in the output text file.
- There should not be a trailing newline at the end of the text file.



### Extra Credit

**Function Name:** moshUp

**Inputs:**

1. (*char*) Filename of a text file containing first song's lyrics
2. (*char*) Filename of a text file containing second song's lyrics
3. (*char*) Filename of a text file containing the instructions on how to combine the songs

**Outputs:**

None

**File Outputs:**

1. A new text file containing the **moshed**-up song

**Background:**

After writing `mashUp()`, your first few 🔥 singles (and eventually your debut album) caused quite a splash in the underground music scene and eventually earned you widespread popularity. Recently, however, your new weird friend (who always dresses in dark colors and listens to death metal) has inundated you with enough rhetoric that you no longer subscribe to conventional notions of popularity, traditional music structure, happiness, and society in general. Mash-ups are dead, and it's time to give the term "experimental music" new meaning.

**Function Description:**

A **mosh**-up is similar to a mash-up, but more avant-garde. Instead of only shuffling lines around, you'll shuffle around lines, words, and even characters. The first two inputs will be the lyrics from two songs, and the third file will be a text file containing the scheme for the **mosh**-up. The scheme will contain lines following the following format:

<song # (1 or 2)>-<line #>-<word # in line>-<character # in word>

Each line will contain, at a minimum, the first two specifications (song number and line number), and can contain one or both of the other specifications (word number and character number). For example, the following line

1-4-6

means "take the **sixth** word from the **fourth** line from the **first** song and put it on the current line."

A line containing nothing but the word 'newline' in the scheme file indicates that a newline should be inserted into the new song at that point. A line containing nothing but the word 'space' indicates that a space should be inserted into the song at that point.

The name of your new text file should be `<song1>_<song2>_moshUp.txt`, where `<song1>` and `<song2>` are the filenames from the input, without the extension.

*Continued...*

**Example:**

song1.txt:

```
I seen some free landed some tricks
Far I see high time man quit
Won't let you know when I get goin
Phone ring too long
```

song2.txt:

```
But please be honest
Tell me was it you?
I won't hate you
I just need to know
Please be honest
Tell me was it you?
```

instructions.txt:

```
1-4
newline
2-6
newline
1-1-1
space
1-1-2
space
1-1-3
space
2-1-4-1
2-2-5-4
```

```
>> moshUp('song1.txt', 'song2.txt', 'instructions.txt')
```

song1\_song2\_moshUp.txt:

```
Phone ring too long
Tell me was it you?
I seen some h?
```

**Notes:**

- Don't insert newlines or spaces except where indicated in the instructions file.
- The instructions file is guaranteed to only refer to existing words, lines or characters (i.e. it won't ask for the fifth word on a line with only four words).
- The output file should not have a trailing newline.
- Only use spaces as delimiters to separate words.