Drill Problem #1

Function Name: countCelery

Inputs:

1. (*cell*) A 1xN cell array
2. (*double*) Total number of crops

Outputs:

1. (*double*) Number of crops stolen

Function Description:

Count Celery is at it again! The illiterate ruler of the MATLAB Kingdom refuses to do his own dirty work, and has once again employed you to do his bidding. His supreme empire built on the sale of the magical zero-calorie vegetable - Celery - is in danger, because his workers have once again been stealing from him. Knowing you are the Minister for MATLAB, he has tasked you with writing a MATLAB function to figure out how many crops of celery have been stolen from his fields. However, the workers are particularly cunning and have insider knowledge that you have not yet learned recursion. To combat this, they have hidden some of the celery in nested cell arrays. Because of this you are only able to find the celery that is not contained in a nested cell.

Write a MATLAB function that takes in a 1xN cell-array and the total number of celery crops that the Count has in his fields. The function outputs the number of pieces of celery that were stolen; to do this, subtract the number of times the word `'celery'` occurs within the cell array from the total number of crops Count Celery has in his field (the second input), while ignoring all instances of the word `'celery'` within nested cell-arrays.

Example:

```
cellArr = {'celery', 'celery', {{{'celery'}, [], []}, 'celery'}, [], {'celery'}, 'celery'}

stolen = countCelery(cellArr, 9)
stolen => 6
```

Although the word `'celery'` occurs 6 times in the cell array, 3 of those instances are nested and therefore ignored.

Notes:

- The `iscell()` function will be useful.
- The word `'celery'` will always be in lowercase.
- There could be other strings, doubles, etc. stored in the cell array.

Drill Problem #2

Function Name: pantsOnFire

Inputs:

1. (*cell*) A 3xM cell array

Outputs:

1. (*char*) A string saying what items of clothing you should and should not wear

Function Description:

You have just finished your MATLAB homework for the week and you are ready for a night at the fair. The only problem is that you have no clue what to wear. You decide to ask your friend; however, this friend has a habit of lying to you to spare your feelings. Thus, you don't know whether to trust what they say. Thankfully, you can *always* trust MATLAB, so you decide to use MATLAB to figure out what to wear from what your friend says.

You will be given a cell array with three rows. The cells in the first row will each contain a string of an item of clothing you are considering wearing. The following two rows in your cell array consist of true and false values; these values correspond to whether or not your friend said an item of clothing looked good on you, and to whether this friend actually thinks an item of clothing in question looks good on you.

Find which items your friend both said looks good on you and also thought looks good on you. These are the items you should choose from to wear, and the rest are items you should definitely not wear. You will then print a statement with the following format:

```
str = 'You should wear the <items>, but DO NOT wear the <noItems>.'
```

With the following caveats:

1. There will always be at least one clothing item that looks good and one clothing item that does not.
2. Multiple items should be separated by a comma (except for the last item). For the clothes to wear, the string should include an `'and/or'` before the last item (instead of the comma). For clothes to not wear, the string should only include an `'or'` before the last item (again, instead of the comma).
3. If there are only two items, there will not be commas—there will only be the `'and/or'` or `'or'` keywords.

For example, given the following input of `closetTest`, the output should be `wearTest`.

```
closetTest = {'socks', 'boots', 'corduroy pants'; false, true, true;
     true, false, true}
wearTest = 'You should wear the corduroy pants, but DO NOT wear the
     socks or boots.'
```

Notes:

- Everyone should have an opinion about the Oxford comma. Despite your opinion, there should not be a comma before either the `'and/or'` or the `'or'` in the output string.

Drill Problem #3

Function Name: driveTime

Inputs:

1. (*char*) The string of an Excel ('.xlsx') file name

Outputs:

1. (*double)* An Nx2 array of sorted and filtered data

Function Description:

You have acquired a summer internship doing modeling and simulation of various traffic scenarios. One particular model takes progressive snapshots over time and records which vehicles are within the model at those times. You would like to know the total amount of time each vehicle spends in this model to get an idea of the delay through several blocks of congested traffic. Your modeling software outputs Excel files, so you decide to write a MATLAB function to organize the data for you.

The first two column headers of the file will always contain the strings 'TIME' and 'VEHIC_ID' (in that order), possibly followed by other extraneous column headers. Each row will contain a single vehicle number under the 'VEHIC_ID' column, as well as a timestamp under the 'TIME' column indicating that the vehicle was in the model at that particular time (given as an increasing time count). You need to figure out the time at which each vehicle enters the model and the time that it leaves, i.e. the lowest and highest time values. The difference will be the total time that each vehicle spent in the model. You will then output the data to an Nx2 array of doubles with the first column being a sorted vector of all of the car IDs (without duplicates) and the second column being the amount of time each car was in the model, rounded to the nearest tenth of a second.

You are guaranteed that there will be no blank spaces, strings, NaNs or other erroneous data within the TIME and VEHIC_ID columns. There are no guarantees that the data is in any particular order (increasing timestamps, incremental vehicle IDs, etc.). It is possible that a vehicle enters the model just as the model is ending and therefore the vehicle only has 1 data point. In this case, you should output a duration of 0 for that vehicle.

Notes:

The `unique()` and `sortrows()` functions may be useful.

Drill Problem #4

Function Name: jackBlack

Inputs:

1. (*cell*) A 1xM cell array of Jack Black movie names
2. (*cell*) A 1xM cell array of vectors containing viewer ratings corresponding to the movies from the first input
3. (*char*) The name of an Excel ('.xlsx') file containing movie names and times

Outputs:

1. (*char*) A text to your friend letting her know which movie you two will see and at what time

Function Description:

You and your friend have received free tickets to a Jack Black film festival and decide to spend your whole day watching the greatest hits and "talking about [movies] that rock!" You and your friend prioritize a list of movies that you definitely want to see (first input) and then decide to watch the one with the best viewer ratings (second input) first so that you don't miss it later. Your friend, having already seen all Jack Black movies, decides to search the fairgrounds for the man himself, while you watch the first film, and join you for the second movie viewing.

You are ok with watching either the 2nd highest rated or 3rd highest rated film when your friend joins you, so you decide to choose between the two based on which one has a shorter wait time between the end of your first movie and the beginning of either of those two movies. For example, if your first movie ends at 12:20, the second ranked movie's next starting time is 13:30, and the third ranked movie's next starting time is 12:45, then you will watch the third ranked movie because it has a shorter wait time. If the second and third ranked movies start at the same time, then you will watch the second highest rated movie.

Your friend keeps changing their mind about which movies to prioritize and whose ratings to use, so you decide to "service society by [writing a MATLAB function]" that will take in those two pieces of information and the published movie schedule and output a string that you will text to your friend that says: `'We''re going to see <movie name here> at <time movie starts here>. See you then :)'`.

The movie schedule will be formatted as shown below with column headings in the first row, an unknown number of movie titles in the first column, the 5 show times for those movies in the corresponding columns, and the total run time of the movie is the seventh (last) column of a movie's row. The schedule information is composed entirely of type 'char' except for the run time, which is type 'double'.

Title	ShowTime1	ShowTime2	ShowTime3	ShowTime4	ShowTime5	RunTime
Tropic Thunder	12:35	13:35	15:55	…	…	132
Ice Age	14:00	16:35	…	…	…	85
School of Rock	…	…	…	…	…	…
…	…	…	…	…	…	…

Drill Problem #4

The viewer ratings are given as a cell array, where each cell contains a vector of viewer ratings. Average each vector of ratings per cell entry to find the average rating, and use these mean values to rank your input movies list. The vector of ratings may be different sizes in each cell.

Notes:

- Because you don't need to see the credits and the opening trailers at every movie, you may start watching a new movie at the same moment the previous movie ends (If your first movie gets out at 12:00, you can start watching your second movie as 12:00, too).
- For your highest rated movie to see first, you will want to see the first viewing time available.
- All times in the `'.xlsx'` file are written in military time.
- You are guaranteed to have three movies on your list of movies to see.

Hints:

- Consider writing a helper function to convert the strings of military time to doubles that can do math with the run time of the movie (in minutes).

Drill Problem #5

Function Name: feeFiFauxPho

Inputs:

1. (*char*) The name of a '.txt' file representing a pho recipe
2. (*char*) The name of an Excel ('.xlsx') file representing your purchased ingredients list

Outputs:

1. (*logical*) A logical value stating whether the recipe can be used or not
2. (*cell*) A cell array containing updated values from the purchased ingredients list

Function Description:

Have you ever seen a restaurant sign advertising the wonderful Vietnamese dish known as pho? (Fun fact: actually pronounced as "fuh," not "foh"). Well, while grocery shopping yesterday you decided to pick up some extra ingredients so you could make your own pho! The only problem was, you didn't have a recipe or ingredients list on hand. Now, once you get home you must check different recipes you found online to see if any will work with the ingredients you've purchased.

Write a function named `feeFiFauxPho()` that will take in a string with the name of the text file containing one of the recipes, and another string representing an excel sheet with a list of the items that you've purchased. You should check if you have all the required ingredients, and also if you have an adequate amount of each ingredient. The first output should be a single logical value indicating whether the specified recipe can be used or not according to these criteria. The second output should be an updated cell array in the similar format as the raw input data from the Excel file; this should contain the remaining amounts of the ingredients after the recipe is used. If the specified recipe cannot be used, the second output would be a cell array containing the original amounts of ingredients (since none were used), again similar to the raw input data. The input Excel file will always contain columns labeled `'Amount'`, `'Unit'`, and `'Ingredient'`, however, they can be in any order and there can be any number of extraneous columns of data. Your output cell array should *only* contain the columns `'Amount'`, `'Unit'`, and `'Ingredient'`, *in that order*, and the order of the ingredients should be the same as the original.

Notes:

- The first line of the '.txt' file will always be the name of the recipe.
- Each line of the text file will have an amount as a number, followed by a unit, then the item.
- A single space will always separate each number and word in the recipe text file.
- The units of measurement will always be identical in both the recipe and the shopping list. Furthermore, the units of measurement will always be a single word and will immediately follow the amount in the recipe text file.

Hints:

- The `strcmp()` function works for cell arrays.

Drill Problem #6

Function Name: superSmashBros

Inputs:

1. (*char*) Filename of an excel file (`'<name>_Characters.xls'`) indicating which characters each player used
2. (*char*) Filename of an excel file (`'<name>_Scores.xls'`) with the scores of each player per game

Outputs:

1. (*char*) An output string describing which player had the most wins and which player is the best
2. (*cell*) An MxN cell array with scores and winners

Function Description:

Super Smash Bros.! The epidemic that has taken hold of most of Georgia Tech and all engineering schools alike. After many arguments, you and your friends finally decide to meet up to decide once and for all who the best Super Smash player is. You, however, realize that there is much more to being the best than just winning. There is also the scores at the end of the games and the difficulty of using certain characters. Luckily, with your newfound MATLAB skills, you decide to write a function that will determine who is the best player!

Given two Excel files that contain the characters chosen and the scores obtained from each game, respectively, write a MATLAB function named `superSmashBros()` that outputs:

- A cell array with the scores AND a winners column.
- A string saying who won the most games AND who the best player is (determined by who has the most weighted points).

Use the following table to determine the weight each character has on the player's score (characters with higher weights are harder to use). These values are determined by the First Tier List for Super Smash Bros. (N64).

Character	Weight
Link	8
Samus	
Luigi	
DK	4
Yoshi	
Mario	
Jigglypuff	2
C. Falcon	
Fox	
Ness	0
Kirby	
Pikachu	

Weights are added to score values to determine weighted scores. Each player's weighted scores will then be added together to get the total weighted score. The person with the highest weighted score is the best player.

- If the player that won the most games is the same player that has the highest total weighted score, then the output string should read `'<PlayerName1> won the most games and is the best player!'`.
- If, however, the players are different, then the output string should read `'<PlayerName1> won the most games, but <PlayerName2> is the best player!'`.

Drill Problem #6

Here is an example:

The file `'finalDestination_Characters.xls'` contains the following information:

Game	Zack	Nairita	Juan	Pranaya
1st	Jigglypuff	Mario	Pikachu	Samus
2nd	Luigi	DK	Yoshi	Fox
3rd	Pikachu	Pikachu	Pikachu	Pikachu

And the file `'finalDestination_Scores.xls'` contains:

Game	Zack	Nairita	Juan	Pranaya
1st	-5	3	4	-2
2nd	2	4	5	-10
3rd	-3	1	0	2

The output cell array should contain:

Game	Zack	Nairita	Juan	Pranaya	Winner
1st	-5	3	4	-2	Juan
2nd	2	4	5	-10	Juan
3rd	-3	1	0	2	Pranaya

However, because of the weight of using specific characters, the weighted scores would be:

Game	Zack	Nairita	Juan	Pranaya
1st	-3	7	4	6
2nd	10	8	9	-8
3rd	-3	1	0	2
Total	4	16	13	0

And Nairita would have the highest weighted score. Hence, the output string should read:
`'Juan won the most games, but Nairita is the best player!'`

Notes:

- There will be 2 to 4 players, and there can be any number of games played.
- The first column of each file will contain the game number (as a string).
- You are guaranteed to have no ties in the number of wins or in total weighted scores.

Hints:

- You may find both outputs of the `sort()` and `max()` functions useful.
- The `strcmp()` function, and its use with cell arrays, may be useful.