Notice

This homework requires you to create images, which can be difficult to check using `isequal`. To mitigate this, we have given you a function called `checkImage` that will compare two images. Use this to compare the image file that your code outputs with the image file that the solution code outputs. You can look at `help checkImage` to see how to use the function.

Also, all output images should be saved as `.png` images.

Happy coding,
~Homework Team

Function Name: `swipeLeft`

Inputs:
1. (*double*) NxM array of date information
2. (*double*) Number of profiles you want in the final array

Outputs:
1. *(double*) An (input2)x(M+1) array of your top potential matches

Background:

Singles' Awareness Day (S.A.D.) is coming up, but this year you won't stand for lonely nights eating *Ben & Jerry's* and watching *Friends*. You are a smart, eligible, Georgia Tech student. You know that it's time to get technology on your side, so you downloaded the "social discovery" app, Tinder, to get your search started.

It comes as no surprise when you're getting matches left, right and center - you expected this - but you are still finding yourself a little bit *lost in the sauce*. There is only one thing that you trust to help you make these difficult decisions on how to swipe: MATLAB! You arrange all of your matched profiles' data into different categories in an array. Now all that's left to do is write a function to thin the candidate pool and find out who gets one step closer to being your MATLAB soulmate and who's getting *YEETed* this S.A.D.

Function Description:

Scores on the profiles you have matched with are in an array, where each row represents a different candidate and each column is a category. Find the average score for each candidate and concatenate the averages to the right of the array. Sort the array by the averages with the highest average on top, and extract the number of profiles given in the second input.

For example, if the second input is 3, the output array should be the top 3 profiles with the highest average on top.

Notes:
- Round the average score to the nearest whole number before sorting or concatenating.
- The number of profiles given in the array will never be less than the number of profiles you are trying to output.
- No two profiles will have the same average.

Hints:
- The second input and second output of `sort()` will be useful.

Function Name: `loveSearch`

Inputs:

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

Outputs:

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

Background:

You've been searching for love for a while but haven't had any luck. Maybe you have been searching in all the wrong places… so why not try searching in a word search? Even better - you can use MATLAB to help!

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 '`date`', an example input puzzle and output are below.

```
input =                 dklr                    dklr
                        date        ->          ####
                        gfnj                    gfnj
                        cqrw                    cqrw
```

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 exist once or not at all.
- If your solution recognizes words that span across several lines, i.e. recognizing '`valentine`' in the puzzle below, that is okay. This edge case will not be tested for grading.

```
abcvale
ntinelw
```

Hints:

- You may find the `strrep()` and `reshape()` functions useful.
- Think about how you can use transposing and linearizing to check for the word in the different directions.

Function Name: `beMemeValentine`

Inputs:

1. (*char*) The filename of the main image, including the file extension
2. (*char*) The filename of the background image, including the file extension

Outputs:

None

File Outputs:

1. The edited image

Background:

Valentine's Day is right around the corner, and you realize you don't have anything for the cutie in your 1371 recitation! Good thing you have MATLAB to confess your love in the best form - memes.

Function Description:

Write a function that replaces the background color of the main image with the corresponding pixels in the background image. Resize the background image to the size of the main image. To find the background color of the main image, find the RGB values of the top left pixel. The name of the output image should be the original name of the main image before the file extensions with `'_Meme.png'` appended.

Notes:

- Resize the images before masking.
- The input filenames will always have the extension `'.png'`
- The solution function will output a file with the original name, minus the file extension, with '`_Meme_soln.png`' appended.

Hints:

- The `imresize()` function will be useful.
- Try putting two names as the third and fourth inputs in the solution function! (You don't need to implement this part into your function.)

Function Name: `fourEverAlone`

Inputs:

1. (*char*) The filename of an image, including the file extension

Outputs:

None

Image Outputs:

1. The edited image

Banned Functions:

`rgb2gray(), mat2gray(), ind2gray(), flip(), fliplr()`

Background:

Sadly, you realize you'll be alone on this Valentine's Day… until you remember you have the only bae you really need: MATLAB. However, still in need of human company, you shed a tear and decide to write a program to express your loneliness in the form of 4 edited images.

Function Description:

Write a function that takes in the name of an image and edits it in the four ways described below, and concatenates it into one image.

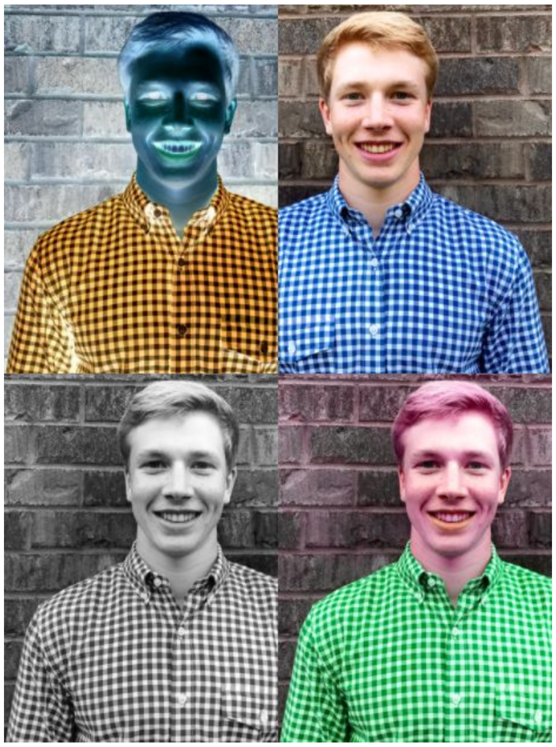
1. Top right: the mirror image of the original image
2. Top left: the negative of the original image
3. Bottom left: the grayscale of the original image
4. Bottom right: the original image with the green and blue layers swapped

The final image should have twice the number of rows and twice the number of columns compared to the original image. The name of the output image should be the original name before the file extensions with `'_Alone.png'` appended.

Example:

`addison.png`

Continued...



`addison_Alone.png`

Notes:

- To find the grayscale of the image, take the average of the three color values for each pixel by adding first and then dividing.
- The solution will produce files with the original name, minus the file extension, will `'_Alone_soln.png'` appended

Function Name: `magicSquarePair`

Inputs:

1. (*double*) An LxK array
2. (*double*) An NxM array

Outputs:

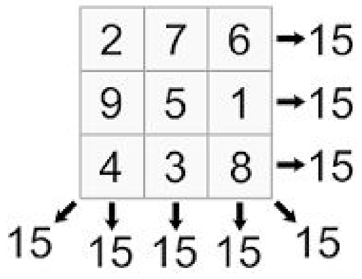
1. (*logical*) A logical representing whether both arrays are magic squares

Banned Functions:

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

Background:

Your friend set you up on a blind date. Will there be magic between you two? Or will the evening crash and burn? Luckily for you, you realize you can use MATLAB to see if there is a connection. You and your date select nine of your favorite numbers and arrange them into arrays as shown. According to the rules passed down by the ancients, any two who both select magic squares are forever entwined in the realm of love. You decide to see if both arrays are magic squares using MATLAB.



Function Description:

Write a function in MATLAB that takes in two arrays and checks whether they are both magic squares 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.
- A helper function might be useful.

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:

You planned to get your special pal a box of chocolates for Valentine's day but forgot until the last moment! Since the stores are sold out of chocolates, you decide to create a special puzzle box to give as a present. And you can use MATLAB to help you do it!

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.