**Notice**

For this week's homework, some problems can only be tested after "loading" certain `.mat` files as instructed in the `hw05.m` file. You can load these files three ways before running test cases: right click the `.mat` file in your directory and select "Load" OR double click on the `.mat` file in your directory OR use the built-in `load()` function (see function documentation: https://www.mathworks.com/help/matlab/ref/load.html). If you are writing a test script, it would be wise to use the `load()` function appropriately. **DO NOT** use the `load()` function in your function code! To test your function(s), load the `.mat` file first and then run the function with the specified inputs just as you have been doing up until now. The Autograder will automatically load the appropriate `.mat` files for you before grading your homework. If `load()` is included in your function, it will error, and you will not receive credit.

Happy coding!
~Homework Team

**Function Name:** `castCharm`

**Inputs:**

1. *(double)* An MxN array
2. *(double)* Another MxN array
3. *(double)* The magic number

**Outputs:**

1. *(double)* The first MxN array with replacements made

**Function Description:**

For your Charms class at Hogwarts, Professor Flitwick asks you to replace some numbers in an array using magic. However, being more comfortable with a computer than with a wand, you decide to do it with MATLAB instead. Write a function that takes in two arrays and a magic number; it should replace all instances of the magic number in the first array with the *sum* of all the values in the corresponding positions of the second array. If the magic number does not exist in the first array, it should just return the original array with no changes.

**Example:**

```
magicNum = 2

arr1 =                  arr2 =
    [ 1   2   3   4             [ 4   1   3   4
      7   5   9   2               1   2   3   4
      2   2   3   5 ]             3   5   3   4 ]

arr1Out =
    [ 1  13   3   4
      7   5   9  13
     13  13   3   5 ]
```

**Notes:**

- The two input arrays are guaranteed to have the same dimensions.

**Function Name:** `quibbler`

**Inputs:**

1. *(char)* The word to search for
2. *(char)* An NxN array representing the word search

**Outputs:**

1. (*char*) The NxN array with the word "crossed out"

**Background:**

The editors of The Quibbler, the hottest magazine in print in the wizarding world, want to spice their next issue up for the readers. To make it more interactive, they are going to add a word search! As a busy student with your OWLs right around the corner, you don't want to take the time to solve these puzzles by hand, so you need to come up with a little magic.

**Function Description:**

Write a function in MATLAB that solves a word search puzzle. The function will take in two inputs: the word to search for and a character array representing the word search. You will find the occurrence of the word in the puzzle and 'cross it out' by replacing the letters with '#'. You will then return this updated array.

**Example:**

If the word to search for is '`owl`', an example input puzzle and output are below.

```
input =
                    abc                         abc
                    owl          ->             ###
                    xyz                         xyz
```

**Notes:**

- You don't have to consider diagonals, but must consider the word horizontally forwards and backwards and vertically forward and backwards (the word can read up, down, left, or right)
- The word may or may not exist in the search; if there is an occurrence there will only be one
- If your solution recognizes words that spans across several lines, i.e. recognizing '`wizard`' in the puzzle below, that is okay. This edge case will not be tested for grading.

```
abcwiz
arddef
```

**Hints:**

- You may find the `strfind()` and `reshape()` functions useful.

**Function Name:** `sortingSquare`

**Inputs:**

1. *(double)* An NxN array of positive integers (students to be sorted into a house)
2. (*double)* A double value `n`, the dimension of the square array of students to be taken out

**Outputs:**

1. *(double)* A NxN square array representing the new array of students and ghosts

**Background:**

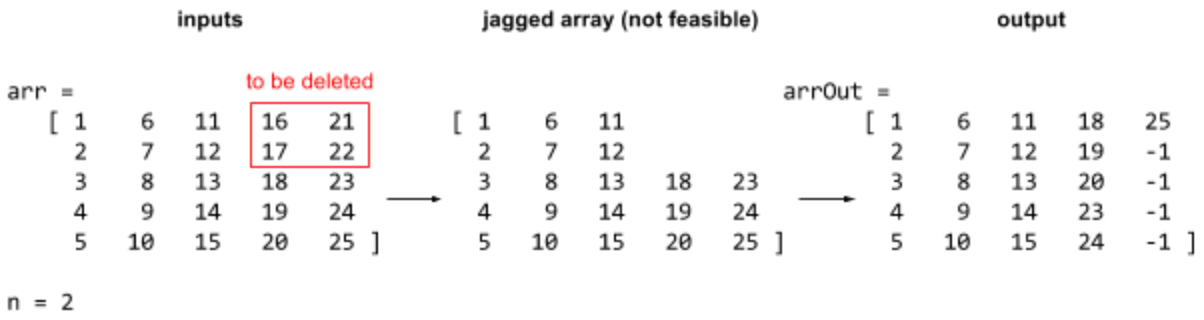
The new Hogwarts first years are waiting in an NxN array so that the Sorting Hat can put them into their corresponding houses! However, the Hat has a very unique way of picking students out of the array to be sorted.

**Function Description:**

Given a random double integer, `n`, the Hat takes out the students in a square array (size `nxn`) from the top right corner of the original NxN array.

However, this new array will most likely be jagged. The Hat would like to preserve the size and shape of the original array, so it asks ghosts to line up at the end of the array to act as placeholders for students who were just removed. The ghosts should not replace the students' positions, but be appended after the last position of the array (`end`). Ghosts will be represented by negative ones (-1). Output the final array with both students and ghosts

**Example:**




**Notes:**

- `n` can range from `0` to `N` (nothing deleted, or everything deleted)
- Students will always be represented by positive integers.
- The output array *must* be square. How do we know how many -1s to add, if any?

**Hints:**

- You may find the `reshape()` function useful for this problem.

**Function Name:** `spellCheck`

**Inputs:**

1. (*char*) An NxM array of characters

**Outputs:**

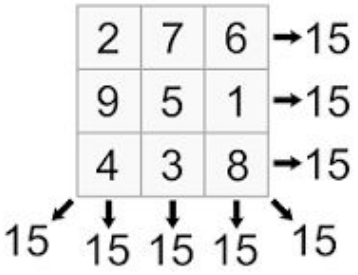
1. (*logical*) A logical representing whether the array is a magic square

**Banned Functions:**

`diag()`, `unique()`, `trace()`, `eye()`, `magic()`

**Background:**

It is up to you to help Ron complete his Charms homework. He needs to identify which spells are real, and which ones are made up. The trick is, all of the best spells can be rearranged into an array such that the ascii values will make a magic square!




As a reminder, a magic square is a matrix of numbers where the rows, columns, and diagonals all add up to the same number. Ron is able to rearrange the letters from the spell into an array, but since he is a wizard and he has no coding experience, he is counting on you to examine the arrays to figure out if they are magic or not.

**Function Description:**

Write a function in MATLAB that takes in an array of characters and checks whether the corresponding ASCII values create a magic square by outputting a logical. Remember the key aspects to check for a magic square:

1) the array must be square
2) all elements must be unique
3) all columns and rows add to the same number
4) *both* diagonals add to the same number as the rows and columns.

**Notes:**

- A 1x1 matrix is a magic square.

**Hints:**

- Think about how you can use linear indexing to get the diagonal of the matrix without using any of the banned functions.

**Function Name:** `hippogriffCase`

**Inputs:**

1. *(char)* A string of any length

**Outputs:**

1. *(char)* The modified input string, now in hippogriff case

**Background:**

Buckbeak the Hippogriff is on the run with Sirius Black after slashing Draco Malfoy during Rubeus Hagrid's Care of Magical Creatures class. Hagrid misses Buckbeak terribly, so as his friend (and Hogwarts' resident expert on MATLAB) you decide to help him out by giving him a way to receive updates on Buckbeak from Sirius!

**Function Description:**

To prevent the Malfoys from discovering you are talking about hippogriffs, you decide to create a new typing convention called `hippogriffCase`. It is similar in idea to camel case, although a little more complicated. Here are the rules for your new `hippogriffCase`:

Given a string...

1. Remove all numbers and punctuation (spaces should remain unchanged).
2. Capitalize the first letter of every word and make sure everything else is lowercase.
3. Shift the sentence around so that the LAST instance of `'hippogriff'` becomes the third word in the sentence.
4. Place any leading spaces at the end of the sentence

**Example:**

The input:

```
'Hello there, YOU 7 handsome Hippogriffs!!'
```

Would become:

```
'You  Handsome HippogriffsHello There '
```

**Notes:**

- There will always be **at least** one space before each word, except the first word, which may have none.
- There **can** be leading spaces and/or trailing spaces.
- When shifting, spaces should be considered as part of the words they precede.
- The total number of spaces in the input string should equal the total number of spaces in the output string.
- `'hippogriff'` can appear in the input string in any case (upper, lower or both).

- `'hippogriff'` may be a substring of another word. For example, `'hippogriffology'`, in which case the output word should be `'Hippogriffology'` and should be the 3rd word in the sentence.
- If `'hippogriff'` is present as a substring, then you are **NOT** guaranteed that it will be at the beginning of the word. For example, `'anti-Hippogriffs'` could appear in the input string and should be outputted as `'Antihippogriffs'`
- You are guaranteed that `'hippogriff'` will appear **at least** once in the input string.
- You are guaranteed that the last instance of `'hippogriff'` will be after the second word in the input string.

## Extra Credit

**Function Name:** `puzzleBox`

**Inputs:**

1. (*char*) A jumbled character array
2. (*double*) The row shift vector
3. (*double*) The column shift vector

**Outputs:**

1. (*char*) The solved character array

**Banned Functions:**

`circshift()`

**Background:**

Harry Potter is the "Chosen One", and "The Boy Who Lived", but even he, with all the power to defeat "You-Know-Who", is struggling to solve a simple Puzzle Box and needs your help to do it! The Resurrection Stone may be inside, so it is imperative that you help! Should you chose not to, the world may fall into darkness.

**Function Description**

Write a function called `puzzleBox` that will take in a jumbled character array and two vectors (representing a row shift and a column shift) and produce a satisfying ASCII image. The last elements in the row and column shift vectors represent the number of shifts to make, while all other numbers in the vector represent which row/columns to shift.

For example, if the row shift vector was `[3, 5, 12, 7]`, this would indicate that you should shift every element in row 3, 5 and 12 to the right by 7 columns (shifts should wrap around to column 1 if they go past the right edge of the array). Positive row shifts move elements to the right and positive column shifts move elements down. Negative shifts move in the opposite direction. Your code should be formatted such that the rows slide first, and then the columns.

**Example:**

Given the following inputs:

```
jumbledCharArr =              rowShift = [1, 3, 2];
              ['abc';...
               'def';...
               'ghi']         colShift = [2, -5];
```

You should first shift rows 1 and 3 by 2 elements to the right (wrapping elements around the end of each column). This procedure will produce the array:

```
afterRowShift =
             ['bca';...
              'def';...
              'Hig']
```

Then you should shift column 2 by 5 elements up (wrapping elements around the end of each row). Because the array is a 3x3, shifting by 5 in the upward direction is equivalent to shifting by 1 in the downward direction (**cough** `mod()` **cough**). Therefore, the final character array produced by these inputs is:

```
finalCharArr =
            ['bia';...
             'dcf';...
             'heg']
```

**Notes:**

- The shift value can be any integer.
- The index elements will always consist of valid row/column indices

**Hints:**

- You will find the `mod()` function very useful.
- Character arrays are just normal arrays in disguise.