**Update 4/4/2017**

Having noticed that this week's homework may be a bit too challenging, we have decided to make a couple changes. There are now 9 Drill Problems to choose from on this document. You are only required to complete **4 out of these 9 problems** to receive a 100 on this homework. You will **NOT** receive any extra credit for completing more than 4 problems, but you are of course allowed to do more for extra practice. However, you may only submit up to 4 regular drill problems on the t-square assignment.

Another document called "HW12_DrillProblemsExtraCredit.pdf" contains the 3 extra credit problems: `solveSudoku`, `spaceJam`, and `sixDegreesOfPotter`. Completing any or all of these problems can give you up to 60 extra credit points, still making the max grade for this homework 160/100. If you have any questions regarding this change, please direct them to Piazza.

Happy Coding!
~Homework Team

**Function Name:** `deepestLayer`

**Inputs:**

1. (*cell **or** other*) A 1x1 nested cell

**Outputs:**

1. (*char, logical, **or** double*) Contents inside the input cell

**Background:**

You've just been accepted into Hogwarts School of Witchcraft and Wizardry. You miraculously find your way to platform 9 ¾ and make it onto the Hogwarts Express, with just a few seconds to spare. You're exhausted and hungry, so you decide to buy a chocolate frog to satisfy that sweet tooth of yours, and maybe if you're lucky get a limited edition Dumbledore Chocolate Frog Card! Unfortunately, Draco Malfoy decides to play a prank on you and casts a spell on your chocolate frog so that you're forced to open it anywhere from one time to a million times before you can get to the deepest layer! Little does Malfoy know, you've been taking CS1371. With MATLAB at your side, he won't get away with this!

**Function Description:**

A nested cell is a cell that contains another cell within itself; it can be nested a multiple number of times (cell-ception). You can think of your spell cast chocolate frog as a nested cell. Write a MATLAB function that will take an input, 1x1 nested cell and output the contents inside the innermost cell, through recursion.

For example, if the input cell, `ca`, were the following:

```
ca = {{{{{{{'chocolate frog'}}}}}}}
```

then the function would recursively peel each layer of the cell away as follows:

```
call 1:  ca = {{{{{{'chocolate frog'}}}}}}
call 2:  ca =  {{{{{'chocolate frog'}}}}}
call 3:  ca =   {{{{'chocolate frog'}}}}
…
…
…
```

call n (where n = 7 in this example):

```
ca = 'chocolate frog'
```

Which would be the output value.

**Notes:**

- If the input is not of class `'cell'` just output the input as itself.
- You will not have a cell with a length greater than one.
- You MUST use recursion to solve this problem.

**Function Name:** `anagram`

**Inputs:**

1. (*char*) String representing the anagram
2. (*char*) Name of text file to write to

**Outputs:**

none

**File Outputs:**

1. A text file containing all the permutations of the anagram

**Banned Functions:**

`perms(), permute(), ipermute(), nchoosek(), combnk()`

**Background:**

You are Harry Potter. While studying as a second-year at the Hogwarts School of Witchcraft and Wizardry, you discover a mysterious diary. When you open it and write in it, you discover that it talks back to you! The person who claims to be talking to you through the diary is named Tom Marvolo Riddle. You know that this must be a person of great magical ability to be able to enchant such a powerful thing, but you have no idea who it is! You arrive at the conclusion that it must be an anagram for some other name. Luckily, since you have been taking classes in the wonderful magic that is MATLAB, you decide to use it in order to figure out the anagram!

**Function Description:**

Given a string, recursively determine all the unique ways to rearrange the letters. (i.e all the unique permutations). For example, if the string was `'abc'`, the list of permutations would be `{'abc','acb','bac','bca','cab','cba'}`.

Write each permutation as a separate line in a text file, in alphabetical order. Case should be ignored and all letters should be lowercase in the output text file. Non-letter characters may occur in the string and should not be included. Characters may be repeated multiple times. There should be no newline character at the end of your file.

**Notes:**

- You neglected to buy a wizarding computer from Diagon Alley that would allow you to process large numbers of recursive calls quickly. Calling the function on `'Tom Marvolo Riddle'` will take a while on your ordinary Muggle computer: do so at your own risk.

**Hints:**

- The `unique()` function will be useful
- The solution function has been enchanted to process certain magical phrases and yield a magical result.

**Function Name:** `fabricShop`

**Inputs:**

1. (*double*) A requested square footage of fabric

**Outputs:**

1. (*double*) The number of ways you could provide that much fabric

**Function Description:**

After getting beat by Severus Snape for the top tier teaching position as Potions Master at Hogwarts, you decide you won't settle for second best. Instead, you take a job at Madam Malkin's Robes for All Occasions in Diagon Alley and vow to implement your magical knowledge in a way that is sure to impress Dumbledore enough to get the position next year.

One day, the Magic Fabric Cutter 2000 breaks down and you are left to do the cutting on a muggle device, the circa 1972 Carson Procutter 220 that is only capable of cutting cloth into 16, 9, 4, and 1 square foot areas. Madam Malkin is in a fit and also at a complete loss for what to do, but you spring into action, realizing this is your big chance. You decide to write a spell (function) that is capable of determining how many ways you can provide the customer with the amount of fabric they requested. For example, if the customer wants five square feet of fabric, you could provide 5 one square foot pieces or 1 four square feet piece and 1 one square foot piece, resulting in two possible combinations.

**Notes:**

- The input will always be a positive integer.
- It would be best to assume nobody has occasions where they need a robe for their mansion, since you are still new at casting this spell and would be unable to comply (limit your input testing to something like 300 or it will take forever).

**Function Name:** `recursiveCheckers`

**Inputs:**

1. (*char*) An MxN array representing a checkerboard

**Outputs:**

1. (*double*) The number of jumps in the best move

**Function Description:**

After a near-death experience with wizard's chess, you decide to try out the simpler Muggle game of checkers. Just like in the previous homework problem `checkers`, you are the player with red pieces and it is your turn. The checkerboard array will have the capital letter `'O'` to represent empty spaces, `'R'` to represent red kings, `'r'` to represent red regular pieces, and `'b'` to represent any black piece. All pieces jump forward (up) diagonally. Kings can also jump backwards diagonally. If there is a black piece immediately across the red piece on the diagonal, and the other side is an empty space, a jump can take place.

In the previous homework problem `checkers`, only single-jump moves were considered. However, after a single jump, a checker can jump again if there is another opportunity to jump. This means that multiple jumps can take place in the same move, so long as the positions of black pieces and empty spaces allow for another viable jump to be made. Figure out the number of jumps in the red player's best move. The best move is the one with the most jumps in a single move.

**Notes:**

- You cannot jump off the board.
- Forward is always “up”.
- The checkerboard doesn’t necessarily have to be a realistic size or have a realistic placement of pieces.
- After you jump over a piece, remove the piece you jumped over from the board before figuring out if you have any more valid jumps in that move
- You do not have to account for red pieces reaching the end of the board and becoming a king.
- There will always be at least one one-jump move on the board.

**Hints:**

- Much of the logic necessary this problem is the same logic necessary to solve the original `checkers` problem. Use this to your advantage.
- Consider how you can slightly change the board before a recursive call to move towards a terminating condition where a piece has no more places to go.

**Function Name:** `collatz`

**Inputs:**

1. (*double*) Any positive integer

**Outputs:**

1. (*double*) The number resulting from the algorithm
2. *(double)* The number of recursive steps required by the algorithm

**Function Description:**

The Collatz conjecture (also known as the 3n+1 conjecture) was proposed by Lothar Collatz in 1937. The conjecture says that any positive integer n can be recursively manipulated **to be a number less than 2** by the following algorithm:

1) if the number is even, divide it by 2
2) if the number is odd, multiply it by 3 and add 1
3) repeat (i.e. recursively call the function)

Your job is to write a recursive MATLAB function that implements the Collatz conjecture. Your function should output the final number that the algorithm reaches as well as the number of recursive calls it took to get there.

**Notes:**

- If you try running this function with very large numbers (as in 40-digit numbers), MATLAB will crash as it will reach its maximum recursion limit.

**Hints:**

- You may find it useful to make a helper function to keep track of the number of recursive calls that have been made.

**Function Name:** `speedStack`

**Inputs:**

1. (*double*) The length of the base of the pyramid
2. (char) A pyramid character

**Outputs:**

1. (*char*) The created pyramid of characters

**Function Description:**

Speed Stack is a competition in which contestants stack cups in a pyramid conformation with the goal being to do so in the shortest amount of time. Since not everyone in the world has been blessed with great manual dexterity, your local Speed Stack organizers have decided to hold an alternate competition in which a pyramid with a size of their choosing must be created in the shortest amount of time possible by any means the contestants' desire. Recognizing that using a computer is an almost surefire way to win, you decide to enter the competition with MATLAB.

Write a recursive function that takes in the length of the base of the pyramid as well as any single character with which to build a pyramid (pyramid character) and outputs an array of class `'char'` that contains the pyramid. The dimensions of your array should be `<length of base>` by `<(2*length of base)-1>`. Each row should have 1 more pyramid character than the row above it and a single space (ASCII value 32) should separate each pyramid character within a row. Any other index within the array that does not contain a pyramid character should be filled with a space (ASCII value 32).

For example, for a pyramid of base length 10 and constructed from the character `'/'`, your function should output a 10x19 character array as shown below (in order to better visualize the actual construction of the array, the ASCII values for this array have been printed below with the pyramid character's value highlighted)

```
                  /
                 / /
                / / /
               / / / /
              / / / / /
             / / / / / /
            / / / / / / /
           / / / / / / / /
          / / / / / / / / /
         / / / / / / / / / /
        / / / / / / / / / / /
```

32 32 32 32 32 32 32 32 32 47 32 32 32 32 32 32 32 32 32
32 32 32 32 32 32 32 32 47 32 47 32 32 32 32 32 32 32 32
32 32 32 32 32 32 32 47 32 47 32 47 32 32 32 32 32 32 32
32 32 32 32 32 32 47 32 47 32 47 32 47 32 32 32 32 32 32
32 32 32 32 32 47 32 47 32 47 32 47 32 47 32 32 32 32 32
32 32 32 32 47 32 47 32 47 32 47 32 47 32 47 32 32 32 32
32 32 32 47 32 47 32 47 32 47 32 47 32 47 32 47 32 32 32
32 32 47 32 47 32 47 32 47 32 47 32 47 32 47 32 47 32 32
32 47 32 47 32 47 32 47 32 47 32 47 32 47 32 47 32 47 32
47 32 47 32 47 32 47 32 47 32 47 32 47 32 47 32 47 32 47

**Notes:**

- You **MUST** use recursion to solve this problem. Any function not using recursion will result in a 0 for this problem.
- Your function should work for any base length that is less than 500, since this is MATLAB's maximum recursion limit.
- The pyramid character is guaranteed to be of length 1, and the size of the pyramid base is guaranteed to be at least 1.

**Function Name:** `r_nFib`

**Inputs:**

1. (*double*) A number to begin the sequence
2. (*double*) A positive integer (n) denoting the number of terms to return

**Outputs:**

1. (*double*) A 1xN vector of the resulting Fibonacci sequence

**Function Description:**

The Fibonacci sequence is very important in mathematics, physics, nature, life, etc. Each number in the sequence is the sum of the previous two values, where the first two numbers in the sequence are always 0 and 1, respectively.

Write a MATLAB function that puts a twist on the classic Fibonacci sequence. This function will input a number to begin the sequence and the number of terms of the Fibonacci sequence to evaluate, and it will output a vector of the corresponding sequence. If the initial term is a 0 or 1, the second term will be a 1; if the initial term is any other number, the second term will be that initial number, repeated.

Therefore, one can find the sequence for 6 terms, beginning at the number 2, with the output 1xN vector of the entire sequence to be:

```
out =  [2 2 4 6 10 16]
```

**Notes:**

- You will not have any negative input values.
- You **MUST** use recursion to receive credit for this problem.

**Hints:**

- You may find a helper function.

**Function Name:** `fountainOfYouth`

**Inputs:**

1. (*char*) A character array of `'T'`s, `' '`s (spaces), and, possibly, a single `'X'`

**Outputs:**

1. (*logical*) Whether or not you can reach the `'X'`

**Function Description:**

The search for the Fountain of Youth contributed to the beginnings of overseas exploration and still remains a mystery to this day. The promise of infinite youth was coveted among explorers and is still questioned by today's most esteemed CS1371 professors.

Your job is to write a MATLAB function called `fountainOfYouth` that will determine whether you can find the hypothetical Fountain of Youth from a given array of characters. This array of characters consists of `'T'` and `' '` (space, ASCII `32`) characters, as well as a single `'X'` character that may be found near the center of the array. Your job is to check each outer layer of the array, and if you can find an opening (a `' '` character), then you can keep searching closer to the center of the array and, hopefully, reach the `'X'` that marks the spot of the desired Fountain of Youth!

A couple of sample arrays are displayed below with the openings highlighted in yellow:

| 7X7 | 8X8 |
|---|---|
| 'T' 'T' 'T' ' ' 'T' 'T' 'T'<br>'T' ' ' ' ' ' ' ' ' ' ' 'T'<br>'T' ' ' 'T' 'T' 'T' ' ' 'T'<br>'T' ' ' 'T' 'X' 'T' ' ' 'T'<br>'T' ' ' 'T' 'T' 'T' ' ' 'T'<br>'T' ' ' ' ' ' ' ' ' ' ' 'T'<br>'T' 'T' 'T' 'T' 'T' 'T' 'T' | 'T' 'T' 'T' 'T' 'T' 'T' 'T' 'T'<br>'T' ' ' ' ' ' ' ' ' ' ' ' ' 'T'<br>'T' ' ' 'T' 'T' ' ' 'T' ' ' 'T'<br>'T' ' ' 'T' ' ' ' ' 'T' ' ' 'T'<br>'T' ' ' 'T' 'X' ' ' 'T' ' ' 'T'<br>'T' ' ' 'T' 'T' 'T' 'T' ' ' 'T'<br>' ' ' ' ' ' ' ' ' ' ' ' ' 'T'<br>'T' 'T' 'T' 'T' 'T' 'T' 'T' 'T' |
| output -> false | output -> true |

Note that every other interior layer is guaranteed to be completely filled with spaces. If an opening is found, then you must recursively check the next interior layer of `'T'`s for an additional opening in order to keep moving towards the center of the given array. Once you have reduced the array to either a `2x2` or a `1x1`, check if an `'X'` is contained in the array. If an `'X'` character is found, your output for the function should be `true`. If you cannot reach the inner part of the array, or you cannot find an `'X'` once you get there, then your output is `false`.

**Notes:**

- You must use recursion to solve this problem.
- Every other interior layer is guaranteed to be completely filled with `' '` characters.
- You are guaranteed to be given a square array with `'T'` characters along the outside.
- The `'X'` character (if it exists) will never be outside of the inner `2x2` sub array.
- The character `'T'` was chosen to represent the passing of time, if you are

**Function Name:** `recursiveCampanile`

**Inputs:**

1. (*double*) The length of the sides of the bottom square
2. (*double*) The rotation angle
3. (*double*) A string of line colors

**Outputs:**

None

**Plot Outputs:**

1. A plot of a Campanile-like structure

**Function Description:**

Write a function that will draw a campanile according to the following parameters:

- The first input is the length of the sides of the square at the base of the campanile.
- The center of the base should be the origin (`x = y = z = 0`).
- The second input will be an angle in radians by which you should rotate each square, after the first, counter-clockwise.
- You will draw the campanile by drawing squares of decreasing size at increasing heights. Each square will have a side length that is 0.9 times the side length of the square below it, and it will be plotted at a distance of 1 above the square below it.
- You should stop plotting squares when the side length falls **below** 1.
- The third input is a string of color characters that you should scroll through each time you plot a new square. For example, if the string were `'rbk'`, then the first square would be red, the second would be blue, the third would be black, the fourth would be red again, and so on, repeating until the plot ends.
- Your figure should have the title 'My Campanile', and the x, y, and z axes should be labeled as `'x-axis'`, `'y-axis'`, and `'z-axis'` respectively.
- You **must** use axis equal.
- You **must** call `view(3)` at the BEGINNING of your function. Failing to do so will result in your plot being incorrect.

**Notes:**

- You **must** use recursion for this problem.
- The third input is guaranteed to only contain characters that change the **color** of the lines (no stars, dashes, etc) and can have any number of characters, and the characters may repeat.
- The counterclockwise rotation matrix is:

$$R_\alpha = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

**Hints:**

- You may find one or more helper functions extremely useful in solving this problem.