ENGR 103 Assignment 3: File Input/Output and Image Manipulation

Goals:

In this studio, we will be manipulating images with the help of the package “PIL”. Also, as we are focused on input and output this week, we will continue to practice the skill of inputting files to our programs and also saving the results to a new file.

In the first few steps, you will get familiar with PIL and how to manipulate images. In the last step, you will use what you’ve learned to create some artwork of your own! By overlaying several manipulated versions of the same image, we can create visually interesting results.

Step 1: Import and Display an Image Using PIL

Install PIL:

We will use the Python package called PIL to help with today’s work. A package is a bunch of code that somebody else has written, usually as several functions, and you get to use it conveniently without having to open up the underlying code (or understand it!). Use PyCharm’s “Python Packages” tab to search for “PIL” and install it. If “PIL” gives you an error, then install “pillow” instead.

Import PIL:

We have now *installed* PIL (so that the information lives on your computer), but we still need to *import* it to any script that needs to use it (so that the information is at hand and ready to be used). Near the top of your Python script, next to the other import statements, write: `import PIL.` Any time you use a package in a script, you must write a command like this to import the package.

Display an Image:

Make sure that an image file is in your working directory (the folder that contains your PyCharm project). It can be one of the images provided on Canvas, or any image of your choice, but try to use only .jpg formats. The provided code should have the following lines already written:

```
FileName = 'Image1.jpg'
My_Image = PIL.Image.open(FileName)
plt.imshow(My_Image)
plt.show()
```

Change the variable `FileName` to match the name of your picture file. When you run the code, the next few lines will import your image and show it on a plot.

Deliverable: A screenshot of your displayed image, in its plot window.

<u>Step 2: Manipulate the image</u>

Write the code to manipulate your image:

`PIL` offers functions that change the properties of images, all of which are fully detailed in the API for `PIL.Image` and `PIL.ImageOps`
Here: https://pillow.readthedocs.io/en/stable/reference/Image.html
And here: https://pillow.readthedocs.io/en/stable/reference/ImageOps.html .

For the most part, only the functions `PIL.Image.rotate()`, `PIL.Image.paste()`, `PIL.ImageOps.flip()`, and `PIL.ImageOps.mirror()` will be useful for us today. If you ever become confused about how to use these functions, the API has the answers.

However, as a more user-friendly source of help, the provided function called `Example_Transforms` demonstrates the usage of these `PIL` functions. For today's studio, if you're unsure of how to use these functions, you can simply copy and paste them from `Example_Transforms` and fill in your own image names, rotation angles, etc.

Again: rather than truly understanding the syntax of these functions, it is sufficient to simply copy and paste the provided examples, and fill in your own variables names and numbers where necessary. This is an effective way of learning to code.

If you are still curious about the syntax, let's go through a few examples.

You can use `PIL.Image.rotate()` like this: use PIL to open an image and assign it a variable name, like `My_Image` shown above. Then you can type `Rotated_Image = My_Image.rotate(Angle)`, where `Angle` is the angle to rotate the image counterclockwise in degrees and the variable `Rotated_Image` stores the result.

You can use `PIL.Image.paste()` like this: Open two images and assign them variable names like `My_Image1` and `My_Image2`. Then you can type `My_Image1.paste(My_Image2)` to paste `My_Image2` onto `My_Image1`.

You can use `PIL.ImageOps.flip()`, like this: open an image and assign it a variable name (like `My_Image` shown above). Then you can type `Flipped = PIL.ImageOps.flip(My_Image)`. The result is stored as the variable `Rotated_Image`.

The syntax of each differs subtly enough that it is too hard to *memorize* the syntax. Instead, it is easier to consult previous examples or the official API.

Deliverable:

A screenshot of your image with single manipulation, in its plot window

Step 3: Save the file as an output

Write the code to save your image:

After all of your manipulations, save your output to a file. If the name of the manipulated image is My_Transformed_Image, then write the following command at the end of the main function: `My_Transformed_Image.save('Saved_Output.jpg')`. This should save the image to an output file that you can find in your file explorer.

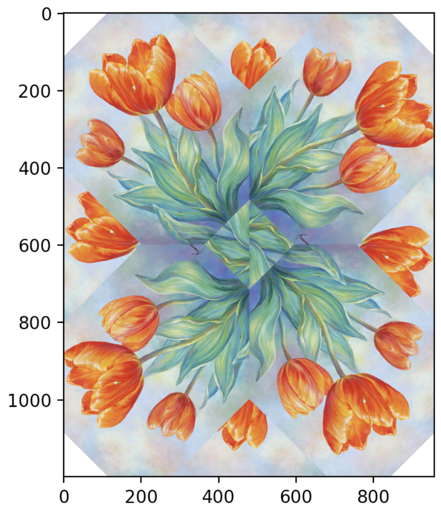
Deliverable:

A screenshot of your file explorer, showing your saved output file in the same folder as your Python code.

Step 4: Make Art by Repeated Manipulations

Write the code to combine several manipulated images:

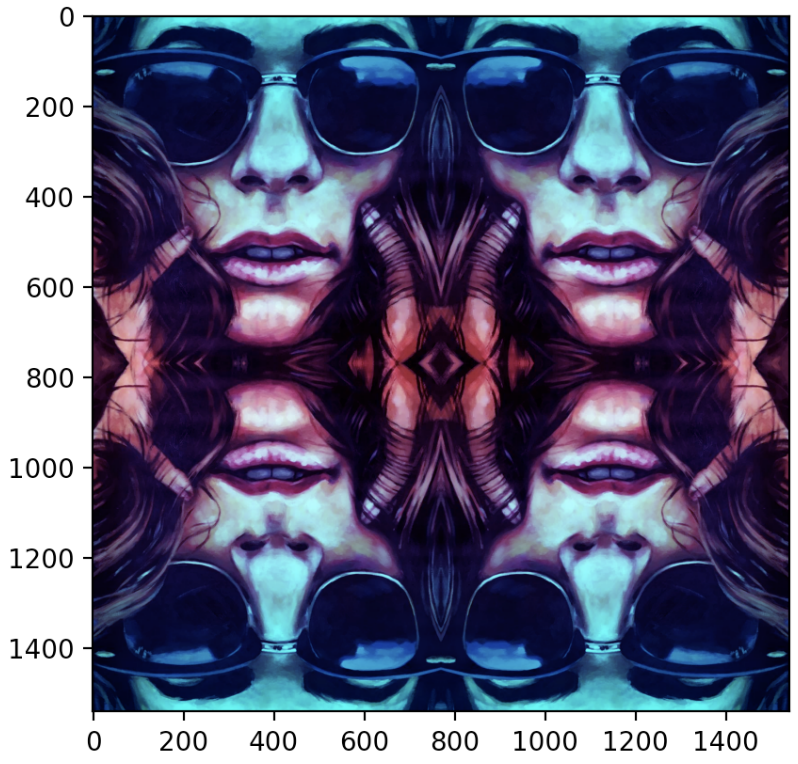
Using what you've learned about manipulations, paste several manipulated images together on one canvas! Try to make the result artistic, beautiful, or visually interesting, like the combination shown below.



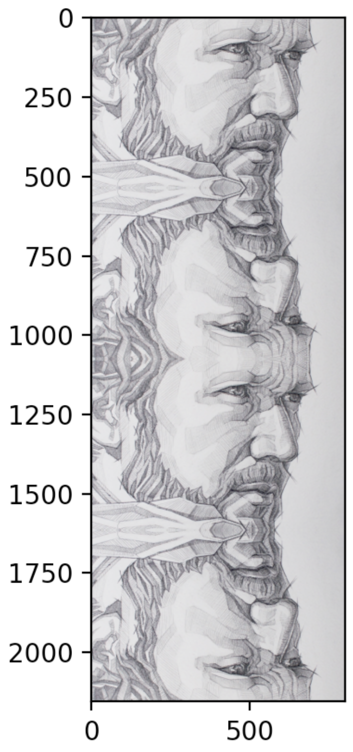
As a general strategy, perform manipulations one at a time and paste each onto a single canvas until your artwork is complete. For optimum brownie points, try to recreate the image above (starting with the provided "image1.jpg)

Deliverable: A screenshot of your artwork.

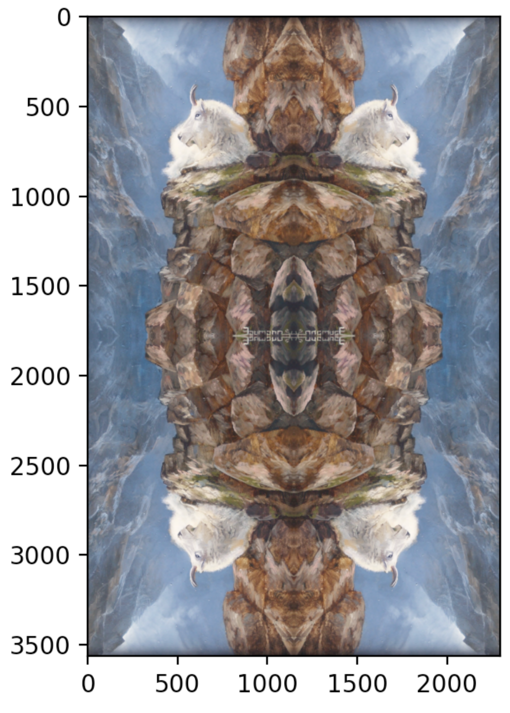
More inspiration:



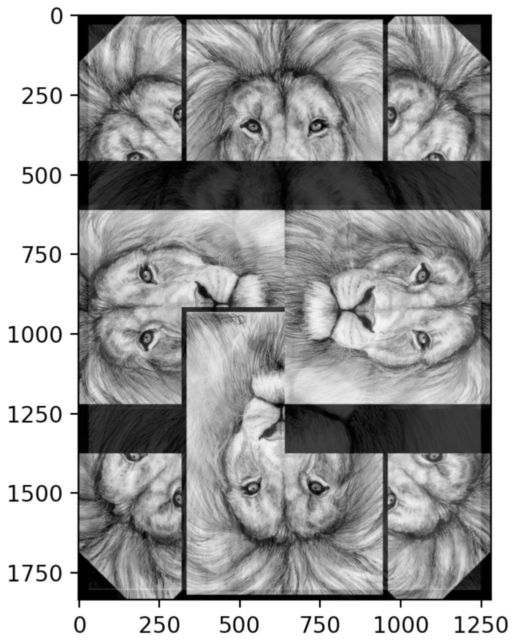
0
200
400
600
800
1000
1200
1400
0
200
400
600
800
1000
1200
1400



0
250
500
750
1000
1250
1500
1750
2000
0
500



0
500
1000
1500
2000
2500
3000
3500
0
500
1000
1500
2000



0
250
500
750
1000
1250
1500
1750
0
250
500
750
1000
1250