Assignment 3

Note: You do not need, nor should you use loops on this assignment.

Your mission is to edit A3.py so that it performs as indicated below.
In this assignment you will create new functions in Python for processing simple binary images with a single bright object in the foreground, and a dark background. The foreground has intensity 1, and the background intensity 0, so at times it is convenient to think of the image as a matrix of logical (true/false) data. The images for this assignment came from the Laboratory for Engineering Man/Machine Systems at Brown University.

Below left is a binary image that shows a bright square in the dark background.

1. Scrolling the Image

Much of the animation in old video games, like Pacman and Super Mario Brothers, consists of a foreground image scrolling across a fixed background. Above right is what the box image looks like after scrolling left and right, respectively. You can make the foreground in a binary image scroll by clever array indexing.

For this part of the assignment, you will create four different functions. Each function will take two parameters; the first is the binary image data, and the second is a number of pixels. Each function will return one value: the new binary image that is the same size as the input image. The names of the functions should be: **`scrollLeft`**, **`scrollRight`**, **`scrollUp`**, **`scrollDown`**, and they should cause the foreground of the image parameter to move in the appropriate direction by the specified number of pixels. You may assume that the number of pixels specified in the second parameter is always smaller than the size of the image in the relevant direction. You should also assume that the background extends to infinity. When part of the image scrolls off of one side, you should create more 0s on the other side, so that the output image is the same size as the input image.

Prototypes for these functions are included in the template for this assignment. After writing your functions, test them on the image provided with the assignment. Use your functions to produce the following results:

Test image: test.png

1A. Scroll the image up by 10 pixels.

1B. Scroll the result of part 1A down by 10 pixels.

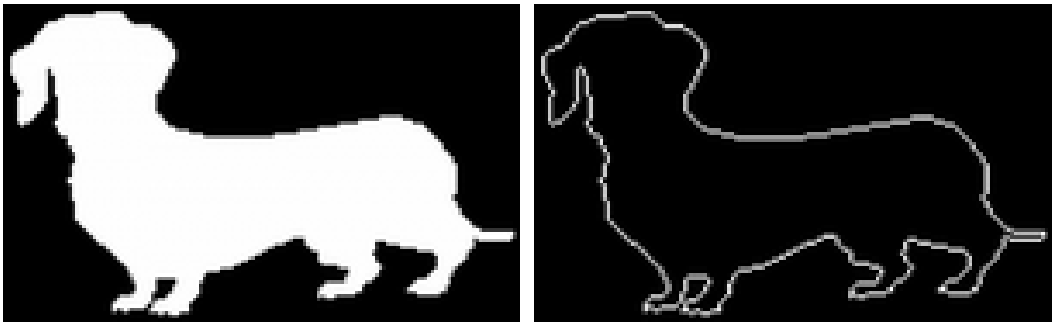
1C. Does the second result look the same as the original image? Why or why not? Answer in the template.

1D. Scroll the original image left by half of its width.

1E. Scroll the result in 1D back to the right by the same amount.

2. Boundary Detector

A common image-processing task is to identify the boundary (outline) of an object. Conceptually the boundary is the set of all pixels in the foreground that are adjacent to the background. Here is an image of a dachshund and another image of its boundary.



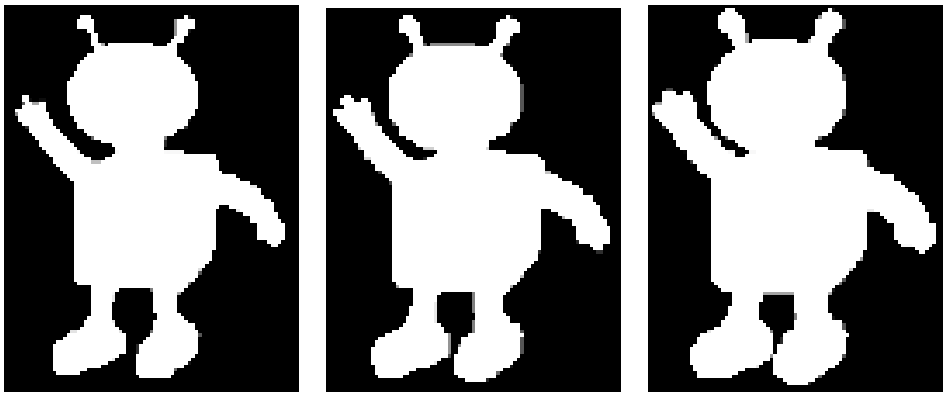
We can use the scrolling functions from the first part of this assignment to determine if a pixel is on the object boundary or not. Suppose that image is **`I`**, and **`LI`** is the result of the command: **`LI = scrollLeft(I, 1)`**.

A pixel, **`[i,j]`** is on the right edge of the boundary if and only if **`I[i,j]`** is foreground and **`LI[i,j]`** is background. You can use the other scrolling functions from part 1 to identify the pixels on the other sides of the boundary. Combine these tests together with the correct logical operator, and you will be able to identify the entire boundary of the object. For this part of the assignment you will write a function that has one input parameter: a binary image, and returns a single value, a new binary image, where only the edge of the original object is in the foreground (has intensity 1). (The boundary images you produce will have only black and white intensities.) You should name this function **`findBoundary`**.

For this part you should save an image 2.png that shows the boundary of the test image.

3. Dilate and Erode

Two other famous image processing operations are dilate and erode. Dilate makes the foreground grow into the background, and erode makes the foreground shrink, as if the background were washing it away. Here is an example showing the dilation and erosion of an image of a robot. The original image in the center is from the collection at Brown. The eroded image is on the left, and the dilated image is on the right.



It is easy to erode an image – just change the intensity of the boundary from 1 to 0. To dilate an image, you will need to write a new logical test to find pixels that are in the background of the original image, but are adjacent to a foreground pixel. For this problem you need to write two functions, each of which takes an image as an input, and returns a new image as its output. Name these functions **`dilateImage`** and **`erodeImage`**.

3A. Dilate the original image, and then dilate that result, and then dilate that result as well. Save the final result. Does it look like the same type of object as the original?

3B. Erode the original image, and then erode that result, and then erode that result also. Show the final result. Does it look like the same type of object as the original?

3C. Take the image you just produced – the result of three consecutive erosions. Now apply three consecutive dilations to it. Show the final result. Is it the same as the original?

3D. Find the boundary of the original image, and then dilate the boundary image. Save the result.

3E. Take the original image, dilate it once, and then find its boundary. Save the result.

3F. Describe the difference between the two images in 3D and 3F.

Things to watch out for:

Global Variables Just say no to global variables. You should NOT be referring to **`test_image`** in your function bodies. You should only operate on parameter **`img`** in the function. If you use **`test_image`** in your function body you won't get the right answer when you try to apply it to other images. I've also seen this mistake with boundary; several of you are reusing it in your erode function. You have to compute the boundary of the **`img`** parameter which will be different from **`test_image`** in later steps.

Comparisons to 1 or 0 Don't compare these images to 0 or 1 like **`img==1`** or **`img!=0`**. There is no need. These things are already **`True`** or **`False`**.

Adding Images Don't combine images (boundaries for example) using addition. You must use **`np.logical_or`** to combine them. Addition will look sort of like it works but it doesn't do the right thing.

Clearing Pixels The trick to clearing all the pixels on the boundary is to use **`np.logical_and`** with the complement of the boundary image.