COMP 208
Winter Semester 2021

INSTRUCTOR: DR. CHAD ZAMMAR
chad.zammar@mcgill.ca

Assignment 4: Exploring Matrices.

Due date: 11 April 2021, 11:59 PM.

Before you start:

- Collaboration and research for similar problems on the internet are recommended. However, your submission should reflect individual work and personal effort.
- Some of the topics may not be covered in class due to our limited time. You are encouraged to find answers online. You can also reach out to your instructor or TAs for guidance.
- Please submit your assignment before the due date to avoid penalties or worse risking your assignment being rejected.
- Submit 2 files called **assignment4.py** containing all the functions together with their implementations.

Make sure your code is clear and readable. **Readability of your code as well as the quality of your comments will be graded**.

- No submission by email. Submit your work to mycourse.
- If your code does not run (without interpreter errors) it will not be graded.
- Be happy when working on your assignment, because a happy software developer has more inspiration than a sad one :).

Image Analysis with Python

For the following assignment you need to install the **scikit-image** package which provides the submodule **skimage.io** that you will be using to read and display an image from disk.

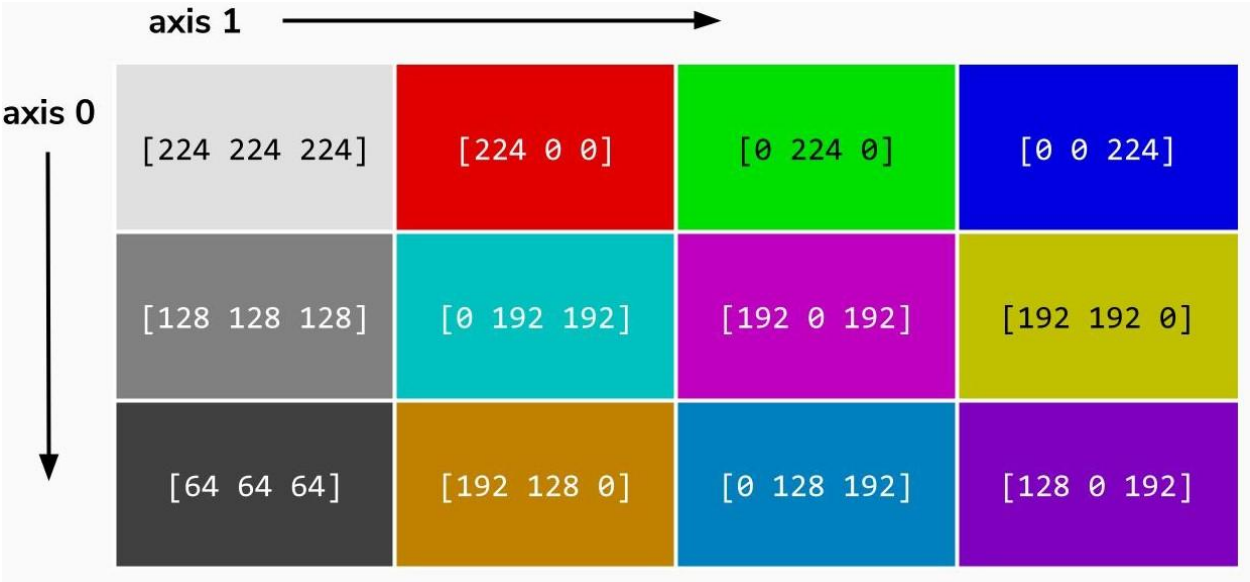
For example, to read an image you can use the following code:

```
import skimage.io as io

# read image into memory
image = io.imread("image.jpg")

# imread() returns a NumPy array,
# with shape of (height, width, 3)
print(image.shape) # this will print something like: (1200, 800, 3)

# Display image to screen
io.imshow(image)
io.show()
```

Images are composed of thousands of pixels, laid in the form of a table or matrix (with rows and columns). The width of an image is the number of columns, and its height is the number of rows.

For a color image, each pixel in the matrix consists of three integers, one for Red, one for Green and one for Blue, or RGB.

Or also you can picture the image pixel layout like this:



When an image is displayed, the three integers are mixed together in order to create the colour we see. Each of the three integers is within the range [0, 255]. E.g., a completely blue pixel will have the value (0, 0, 255). White is defined as (255, 255, 255), and black is (0, 0, 0) (absence of colour).

For example, to change the color intensity at a specific pixel, you can simply assign the matrix at the pixel's location to the desired color:

```
#assuming you have an image called image.jpg
>>> image = io.imread("image.jpg")
>>> print(image[150, 300])
[121 159 146]

# Change the green intensity at (150, 300)
>>> image[150, 300, 1] = 255
>>> print(image[150, 300])
[121 255 146]

# Change the color at pixel (150, 300) to magenta
>>> image[150, 300] = [255, 0, 255]
>>> print(image[150, 300])
[255   0 255]
```

For this assignment, you are provided with an image (mountain.png). This image is actually hiding another image inside it. Your mission, as a Comp208 detective, is to retrieve the hidden image and then implement some utility methods.

Let's implement the software step by step:

Step 1 (10 pts)

First you need to create a class called **ImageAnalysis**.

1. This class will take an image name as a constructor argument and will assign the numpy array of the image to a member attribute called **image**.
2. The class will also implement a method called **show**. When **show** is called the image will be displayed on the screen.
3. If you try to print any object of type ImageAnalysis, the shape of the image should be printed out to the screen.

Example code:

```
i = ImageAnalysis("mountain.png")
print(i) # this will print the following: Shape is: (1512, 2016, 3)
i.show() # this will open a window showing the picture
```

Step 2 (25 pts)

Add a method called **retriveHidden** that will retrieve the concealed image within the mountain image.

1. The retrieved numpy array of the image will be stored in a member attribute called **hiddenImage** and then it will be saved to the disk under the name "**hidden.jpg**". You can use **imsave** method from **skimage.io** library to save the image.
2. To retrieve the image, you have to reverse engineer the algorithm that was being used to hide the image. The algorithm works like this:
    a. The hidden image has the following shape(131, 100, 3)

b. The hidden image was integrated to the mountain image by replacing pixels from the mountain image by the pixels from the hidden image
c. The first pixel (row 0 col 0) from the mountain image was replaced by the first pixel (row 0 col 0) from the hidden image.
d. The 12th pixel (row 0 col 11) from the mountain image was replaced by the second pixel (row 0 col 1) from the hidden image and so on until all the pixels on the first row from the hidden image were concealed.
e. Same thing for all the rows by spacing them by 11 pixels each time. Meaning that both rows and columns are spaced by 11 pixels in the mountain image.
f. Your starting point is row 0 col 0 in the mountain image.

You need to reverse engineer this algorithm, meaning that you need to do the opposite operations in order to retrieve the hidden image from the mountain image. The algorithm is simple, you just need to know that every 2 consecutive pixels from the hidden image are separated by 11 pixels in the mountain image.

You can make sure that the image is retrieved by displaying it to the screen. The code to run your software will now look like this:

```
i = ImageAnalysis("mountain.png")
print(i)
#i.show()
i.retrieveHidden()
```

Open your file explorer and look for the created **hidden.jpg**. Open it to reveal the mystery. If you don't see the picture of someone, then you have to try harder and to review your implementation.

Step 3 (15 pts)

After retrieving the hidden image, you need now to fix the original mountain image. You have obviously noticed that the pixels from the hidden image are creating noise in the mountain picture. One way of fixing this, is to replace each of these pixels by the average color code from the neighboring pixels. This means that you need to replace the RGB tuple for these pixels by the average RGB from the 4 neighboring pixels.

Implement this algorithm in a method called **fix**.

The code to test your software will now look like this:

```
i = ImageAnalysis("mountain.png")
print(i)
i.retrieveHidden()
i.fix()
i.show()
```

Step 4 (25 pts)

Now we need to average the color in each pixel of the retrieved image and store the result to a file on the disk. Write a method called **averageRGB** that will do the following:

1. It will read the numpy array of the retrieved hidden image
2. For each pixel, it will average the values of R, G and B.
3. Write the average value to a new matrix. For example if the RGB tuple was (100, 120, 140), the average of the three is 120, so the new RGB value will be one element instead of 3: 120
4. Write the content of the newly created matrix to a file called RGB.csv
5. Please note that the number of lines in the file should match the number of rows in the corresponding matrix. You can check the number of rows by examining the shape method. The values should be separated by comma ",".

The code to test your software will now look like this:

```
i = ImageAnalysis("mountain.png")
#print(i)
i.retrieveHidden()
i.fix()
#i.show()
i.averageRGB()
```

Open your file explorer and look for the created file **RGB.csv**. Check that the content of the file is OK.

Step 5 (25 pts)

The last step is now to write a method called **load_rgb_from_file** that will read the values from the RGB.csv file (provided as argument) and construct a numpy array out of it. The shape of the numpy array should be (nbre_of_lines, nbre_of_columns, 3). Meaning that axis-0 will match the number of lines in the file, axis-1 will match the number of columns (values separated by comma) and each element in the array is a tuple of 3 elements corresponding to R, G and B. However, the 3 values will be the same and equal to the average value from the file.

After successfully building the array, use the following to test your code:

```
i = ImageAnalysis("mountain.png")
#print(i)
i.retrieveHidden()
i.fix()
#i.show()
i.averageRGB()
i.load_rgb_from_file(“RGB.csv”)
```

The method **load_rgb_from_file** should display to the screen the new image. What do you notice about it?