

**Department of Artificial Intelligence**

**College of Computer Science and Information Technology**

1. **Objectives**

This lab is designed to achieve the following goals:

1. Get Started with Scikit-image.
2. Get familiar with Reading and displaying images.
3. Resizing Images
4. Rotate by different angles.
5. Horizontal and Vertical Flip
6. Image Cropping
7. Altering Image Brightness
8. **Introduction**

## Scikit-image



Scikit-image (a.k.a skimage) is an open-source python package that work with NumPy arrays. It implements algorithms and utilities for use in research and industry applications.

To Install Scikit-image library

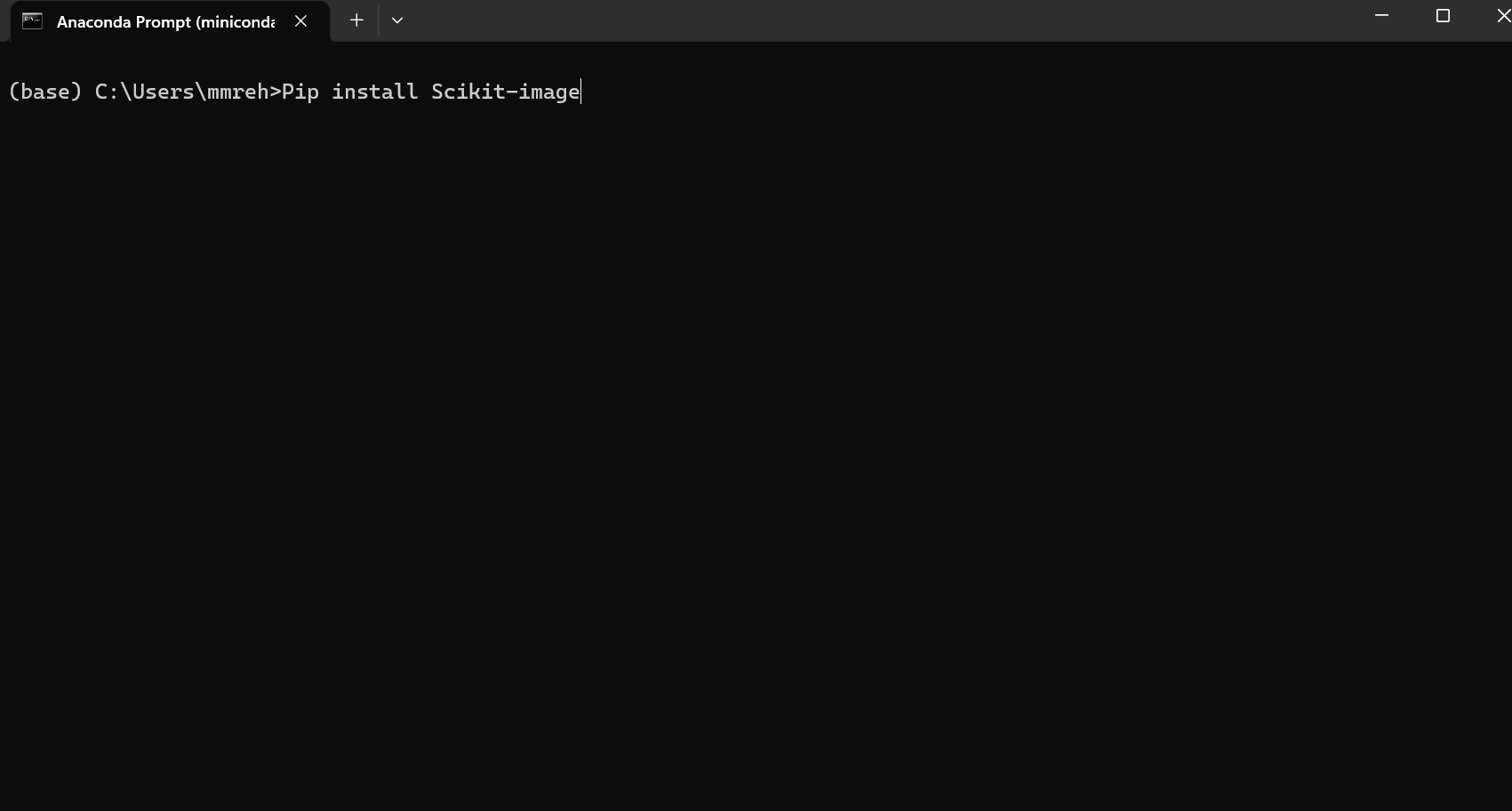
1. Open anaconda prompt
2. Write the following Command

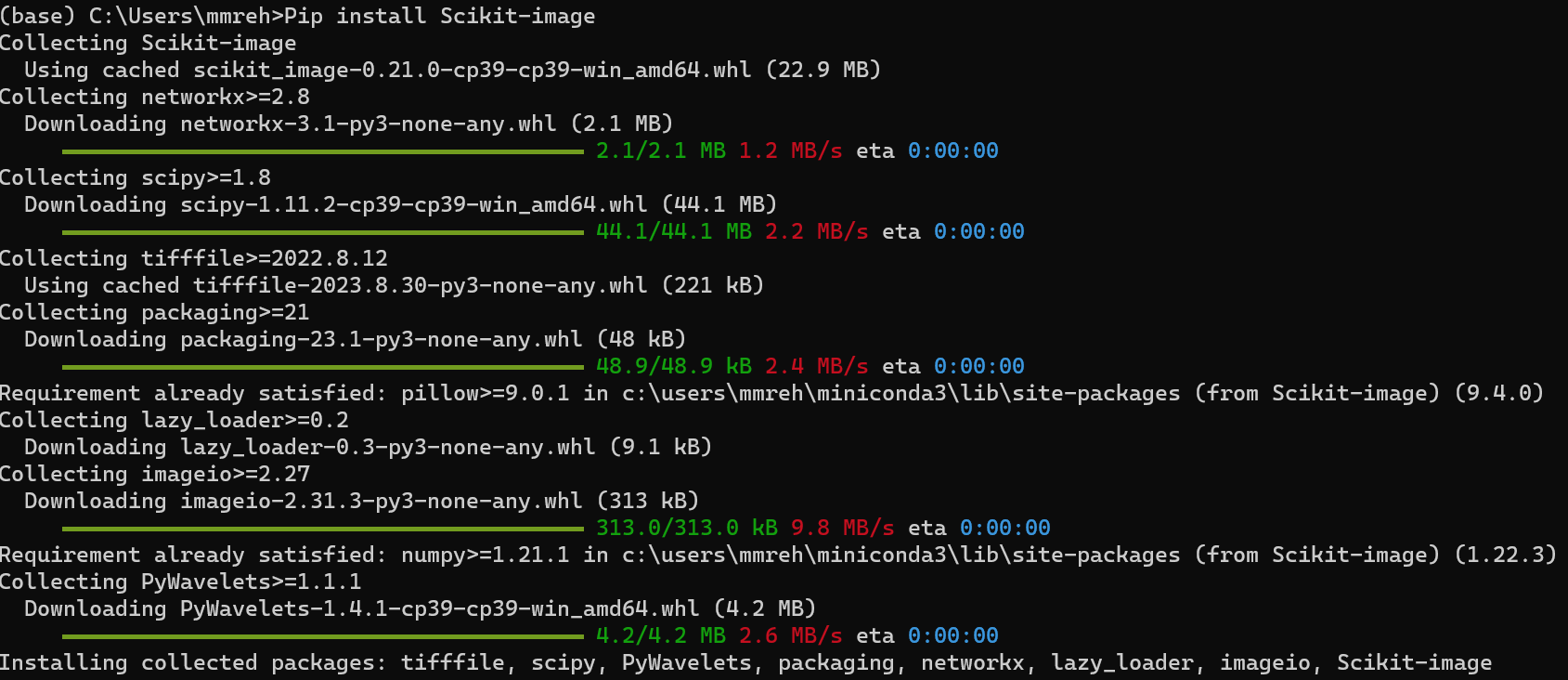
Pip install Scikit-image

# Getting started

The package is imported as skimage:







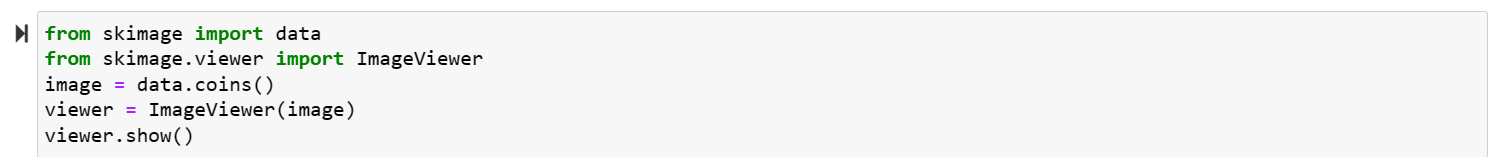
1. **Load and Show an Image**

Within the scikit-image package, there are several sample images provided in the data module. Instead of using an external image, we can simply load one of the images provided within the package!

## Displaying images using Image I/O

****

## Displaying images using Image Viewer



****

## Displaying images using Matplotlib

**A screenshot of a computer screen

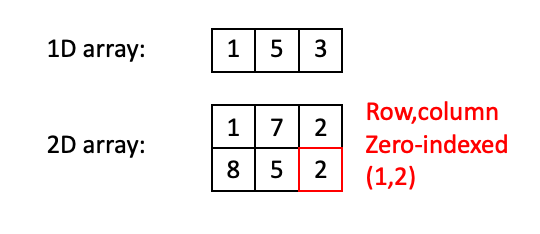
Description automatically generated**

1. **Reading Images from our System using skimage**

****

1. **Image Details**

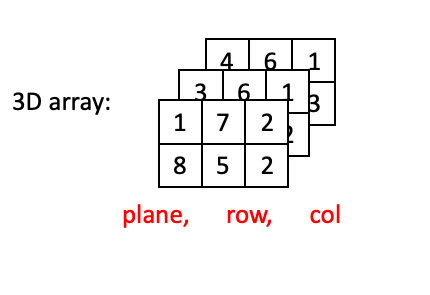
Digital images are represented as numeric arrays, so lets begin with a quick review. In Python (NumPy, specifically), arrays are data structures that consist of collections of items of the same type. Each item in the collection has a unique location with respect to an N-dimensional grid. These locations are zero-indexed, so the "first" item along an axis has index 0.

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Arrays_Slide1.png)

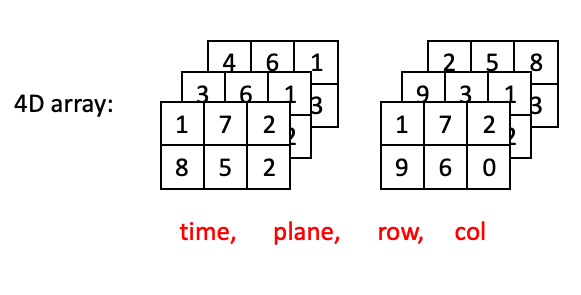
In the 1D case above, there are three items (integers, in this example) of index 0, 1, and 2. In the 2D case, the item in the red box has the index (1,2). Note that NumPy conventions dictate that indexing works like (row, column).

Once you go beyond two dimensions, ordering of dimensions becomes somewhat arbitrary, and different packages/software will use different conventions for how arrays are ordered. We will use the conventions of [scikit-image](https://scikit-image.org/) for indexing.

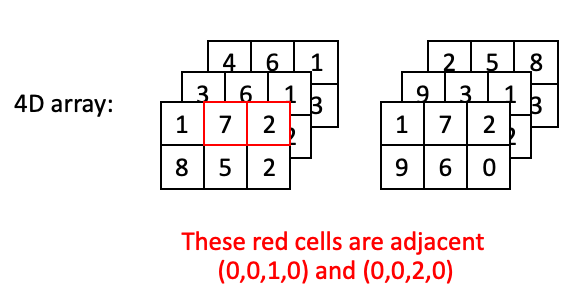
3D arrays use (plane, row, column) ordering:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Arrays_Slide2.png)

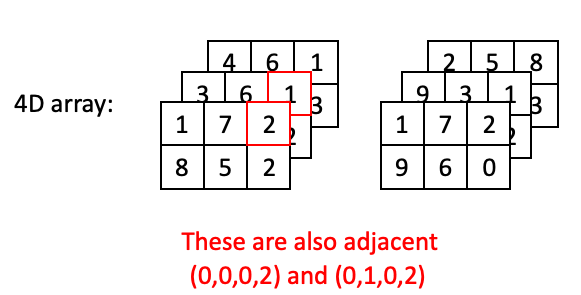
4D arrays are little harder to visualize, but you can think of 4D arrays as a time series of 3D arrays, with dimensions ordered as (time, plane, row, column):

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Arrays_Slide3.png)

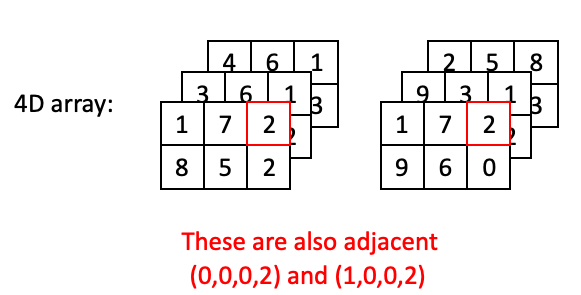
Now lets go through some examples of "adjacent" positions in the array and what they would look like with respect to their index. The most straightforward example would be two positions in the same plane at the same time:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Arrays_Slide4.png)

However, two positions may also be adjacent with a step along the depth axis:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Arrays_Slide5.png)

Or a step along the time axis:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Arrays_Slide6.png)

## Images as 2D (grayscale) or 3D (2D color) arrays

The simplest image to work with is a grayscale image. These are represented as 2D arrays, where each position in the array corresponds to a single pixel. Brighter (whiter) parts of the image are represented by larger values, and darker parts of the image are represented by smaller values. The values themselves may differ with respect to type.

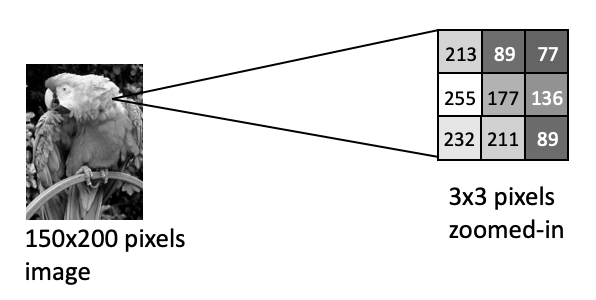
It might be useful at this point to review [numpy data types](https://docs.scipy.org/doc/numpy-1.16.1/reference/arrays.scalars.html#built-in-scalar-types).

The most common types you will encounter are:

* uint8 - integers between 0-255
* uint16 - integers between 0-65,535
* float64(or double) - floating point numbers, e.g., 1.45643, 5.0, 1.23435e9

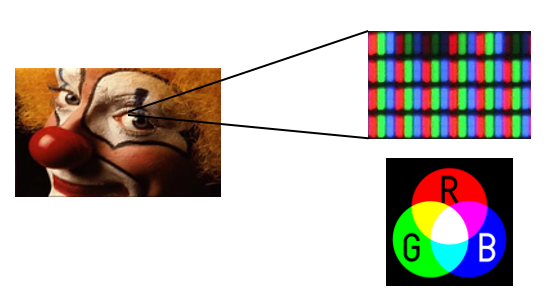
scikit-image functions will usually output float64 values, regardless of which data type you start with, but these can usually be safely converted back to the original type.

In the following parrot image, values are uint8 and thus range from 0-255:

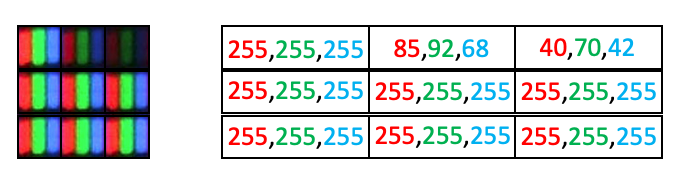
[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Arrays_Slide7.png)

Notice how the brightest pixel is 255, and the darker pixels have much lower values.

Color images are conceptually similar, except each "pixel" has three values, corresponding a a red, green and blue value. Consider the clown image below:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Arrays_Slide8.png)

Because human photoreceptors are tuned to red, green, and blue wavelengths, all of the colors we can see can be represented as a combination of red, green, and blue light. Thus if you were to look at your monitor under a microscope, you would notice the the clown image is actually composed of an array of tightly packed red, green and blue elements. Hence each pixel has a red, green, and blue value:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Arrays_Slide9.png)

However, arrays can only hold one value per position. So 2D color images actually need to be represented as 3D arrays, where each plane in the color dimension (channel) represents a different color:

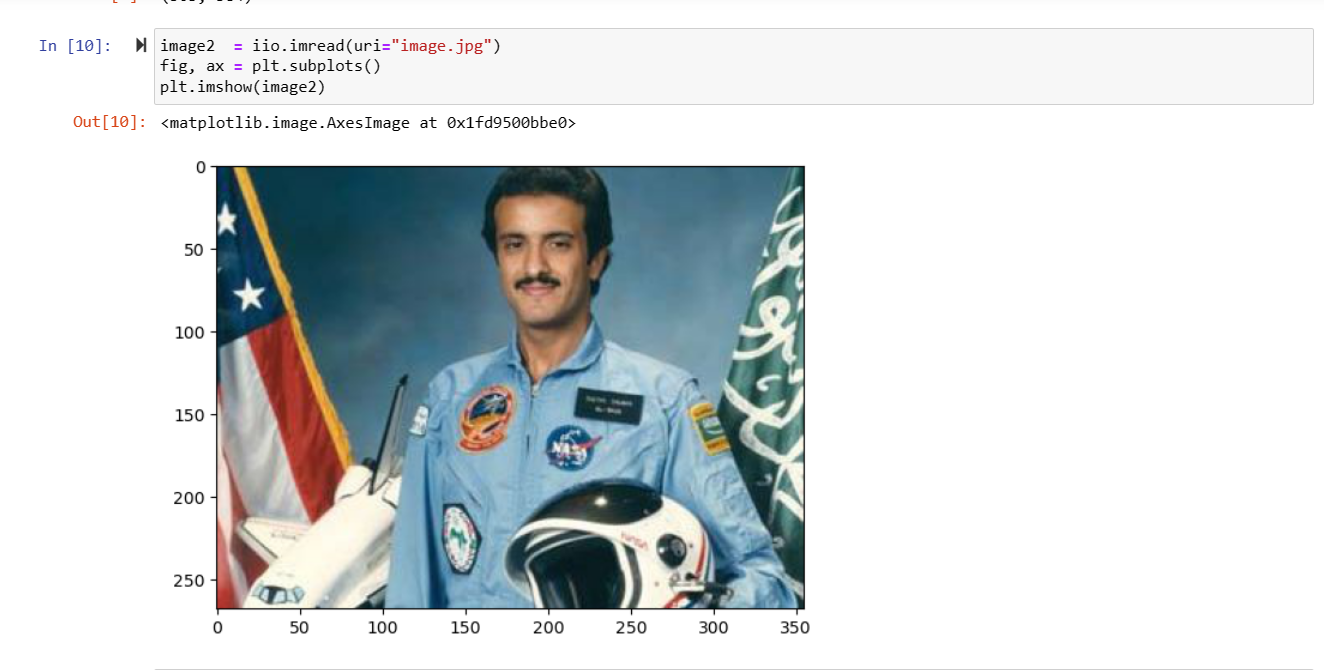
[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Arrays_Slide10.png)

In scikit-image, the indexing convention for color images is (row, column, channel), where the three colors are represented as three channels, with the ordering {0: 'red', 1: 'green', 2: 'blue'}.

The maximum number of dimensions we usually have to deal with in image analysis is five: time, plane (z axis), row (y axis), column (x axis), channel. In scikit-image, you will often see the ordering stated like this: (t, pln, row, col, ch).

## Working with 2-D color images

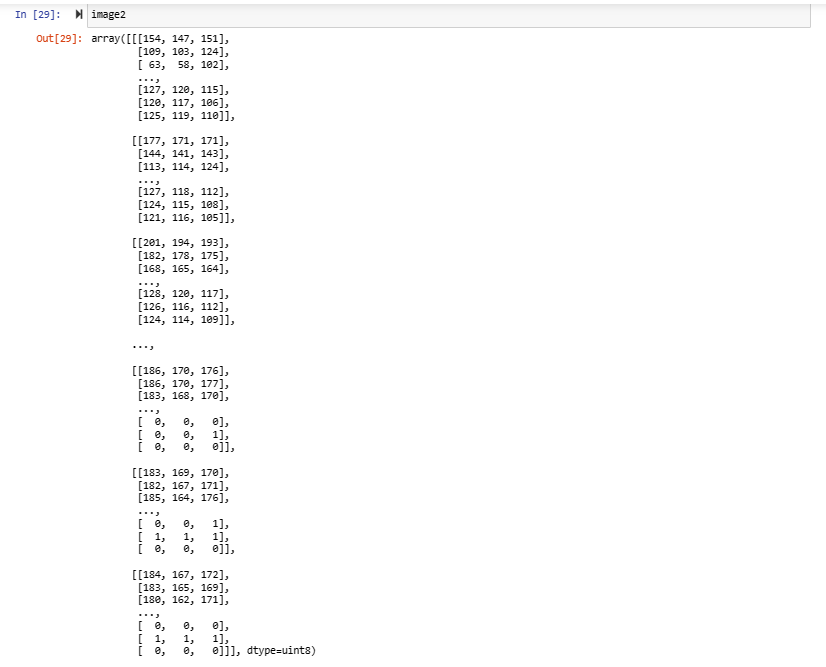
Now let's take a look at using scikit-image (abbreviated as skimage) to work with a two-dimensional color image. We need to first start with some imports. We will use numpy for working with arrays and skimage for loading and visualizing image data and performing some basic operations:



So, what is the type of image?

****

Again, our image is represented as a numpy array. If you just call the image, Python will return a snippet. This isn't terribly useful, but can give you a feel for the type of data you are working with:

****

What is more useful is to look at the values for various array attributes:

Number of dimensions:

**** Size along each dimension (axis):

****

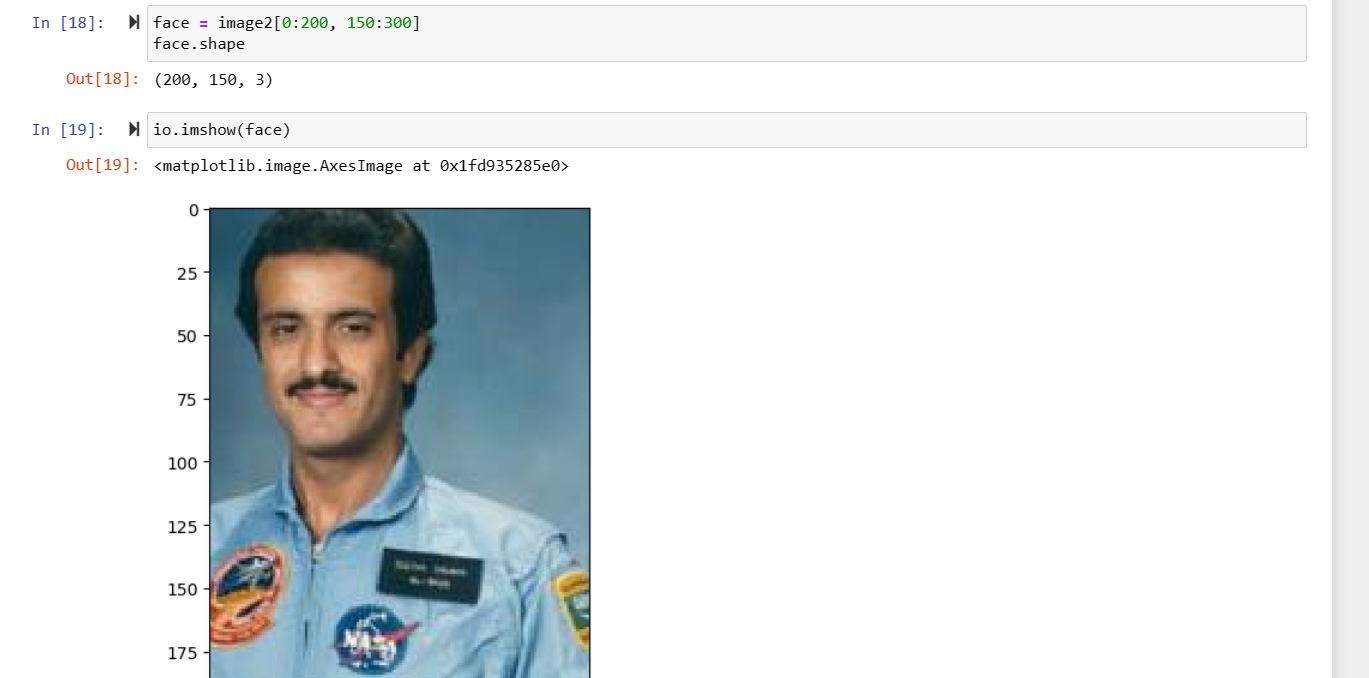
Data type of the image (i.e., data type of the pixel values):

****Retrieving statistical information about image intensity values:

**A white sheet music with black lines

Description automatically generated**

And we can use indexing to select just parts of the array (Slicing)



Specify only the first value of each individual pixel

A person in uniform with a badge

Description automatically generated

Replace some matrix values.

*Set the pixel at (0:200, 150:300) to "Red"*

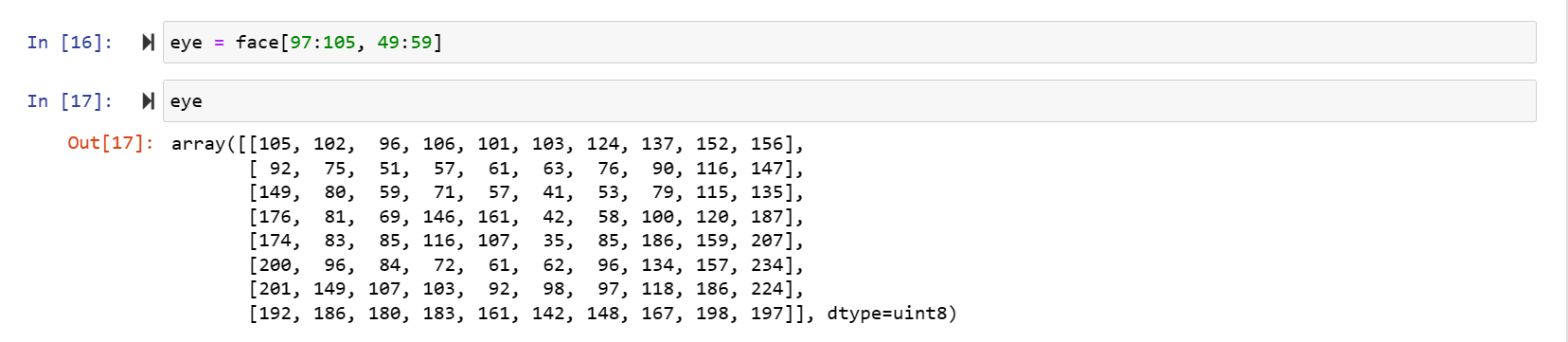
A red square on a screen

Description automatically generated

## NumPy indexing.

NumPy indexing can be used both for looking at the pixel values and to modify them:

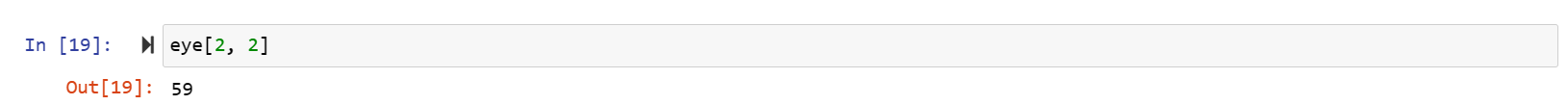
Very small arrays are easier to visualize as images:



A screenshot of a video

Description automatically generated

You can also return the values for specific pixels with indexing:

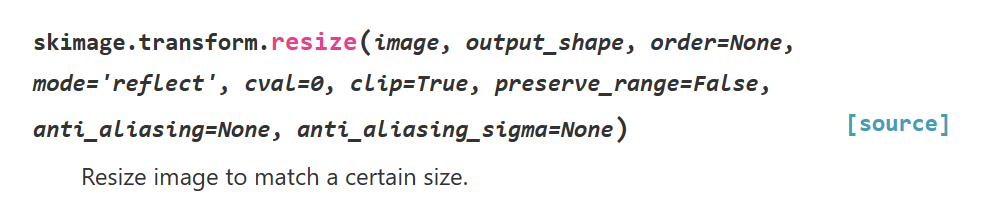


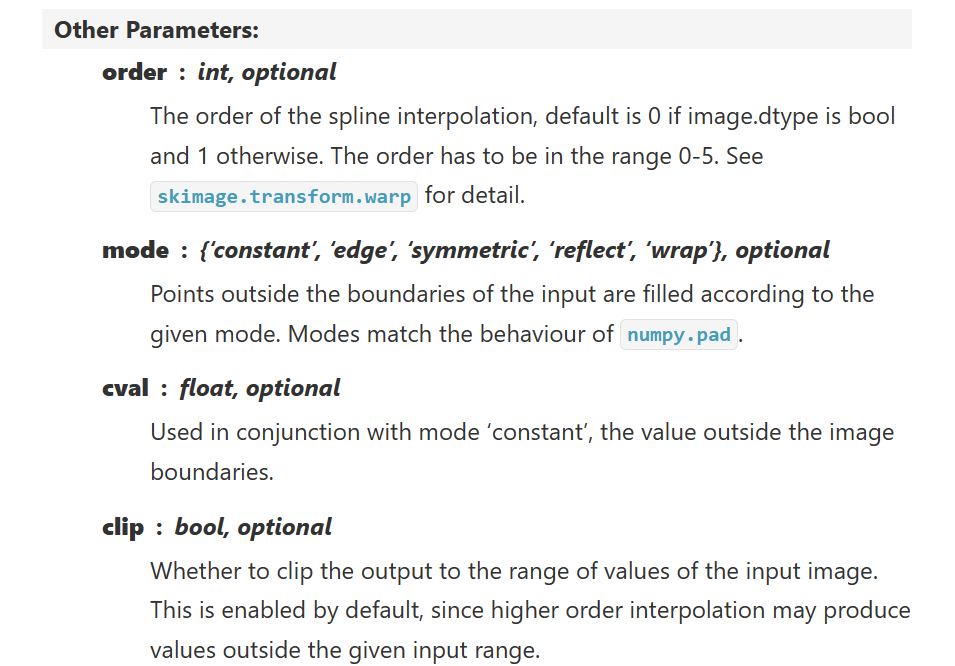
1. **Resize image**

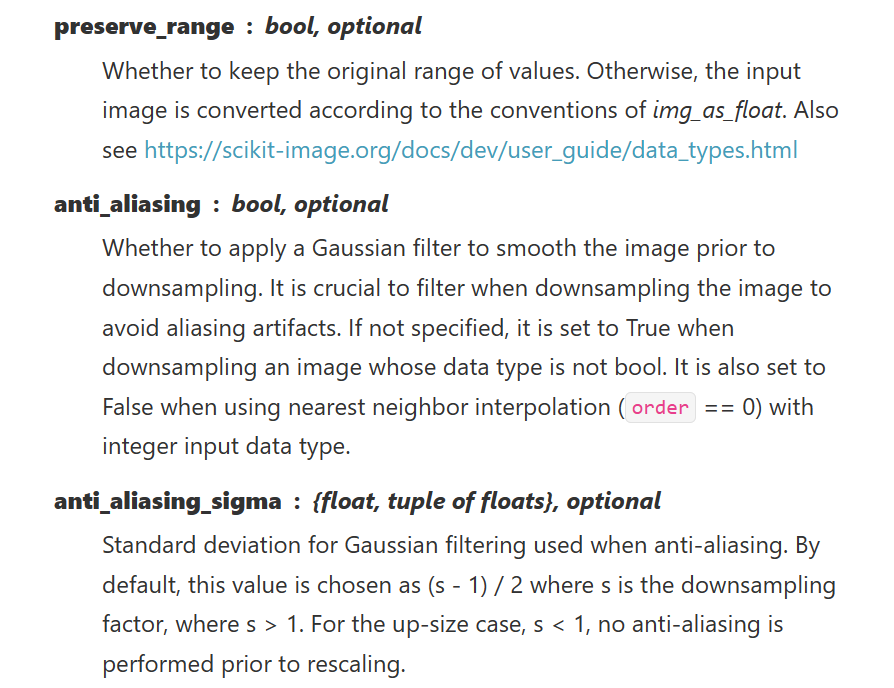
One of the biggest challenges in computer vision is that we require a huge amount of data for training our model. The data we collect is often from different sources which might result in variation in the size of the images. This might be a problem while extracting features from the images, or using the same for data augmentation.

Ideally, the size of the images should be the same when we’re building our model. If we’re using a pre-trained model, it is important to resize and normalize the input data to the same format over which the network was originally trained. This is why resizing images is an important image preprocessing step.

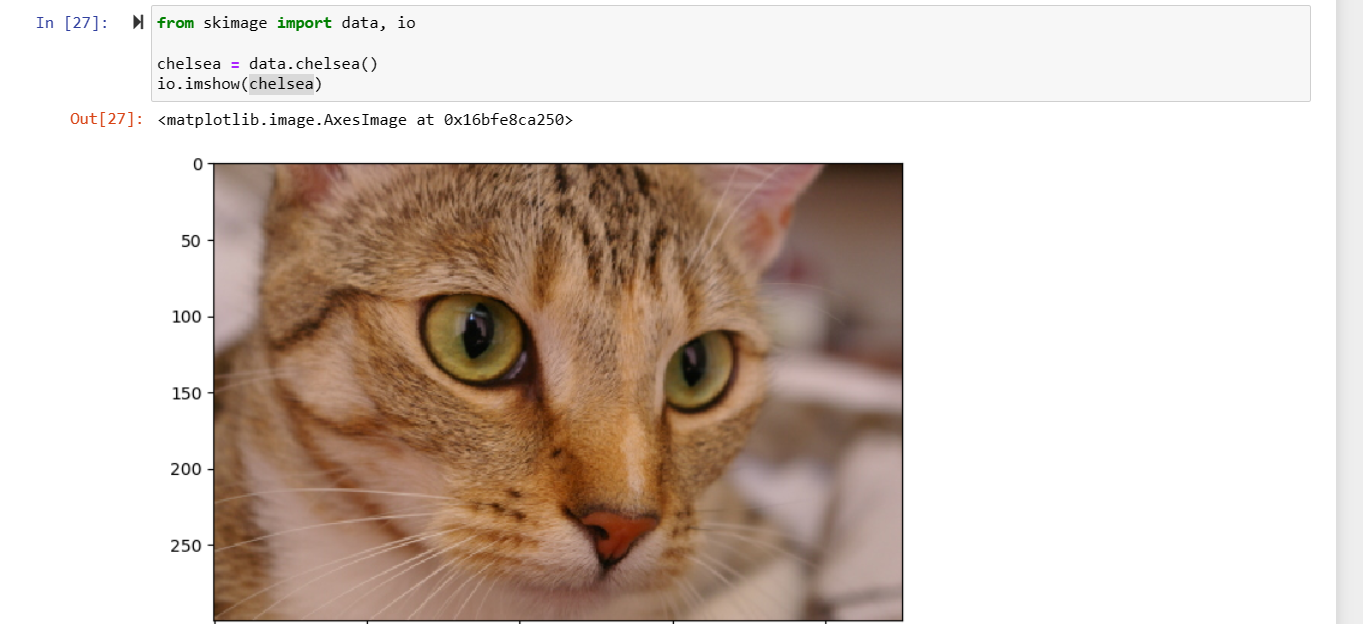
Here, we are going to use the resize function from skimage. The input to this function will be the image we want to update and the required dimensions for the new image:

 A screenshot of a computer

Description automatically generated

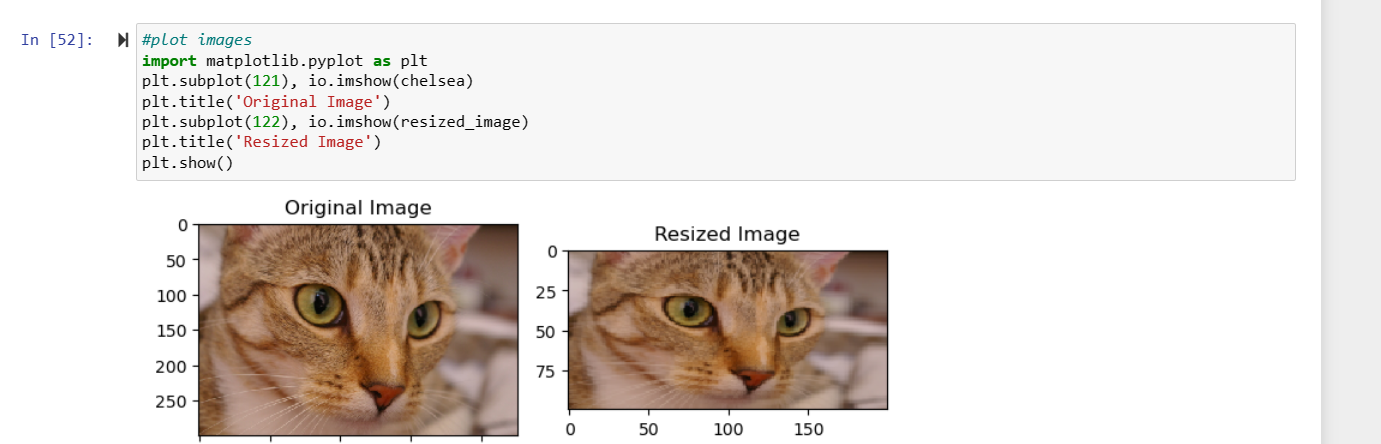


Example:



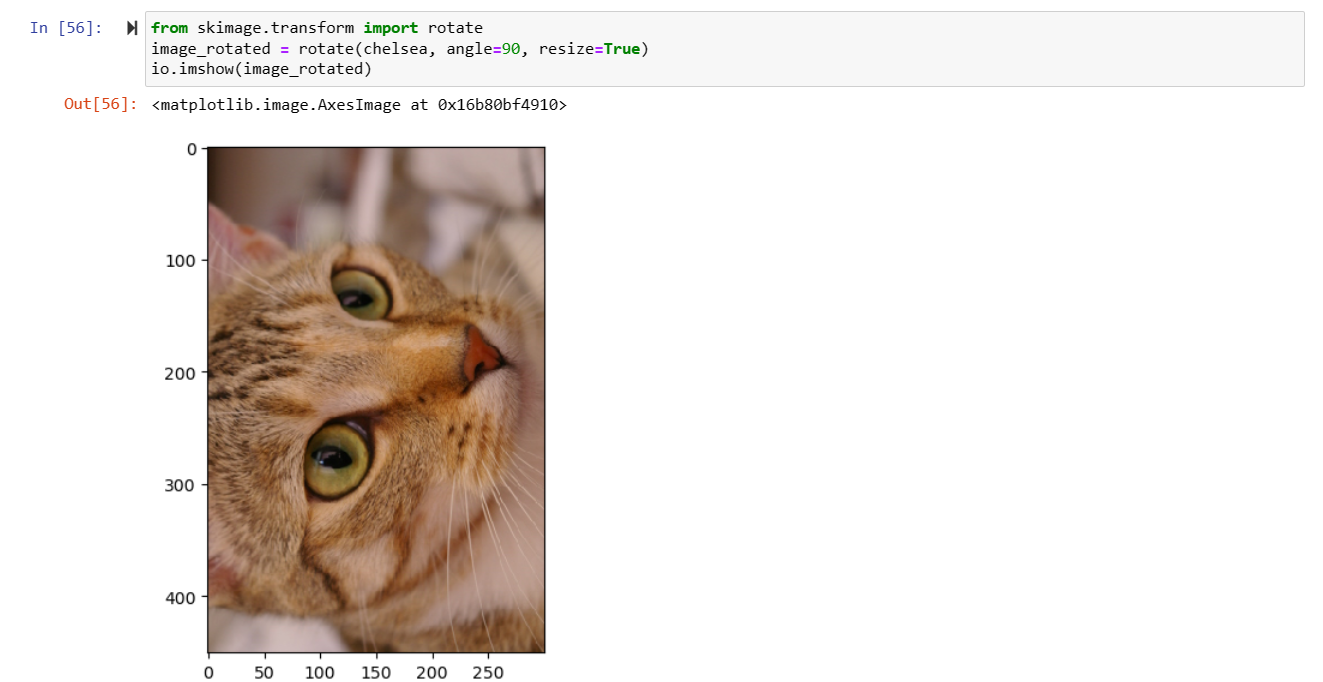
A close up of a cat

Description automatically generated



1. **Rotate image**

Let’s turn our focus and see how we can change the orientation of images.

****

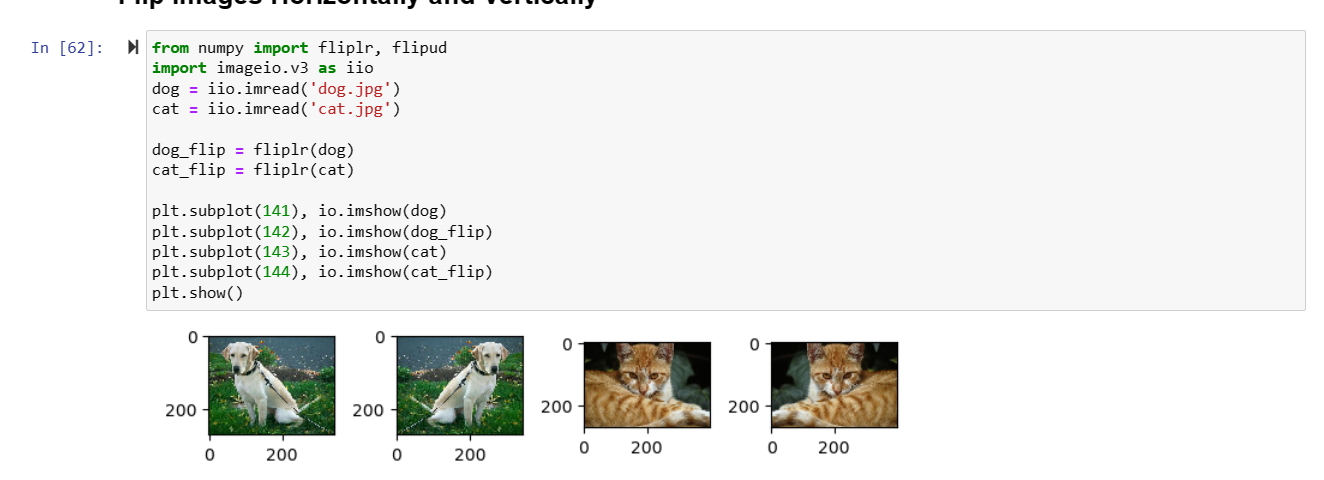
1. **Flip Images Horizontally and Vertically**

We can flip an image both horizontally and vertically. This creates a mirror image along the horizontal/vertical axis. We can use this technique for both image preprocessing and image augmentation.

Although there is no direct function for this in skimage, we can use NumPy to perform this task.

NumPy provides functions ***flipud***and ***fliplr***for flipping the images across the horizontal and vertical axis respectively.

The internal working of the function is very simple. For a horizontal flip, the rows remain intact while the entries in the columns are reserved. Let us take the same cat/dog example and use the flip function on it:



1. **Crop Images**

You must have used the cropping function on your phone a gazillion times.

You can crop images inside your Python notebook as well using skimage. We crop images to remove the unwanted portion of the image or to focus on a particular part of the image.



1. **Sava Image**

****

**References:**

[Getting started — skimage 0.21.0 documentation (scikit-image.org)](https://scikit-image.org/docs/stable/user_guide/getting_started.html)

[Rescale, resize, and downscale — skimage 0.21.0 documentation (scikit-image.org)](https://scikit-image.org/docs/stable/auto_examples/transform/plot_rescale.html#sphx-glr-auto-examples-transform-plot-rescale-py)

[Skimage | Skimage Tutorial | Skimage Python (analyticsvidhya.com)](https://www.analyticsvidhya.com/blog/2019/09/9-powerful-tricks-for-working-image-data-skimage-python/)

[Basic\_skimageJAX/lessons/1-Images\_are\_arrays.md at master · TheJacksonLaboratory/Basic\_skimageJAX (github.com)](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/1-Images_are_arrays.md)