You may notice that the O (output) in High Level File I/O is left out of this homework. This is because `xlswrite()` does not work on Macs. For your convenience, we do not require you to use this function, except on the ABCs. However, you will still need to know the semantics of this function for the tests. We have set up some of the functions such that the output is a cell array that could readily be written to an Excel file, we just don't require that you actually write any files. If you need to check your code for the ABCs and you have a Mac, you can run the code on a library computer or use Virtual Lab.

Happy coding!
~Homework Team

Function Name: `shallots`

Inputs:

1. (*cell*) A 1x1 cell

Outputs:

1. *(double*) The depth, or number of layers, of the cell

Background:

Shallots are the most underrated vegetable on Earth. In order to show that you are an esteemed member of the elite class of home vegetable growers, you tear down your entire house and plant nine thousand square feet of shallots. As one of the world's foremost shallot scientists on the cutting edge of shallot research, you are curious to see how many layers your shallots have.

Function Description:

Given a singular nested cell as your input, write a function that counts how many times the element is nested, or how many ‘layers’ the cell has.

Example:

```
>> layers = shallots({{{{{'shallot'}}}}})
layers = 5
```

Function Name: `myCabbages`

Inputs:

1. (*cell*) An Nx3 cell array of your cabbage cart statistics

Outputs:

1. *(cell*) An (N+1)x3 edited cell array of your cabbage cart statistics

Background:

You love traveling all over the world selling cabbages from your cabbage cart. You journey all the way to the great city of Omashu to sell your spectacular cabbages but get stopped by the guards outside the city's gates because they believe your cabbages are rotten. While you know this is untrue, as you only sell cabbages of the most superior quality, the unappreciative guards throw your entire cabbage cart off a cliff! Oh no! Your cabbages! Distraught, you decide to make a MATLAB function that calculates how much money you lost, so you can sue for damages and be prepared just in case this ever happens again...

Function Description:

Write a function that edits a cell array of your cabbages cart statistics to find how many cabbages and how much money you lost. The cell array will have three columns - the first lists different types of cabbages you sell, the second has how many cabbages of that type you have, and the third has their corresponding price per cabbage. Sort the array by the price per cabbage with the cheapest type of cabbage at the top. Then, calculate the total number of cabbages you had and the total amount of money your cabbages were worth, and concatenate these values to the bottom of the array in that order, with the words `'Total loss'` in the first column.

Example:

```
>> arr =
```

Savoy	3	1
Chinese	5	3
Napa	2	2

```
>> out = myCabbages(arr)
```

Savoy	3	1
Napa	2	2
Chinese	5	3
Total loss	10	22

Notes:

- The columns are guaranteed to be in this specific order.
- Round the total amount of money lost to 2 decimal places.

Function Name: `onions`

Inputs:

1. (*cell*) An MxN cell array

Outputs:

1. *(double*) A 1x2 vector of the position of the element with the most cell array "layers", in the form [row, col]
2. (*double*) The number of "layers" of this element

Background:

Everyone remembers the famous, comical line from Shrek to Donkey in the original Shrek movie: "Onions have layers. Ogres have layers. Cell arrays have layers. You get it? We all have layers." But the question Shrek didn't answer is how many layers these items have. Luckily, you can use your knowledge of MATLAB to count the number of layers in a cell array. We can then use the transitive property to determine how many layers onions and ogres have.

Function Description:

Given an input cell array, write a function that searches through the elements of a cell array. At each position, determine how many layers of cells there are until the base layer (when the next layer is no longer a cell). Your function should output the row and column indices of the element of the cell array with the greatest number of layers, and how many layers this element has.

Example:

```
>> onionGarden = [ {{6}}, {{{'hello'}}};
                   {{{{3}}}}, {false} ]
>> [pos, layers] = onions(onionGarden)
pos = [2, 1]
layers = 4
```

Notes:

- There will be no ties for the greatest number of layers.
- Each cell will always contain either a 1x1 cell or a single non-cell element.
- The contents of cells at the deepest layer do not matter, only the number of layers.

Hints:

- If you are unsure where to start, look at `shallots()`.

Function Name: `asciiGarden`

Inputs:

1. *(char)* A filename of an Excel file containing garden information, including file extension

Outputs:

None

File Outputs:

1. A text file representation of your new garden

Background:

Spring has finally arrived, and you can't wait to finally realize your dream of planting a beautiful garden to provide you with a supply of fresh fruits and veggies. You decide to use your favorite tool, MATLAB, to help you plan the layout of your new garden!

Function Description:

Write a function that reads in an Excel file describing the layout of your garden, and creates an ASCII art representation of your new garden as a text file, named as follows: `ascii_<Name of the Excel file without extension>.txt`.

Each column of the Excel file will contain the following information in this order:

1. The names of the fruits and vegetables to be planted in the garden
2. The character representation corresponds to each plant
3. The number of rows of the garden that should contain the fruit or vegetable

Example:

`>> asciiGarden('garden1.xlsx')` should write the following text file to your current directory:

`garden1.xlsx:`

Tomatoes	O	1
Pumpkins	~Q	2
Carrots	V	3
Peas	oo	1

`ascii_garden1.txt:`

```
OOOOOOOOOOOOOO
~Q~Q~Q~Q~Q~Q~Q
~Q~Q~Q~Q~Q~Q~Q
VVVVVVVVVVVVV
VVVVVVVVVVVVV
VVVVVVVVVVVVV
ooooooooooooo
```

Notes:

- Your garden plot should have twice as many columns as rows.
- All test cases are guaranteed to have full vegetables planted; for instance, if a vegetable is three characters long, the number of columns will be divisible by three.

Function Name: `veggieParty`

Inputs:

1. (*cell*) Cell array of various dishes and their ratings
2. (*char*) Rating type of interest
3. (*char*) Filename of an ingredients text file, including file extension

Outputs:

1. *(cell*) Sorted cell array of dishes and ratings
2. (*cell*) 1xN cell array of the ingredients needed to make the top rated dish

Banned:

`sortrows(), strsplit()`

Background:

You're planning a party where people are vegetarians. It's a veggie party!

Function Description:

Write a function called `veggieParty` that takes in a cell array of vegetarian dishes with their rating scores and sorts the cell array based on a specific rating type. The function should also determine the ingredients needed to make the top rated dish.

The provided cell array input will contain all the possible dish options in the first column, and various types of rating scores for each dish in the other columns. The first row of the cell array will contain the headers for each column. Determine which column contains the rating of interest, as specified by the second input, and sort the entire cell array based on the values from this column in descending order, such that the top rated dishes according to the rating type are at the top of the array.

Then, determine the ingredients of the top rated dish using the text file given as the third input. Each line in the text file will be formatted in the following way:

```
<dish>: <ingredient1>,<ingredient2>,...<ingredientN>
```

Output the sorted cell array of dishes with ratings and the 1xN cell array of the ingredients needed to make the top rated dish. The ingredient array should be in the same order as in the text file.

Notes:

- The first column of the input cell array will always contain the dish names.
- The rating type columns can be in any order.
- There can be any number of ingredients for any particular dish.
- The rating type of interest is guaranteed to appear as a header in the input cell array.
- The top rated dish is guaranteed to be found in the ingredients text file.

Extra Credit

Function Name: `celery`

Inputs:

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

Outputs:

None

File Outputs:

1. A formatted text file containing information about the initial text file

Banned Functions:

`unique()`

Background:

You recently started writing up some of your favorite short stories and scripts on your computer, but since you have started working with celerys *cough cough* I mean cell arrays in MATLAB, they have been corrupted! Oh no! Now, you must embark on an epic quest to use the very same piece of software that corrupted your files to analyze your files.

Function Description:

You will be given a text file. Extract every unique word and count the number of times each unique word appears in the text file. Case should be ignored when looking for unique words, and any characters that are not letters or spaces should be removed. The words should appear in decreasing order of how often they appeared in the text file. If there is a tie, you should sort the ties alphabetically. To make the output text file, follow these rules:

- The output text file should have the name `<filename>_counted.txt`.
- The output file should start with `'A counting of the story <filename>:'`.
- The second line should read `'The number of times celery was seen: <number of celery>'`.
- The third line should read `'All other words:'`.
- The rest of the file should list the unique words in the following form: `'<word>: <num times>'`.
    - Each word should be lowercase.
    - Put each word on a new line.
    - Do not include the word "celery" in this section.

Continued...

Example:

```
twinkle.txt:
      twiNk)(le twinKLe# li,ttle celery
      H34ow i ce//lery wh*at you are

twinkle_counted.txt:
      A counting of the story twinkle:
      The number of times celery was seen: 2
      All other words:
      twinkle: 2
      are: 1
      how: 1
      i: 1
      little: 1
      what: 1
      you: 1
```

Notes:

- There should NOT be a newline character at the end of your text file.
- All the words in the text file should be lowercase.

Hints:

- `sort()` preserves the original order of duplicated elements. In order to sort by two things, think about how you could use the `sort()` function twice.