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.

Remove all uses of `imshow()` in the code that you submit. You are encouraged to use it to debug, but you must remove it before submission. Any code that uses a function that displays the image in a window, such as `imshow()` or `image()`, will automatically receive zero points.

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

Happy coding,
~Homework Team

Function Name: `cmyk2rgb`

Inputs:

1. (*double*) NxMx4 array containing CMYK pixel data of an image

Outputs:

1. *(uint8*) NxMx3 array containing the RGB data of the same image

Background:

You are a field agent enlisted by the top secret spy agency MI7. While having biscuits and tea at your favorite cafe, your super-smart spy phone receives a ping. You've just received a mission! Command requires that you "take care of" an enemy spy! (And I don't mean take them out to dinner...) The only info about this spy received was a photo, but it's highly encrypted in CMYK! The good news is that your smartphone can run MATLAB, so you are able to use your intense spy-coding skills to decrypt the photo before going after the target.

Function Description:

CMYK is an alternative color scheme to RGB (It's what printers use to save colored ink). The layers of CMYK represent cyan, magenta, yellow, and black (in that order) and its values are between 0 and 1. To convert between the two, you first incorporate the black layer into the first three using the following formula:

$$\text{Layer} = \text{Layer} \cdot (1 - K) + K$$

where K represents the black layer. Then, delete the black layer. The array should now be NxMx3. Then, you will need to convert from CMY to RGB. First, multiply the image by 255 to scale the values from between 0 and 1 to between 0 to 255. Each of CMY's layers correspond with the matching RGB layer, but are inverted (Cyan is the absence of red, magenta is the absence of green, yellow is the absence of blue), so take the negative of your image. Then, cast to uint8, and output the resulting image.

Notes:

- After you have an NxMx3 array, use imshow() while testing your code to visualize what's happening.
- Make sure that your output is a uint8.

Function Name: `nightVisionGoggles`

Inputs:

1. (*char*) A filename of an image

File Outputs:

1. A new image file in "night vision mode" with `'_spy'` appended before the file extension.

Background:

You are a spy. You need to see in the dark.

Function Description:

Turn an image into "night vision mode" by doing the following:

- Take the average of the three layers and set the result to be green layer. Use the following formula for finding the average

```
uint8((double(redValues) + double(greenValues) + double(blueValues))/3)
```

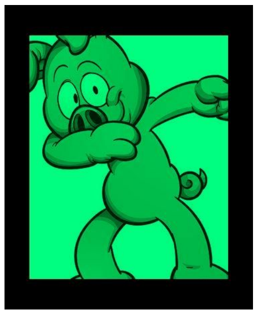
- Set the blue layer to half of its original values.
- Set the red layer values to zero.
- Finally, "focus" in on your target by eliminating unnecessary background. Set 1/10th of the rows and columns to black on each side of the border.

After following these steps, write the image to a new file with `'_spy'` appended before the file extension.

Example:

`'Example.png'` →

`'Example_spy.png'` →



```
>> nightVisionGoggles('example.png')
```

Notes:

- When figuring out 1/10 of the rows/columns for the border, round to the nearest integer.

- You should change 1/10 of the rows/columns on *each side* of the image. For example, if the image array is 100x100, the rows 1-10, rows 91-100, columns 1-10, and columns 91-100 would need to be black.

Function Name: `camouflage`

Inputs:
1. (*double*) A 1X6 vector representing ranges of values in the red, green and blue layers
2. (*char*) The filename of the image to be edited
3. (*char*) The filename of the "donor" image

File Outputs:
1. An image file with `'_camo'` appended to the name given in the second input before the file extension

Background:

Like they say, it's hard out here for a spy. Spies often need to blend into the night, and thankfully, you have just the right MATLAB skills to help them do that.

Function Description:

Write a function that replaces certain pixels in the first image with the corresponding pixels in the donor image. To do this you will first, resize the donor image so that it is the same size as the image to be edited. Then, given a vector that contains a minimum and a maximum value for a range for each of the red, green and blue layers (in that order) in an image, replace any pixels in the first image for which all three layers' values are within the specified ranges with the pixels in the corresponding positions in the donor image.

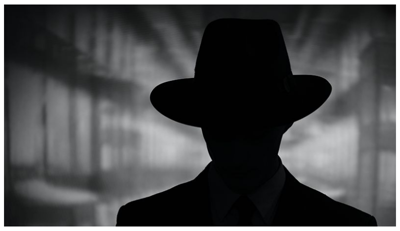
Notes:
- The color value ranges are inclusive of the minimum and maximum values
- The vector input will always be in the form:

```
[redMin, redMax, greenMin, greenMax, blueMin, blueMax]
```

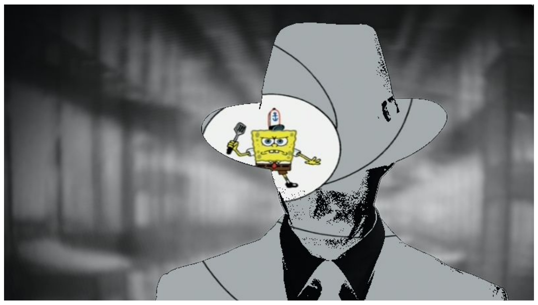
Example:

```
camouflage([0 5 0 5 0 5], 'spy1.png', 'spydonor1.png')
```

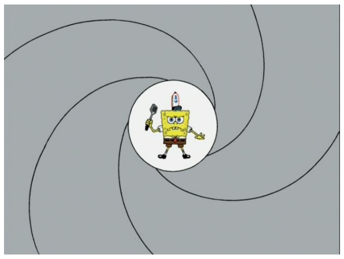
`'spy1.png'` →



→



`'spy1donor.png'` →



Function Name: `disguises`

Inputs:

1. (*char*) An image filename containing the image you will disguise

File Outputs:

1. An image file named with `'_disguised'` appended to the original name before the file extension

Banned Functions:

`rgb2gray(), mat2gray()`

Background:

As a spy, one of the most important things is keeping your identity a secret! This means lots of foolproof, high quality, spy quality disguises. You decide to make a compilation a few different ways to disguise images of yourself in MATLAB!

Function Description:

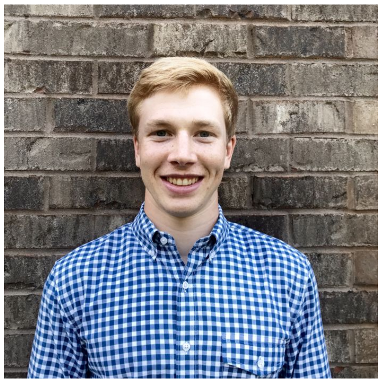
Write a function that makes a single image made up of a collage of four different versions of the original image. After you have created these versions, place them according to the following order. The new image should be the same size as the original image. Follow these guidelines for each quadrant of the image, then resize as the final step after concatenation.

- Upper left: vertically flip the image. Then, change the white pixels (ones that have red, green, and blue values greater than 200) to be pure green.
- Upper right: Swap the red values and the blue values of the image
- Lower left: Grayscale the image, using the following formula:

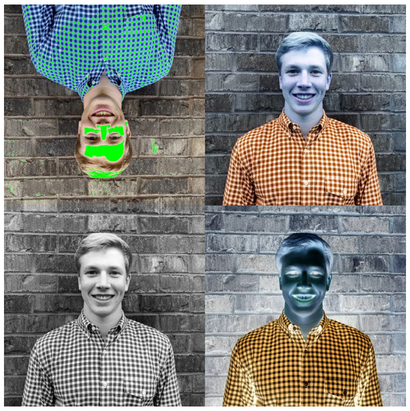
```
uint8((double(redValues) + double(greenValues) + double(blueValues))/3)
```

- Lower right: Take the negative of the original image

Example:



```
Addison.png               (left)
Addison_disguised.png    (right)
```



Function Name: `secretDocs`

Inputs:

1. (*char*) The filename of an image
2. (*double)* An MxN array of the correct positions for the pieces of the image

File Outputs:

1. A file containing the correctly reassembled image with `'_recovered'` appended after the original file name but before the file extension.

Background:

You have just infiltrated the secret evil base of u[sic]ga in order to obtain the documents they need back at HQ. As you crawl out of the air duct into the archive room and begin to sneak around, you spot the filing cabinet you were looking for. “Bingo,” you whisper to yourself triumphantly. You pick the lock and gently open the drawer to find… Oh no! It was booby trapped, and a hidden device has shredded all the documents you needed to acquire! No worries, you still have a secret gadget left in your arsenal that can help you save the mission: MATLAB.

Function Description:

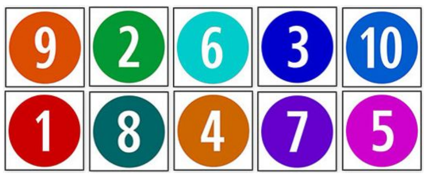
Given a shredded image and an array of the correct positions of the pieces, break up the image and reassemble it according to the new positions. In order to do this, you will need to find the size of the position array, and break the image into that number of pieces. Once you have done this, you can convert it back to an array and write it to a new image file with `'_recovered'` appended to the original name.

Example:

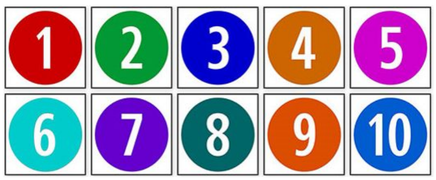
```
positions =
      [6 2 4 8 10
       3 9 7 1 5 ];

secretDocs('numbers.png', positions)
```

`numbers.png =>`



`numbers_recovered.png =>`



Continued...

Notes:

- The dimensions of the image will always be divisible by the dimensions of the index array.
- The numbering for the position array goes along each of the rows first, then down the columns- this is opposite from how regular linear indexing works.
- The number at each index of the position array represents what number piece should go in that index. For example, if the first number in the position array is a 6, that means go to the 6th piece in the scrambled image and put it in the first position.

Hints:

- One way to keep track of the pieces is to create a cell array, where each piece of the image is contained within its own cell. You can then use this cell array to apply the array of positions.
- Think about how you can use the positions in the array as linear indices.