Notice

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

Also, all output images should be saved as `.png` images, even if the input images are of another image type (such as `.jpg`).

Happy coding,
~Homework Team

Function Name: `memeGenerator`

Inputs:

1. *(char)* Top text
2. *(char)* Bottom text
3. *(char)* Name of the original image
4. *(double)* 1x3 RGB triplet representing text color

Outputs:

none

File Outputs:

1. A dank meme

Function Description:

Memes are the bread and butter of online communication today, so you've decided to expedite the process of creating your own memes by writing a MATLAB function to do it for you! Using the helper function `str2img` (provided), you will convert a string into an image of the text and resize it to match the width of the original image.

The top text (first input) should be placed across the top of the original image and the bottom text (second input) should be placed across the bottom of the original image. The image produced by `str2img` will have pure white text and the background will be pure black. When putting the text on the original image, the black background should be replaced with the corresponding pixels on the base image and the white text should be replaced with the color specified by the fourth input.

The output image filename should be the same as the input image name with `'_meme'` appended to the end.

Notes:

- The output from `str2img` is not guaranteed to have the same number of columns as the original image, so left justify the text when placing it in.
- None of the values in the fourth input (color) will ever be 0.
- See `help str2img` for details on how to use this function.

Function Name: `fifteenPuzzle`

Inputs:

1. *(char)* Filename of an image of a 15 puzzle

Outputs:

1. *(char)* The move to make in order to solve the puzzle

File Outputs:

1. Image of the solved 15 puzzle

Function Description:

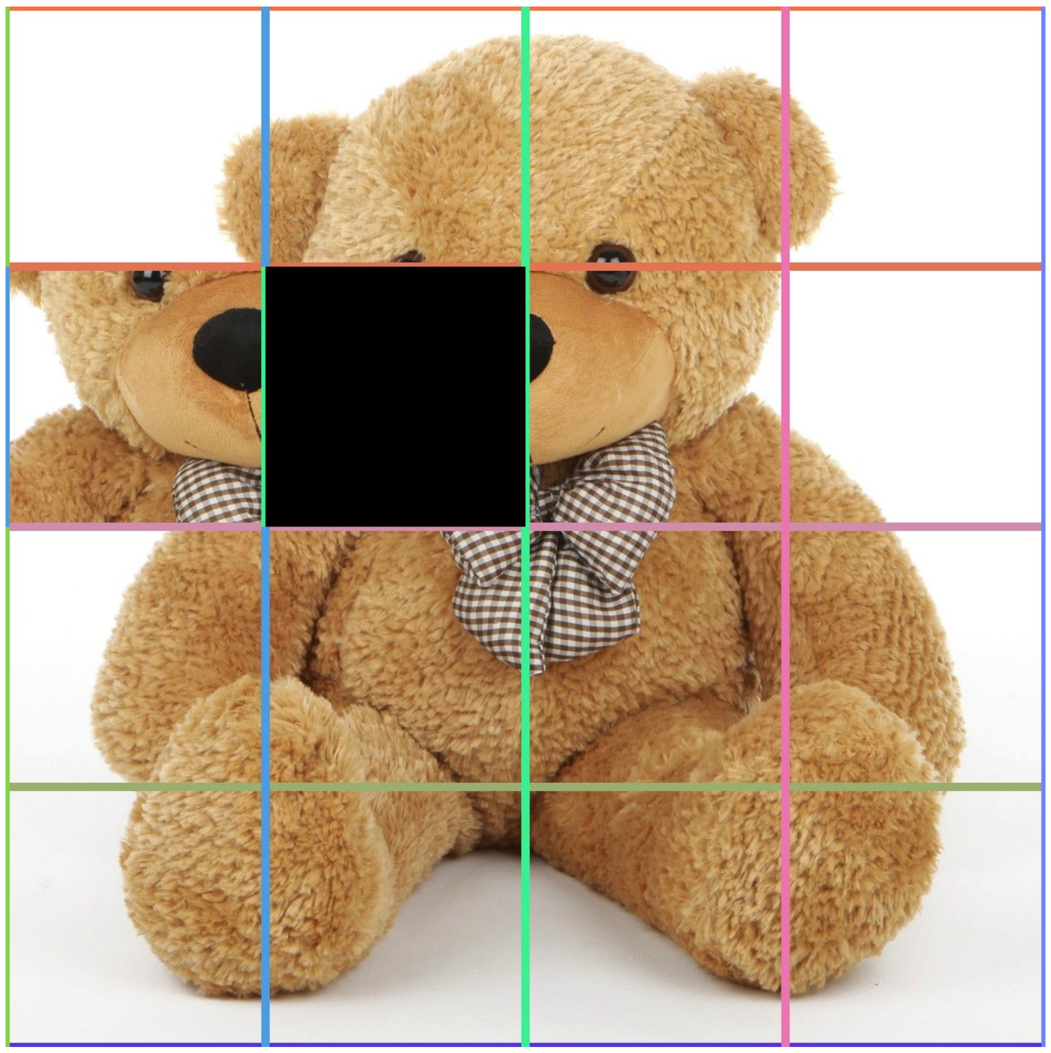
The 15 puzzle is a sliding square puzzle that you may have commonly seen as a children's game. It consists of a 4x4 box with 15 square tiles placed in all but one of its 16 sections. At first, the tiles are arbitrarily arranged, and the goal is to slide them into the correct configuration, revealing the hidden picture.

For this problem, you will be given an image of a 15 puzzle that is one step away from completion. Write a MATLAB function called `fifteenPuzzle` that takes the image's filename and outputs a string of what move needs to be made to solve the puzzle. In addition, you should actually solve the puzzle and write the new image to a file with `'_solved'` appended before the file extension.

To help solve the puzzle, each image will be formatted in the following way:

- The image is guaranteed to be square and divisible by 4 in both length and width.
- There are 10 total colored lines: 6 “inner” lines that divide the sections and 4 border lines that surround the entire image.
- No two lines will be of the same color.
- Each inner line is 2 pixels thick.
- Each border line is 1 pixel thick.
- In other words, each of the 15 tiles will have a 1-pixel border, with each side being a different color.
- The 16th, blank tile will be represented as pure black without any colored borders.
- See the image on the next page for an example. The line thicknesses are exaggerated for easier viewing.

Therefore, if you can correctly match the colors, you can figure out which tile needs to be moved to solve the puzzle. To describe this movement in your output string, indicate which direction the *blank* tile will move into. Use `'L'` for left, `'R'` for right, `'U'` for up, and `'D'` for down. So, for the puzzle on the next page, your output would `'L'`.



We have also provided a `puzzleMaker` function so that you can make your own test cases. To use it, input the filename of an image, and it will create a new file with `'puzzle'` appended to the end.

Notes:

- The blank 16th tile can end up in any section of the solved puzzle.
- Some line separators may appear to be the same color, but do in fact have different RGB values.

Easter Egg:

- “Why is it called Puzzles?”

Function Name: `beeMovie`

Inputs:

1. *(char)* The name of a file containing an image of a bee
2. *(double)* A vector of integers from 1 to 4

Outputs:

None

File Outputs:

1. Image with bees overlaid

Function Description:

Inspired by some of your favorite memes, you decide to write a MATLAB function that takes in an image of a bee and recursively overlays it with smaller copies of itself, so that each bee image is overlapped by a smaller one. In other words, a picture of a bee but every time there's a bee there's another bee.

Your function will take in the name of a file containing the original image, and a vector of numbers from 1 to 4 specifying which quadrant of the image will be overlayed with the smaller images. The quadrants are numbered like the math coordinate plane, with the upper right as the 1st quadrant and the numbers continuing counterclockwise. The first number in the vector identifies the quadrant of the original image where the sub-images will be overlayed. The second number identifies the quadrant of that sub-image where the next sub-image will be overlayed, and so on. Continue overlaying images until you run out of vector elements.

Finally, you don't want to cover up the entire quadrant of the image as you overlay, so you must crop out the background of each sub-image before you overlay it. The background of each image will be pure white (RGB: `[255 255 255]`).

Write your completed image to a new file, with `'_swarm'` appended before the file extension.

Notes:

- Use `imresize()` to resize your image to fit in each quadrant.
- Remove the white background *after* you resize each image.
- The cutoff point for the top half should be `floor(rows/2)`, and likewise for the columns.

Hints:

- Use the solution function to visualize what this function does. You can easily create your own test cases using the provided images.

Function Name: `imagePoker`

Inputs:

1. *(char)* Filename of an image of a poker table
2. *(struct)* Structure containing images of all the ranks

Outputs:

1. *(cell)* List of cards

Function Description:

After some mixed luck with poker, you decide that you want to improve for your next trip to the casino! You decide to write some code in MATLAB that will allow you to analyze poker games in real time. The first step is to determine what cards are on the table. Given an image of a table and some cards, determine what cards are on the table. Each card is guaranteed to be a pure white rectangle and the only pure white on the table is guaranteed to be part of a card.

Each card will also have two images on it that determine its rank. One image will be in the upper left. Another image, rotated 180 degrees, is located in the bottom right. You have been provided a structure array called `cards`, where each structure represents a rank. Each structure contains a `Name` field containing each rank's name (`two, three, Jack`, etc.) and an `Image` field that contains the image for that rank.

Find where each card is in the image and determine if it is a valid card by comparing the images in the bottom right and top left. If they correspond to the same rank, determine what rank the card is by comparing it to the images stored in the structure array. Output a cell array containing the ranks of all the cards present, sorted alphabetically.

Notes:

- Each card will be surrounded by at least one non-white pixel on all sides.
- The images that represent rank are not necessarily the same size as the images stored in the structure, but if they are the same when scaled to the same size, they should be considered equal.
- The top left rank image will be further up and further left than the bottom right image.
- The rank images will not have any pure white on them.
- If there are no valid cards, the function should output a 0x0 cell array.
- Cards will never overlap.
- The background can be any variety of colors, but will not have pure white on it.

Hints:

- Think about the pixels adjacent to the top left and bottom right pixels of each card.
- The `imresize()` function will be useful.