Python Assignment 3 – Images

(Note: You may work with one partner (team of 2) if you wish.)

Question 1 – Add Borders

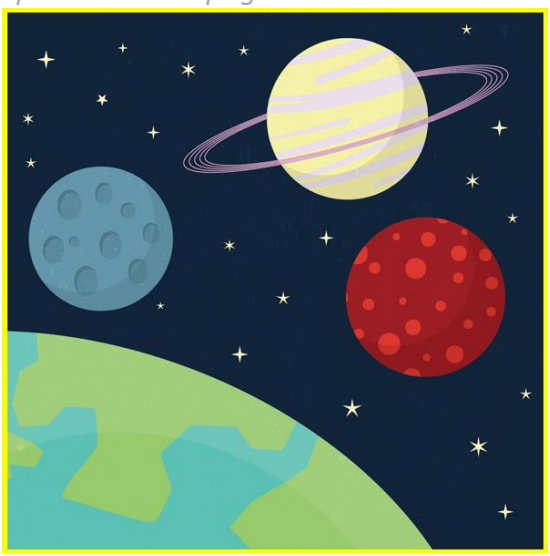
```
addBorders(imageFileName, thickness, colour, outputFileName)
```

Example

```
addBorders("space.png", 5, (255,255,0), "spaceBordered.png")
```

space.png

spaceBordered.png



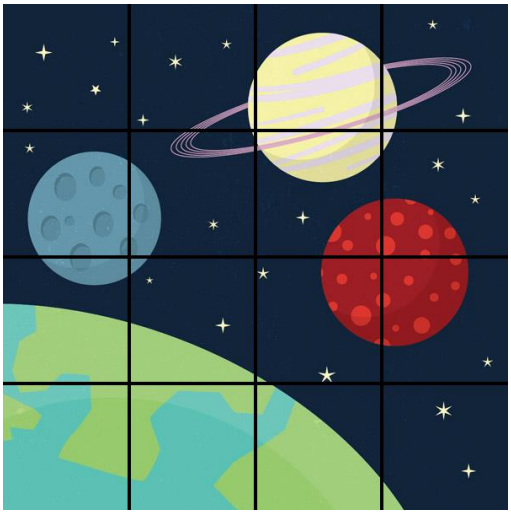
Question 2 – Add Dividers

```
addDividers(imageFileName, rows, cols, thickness, colour, outputFileName)
```

You might find it easier to get it working with rows == cols, then get it to work more generally.

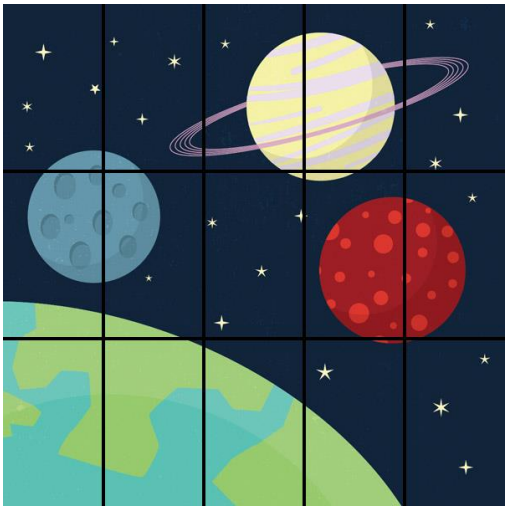
Example – 4x4

```
addDividers("space.png", 4, 4, 2, (0,0,0), "space44.png")
```



Example - 3x5

```
addDividers("space.png", 3, 5, 2, (0,0,0), "space35.png")
```



Question 3 – Create Image from Binary

```
createImageFromBinary(sourceFileName, targetFileName)
```

Background

Sample images are stored in two text files - "pixels_small.txt" and "pixels_big.txt". Each file contains one line, which is all 1's and 0's. Each set of 32 bits represents one pixel. (You can open the small one in a text editor, but the big one is very, very big and might not load.)

Structure of the Binary Data

- The first 32 bits is the width of the image to be constructed. (This is metadata, not part of the image itself.)
- The second 32 bits is the height of the image. (Also metadata)
- All remaining bits, in sets of 32, are pixels. For each set of 32 bits:
    - The first 8 bits are the alpha (opacity). You can ignore this because the images we're using are fully opaque.
    - The second 8 bits are the red value.
    - The third 8 bits are the green value.
    - The last 8 bits are the blue value.

Example Pixel

The string "11111111110110101010000000100000" is an orangey-yellow pixel. Here's why:

- Alpha: 11111111 (255)
- Red: 11011010 (218)
- Green: 10100000 (160)
- Blue: 00100000 (32 – note the leading 0's)

Expected Output Image from "pixels_small.txt"

Expected Output Image from "pixels_big.txt"

(actual size is bigger)

Question 4 – Save Image as Binary

```
saveImageAsBinary(sourceFileName, targetFileName)
```

Create a text file containing a single string. That string contains the binary encoding of an image, in the same format as the files "pixels_small.txt" and "pixels_big.txt". (Make sure you include the image's width and height in the first 64 bits.)

You can use any images you like, but "teapot.jpg" and "road.jpg" are included as samples of small and big images, respectively.

Question 5 – Unscramble 4

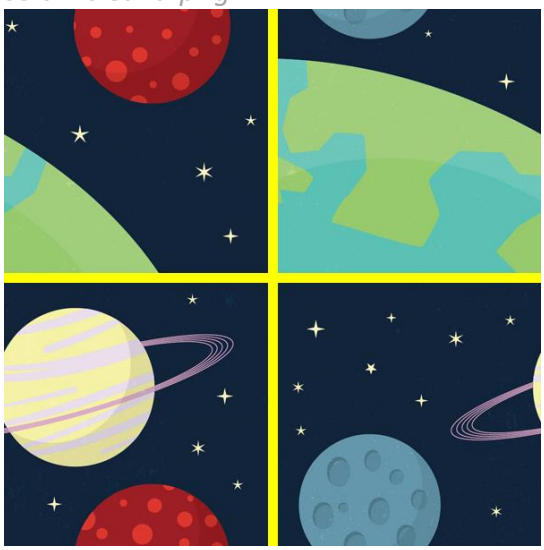
```
unscramble4(imageFileName, outputFileName)
```

An image with a yellow border has been divided into four equal regions, and those regions have been scrambled (rearranged) and saved to a new image. Your job is to unscramble the new image. (Find the yellow borders to tell how to re-arrange the regions.)

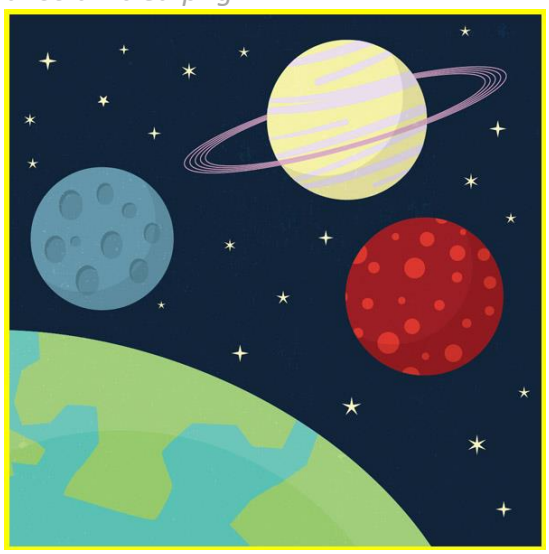
Example

```
unscramble4("scrambled4a.png", "unscrambled.png")
```

scrambled4a.png



unscrambled.png

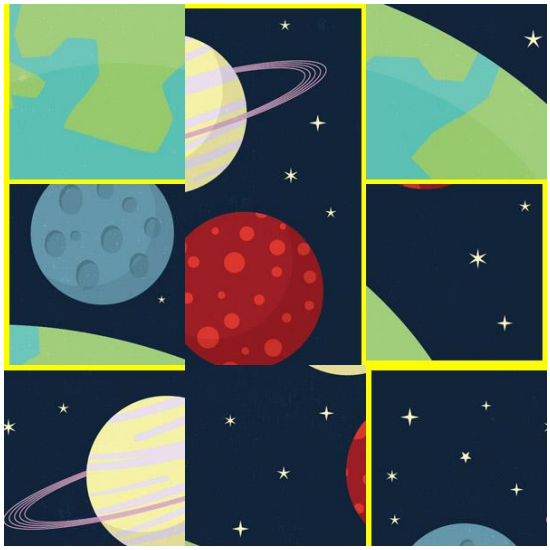


Question 6 – Unscramble 9

unscramble9(imageFileName, outputFileName)

An image with a yellow border has been divided into nine equal regions, and those regions have been scrambled (rearranged) and saved to a new image. Your job is to unscramble the new image. (Find the yellow borders to tell how to re-arrange the regions.)

scrambled9a.png



unscrambled.png