

**Department of Artificial Intelligence**

**College of Computer Science and Information Technology**

1. **Objectives**

This lab is designed to achieve the following goals:

1. Get an introduction to image processing.
2. Get familiar with Installing python and jupyter notebook.
3. Explain how a digital image is composed of pixels.
4. Explain how images are stored in NumPy arrays.
5. Explain the left-hand coordinate system used in digital images.
6. Explain the RGB additive colour model used in digital images.

**Due Date: Sunday September 1, 2024 @ 11:59 PM \***

**Late Submissions:**

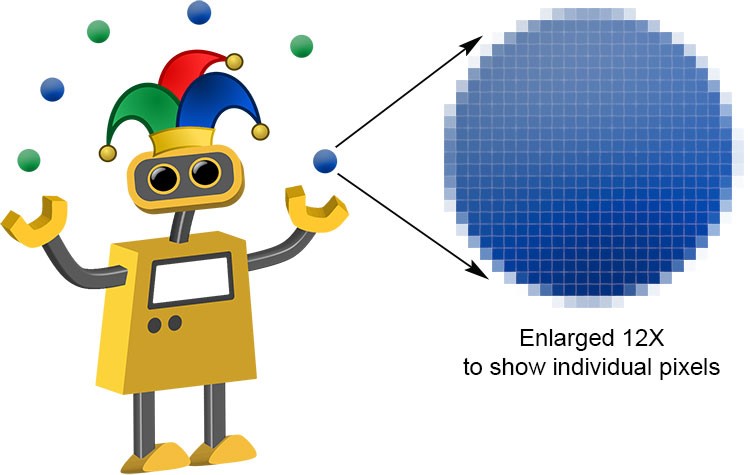
* Q: Can I skip the lab and submit the solution?
  + You will receive a mark of **zero** if you do not attend the lab, even if you complete the exercise. Attending the labs is compulsory for evaluation. If you have a justified excuse, you may receive a partial mark depending on the circumstances. See the next question for information on late submissions.
* **Q:** If I submit it at 12:00am, you’ll still mark it, right?
  + **A:** 11:59pm and earlier is on time. Anything after 11:59pm is late. Anything late will **NOT** be probably marked. If I find you have a legitimate cause, you will be graded according to the following rules (24 hours after deadline 🡪 assignment is marked out of 75% only, 48 hours after deadline 🡪 assignment is marked out of 50% only, 74 hours after deadline 🡪 assignment is marked out of 25% only).

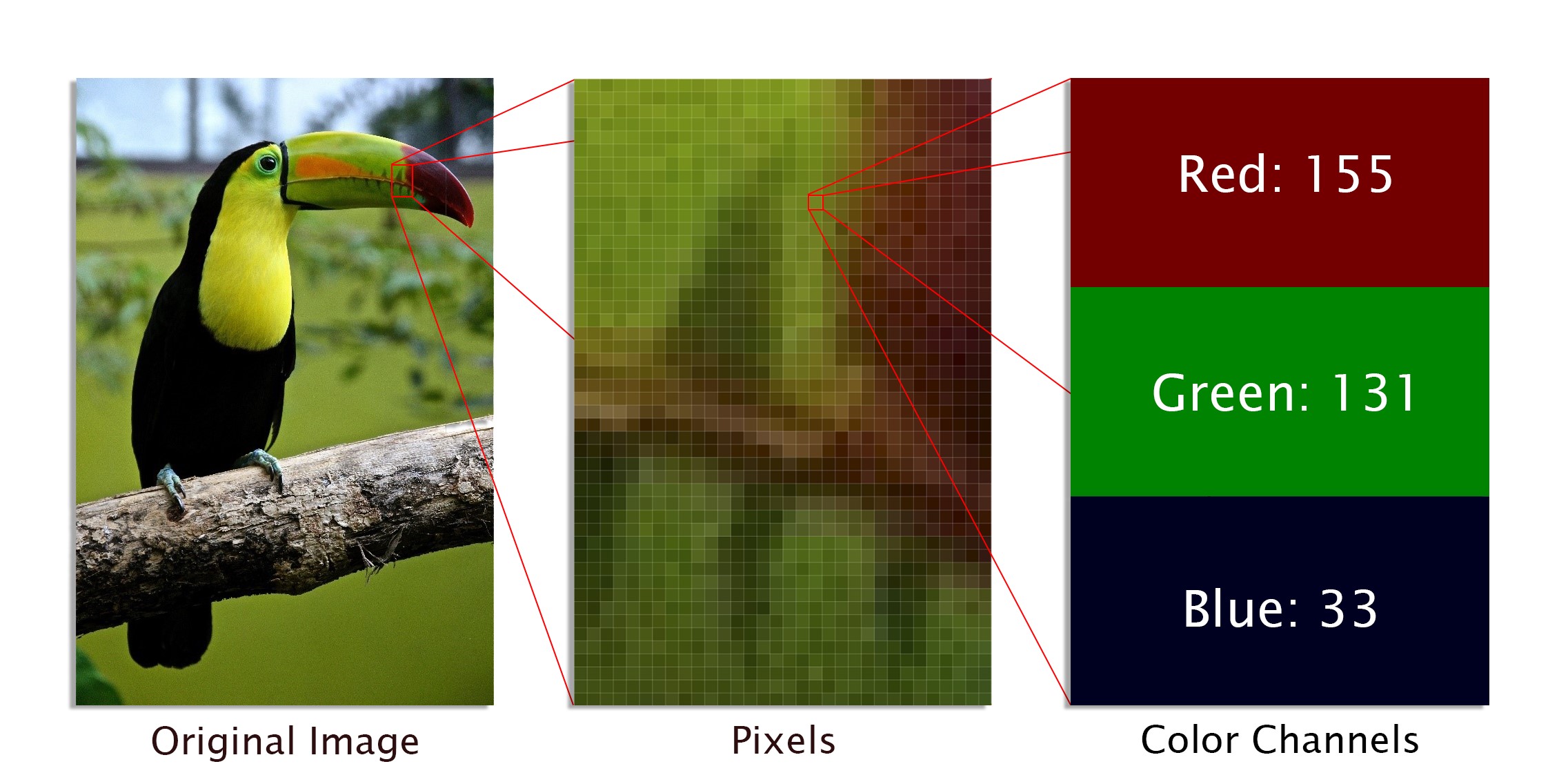
1. **Introduction**

## 2.1 What is a Digital Image

A digital image is a representation of a real image as a set of numbers that can be stored and handled by a digital computer. In order to translate the image into numbers, it is divided into small areas called **pixels** (picture elements). For each pixel, the imaging device records a number, or a small set of numbers, that describe some property of this pixel, such as its brightness (the intensity of the light) or its color. The numbers are arranged in an array of rows and columns that correspond to the vertical and horizontal positions of the pixels in the image. Digital images have several basic characteristics. One is the type of the image. For example, a black and white image records only the intensity of the light falling on the pixels. A color image can have three colors, normally RGB (Red, Green, Blue) or four colors, CMYK (Cyan, Magenta, Yellow, black). RGB images are usually used in computer monitors and scanners, while CMYK images are used in color printers. There are also non-optical images such as ultrasound or X-ray in which the intensity of sound or X-rays is recorded. In range images, the distance of the pixel from the observer is recorded. Resolution is expressed in the number of pixels per inch (ppi). A higher resolution gives a more detailed image. A computer monitor typically has a resolution of 100 ppi, while a printer has a resolution ranging from 300 ppi to more than 1440 ppi. This is why an image looks much better in print than on a monitor. For the colored image it works a little bit dierent every pixel contains 3 values

R(red) G(green) B(blue) and these values are the ones who controles our image’s color.





## 2.2 Image Processing

**Image processing** is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them. It is among rapidly growing technologies today, with its applications in various aspects of a business. Image Processing forms core research area within engineering and computer science disciplines too.

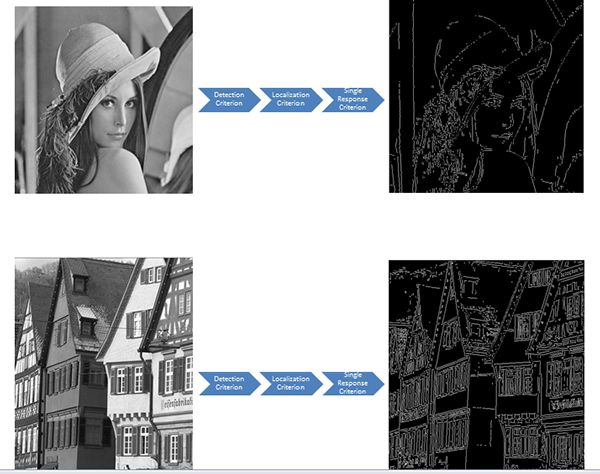
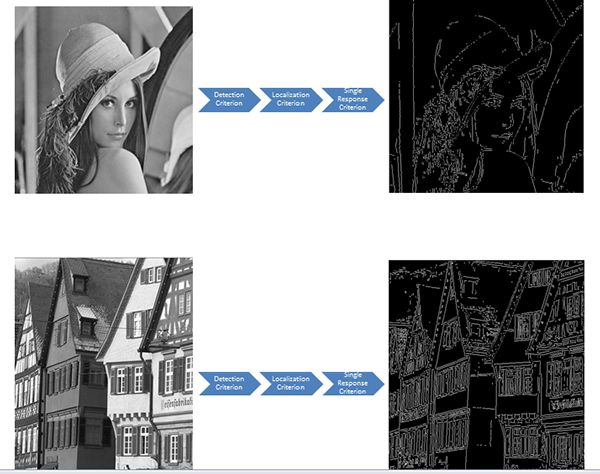
Image processing basically includes the following three steps:

* Importing the image via image acquisition tools.
* Analyzing and manipulating the image.
* Output in which result can be altered image or report that is based on image analysis.

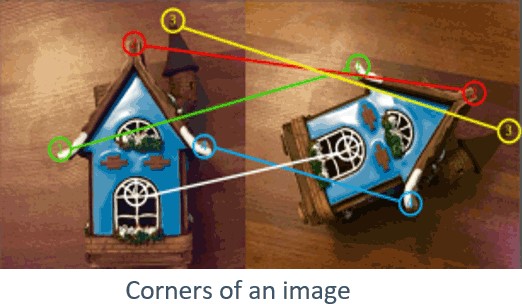
Low-level image processing algorithms include:

* Edge detection
* Segmentation.
* Classification.
* Feature detection and matching.

**Edge detection**



**Classification and Feature detection and matching**



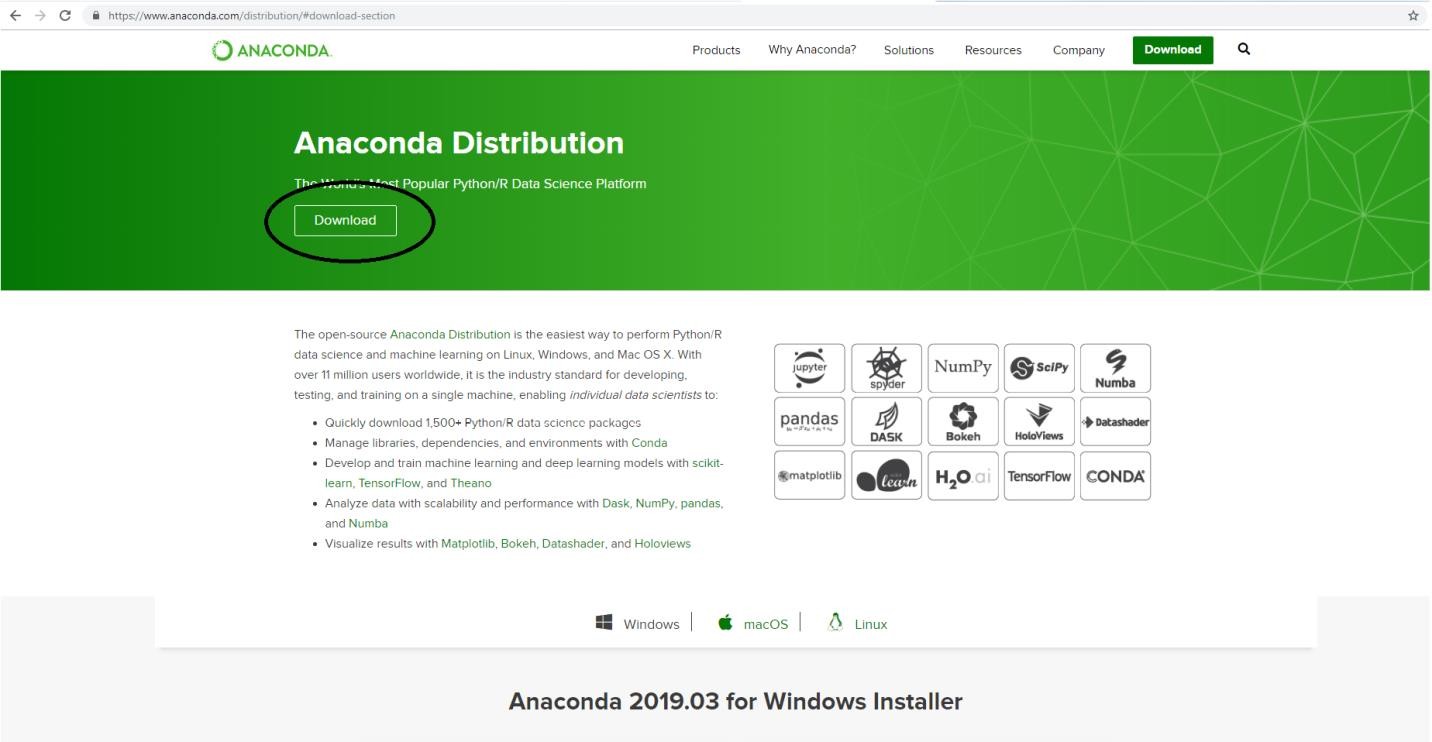
1. **Installing python and jupyter notebook**

In this part, we will show how to install Jupyter Notebook on your system. **The Jupyter Notebook is free of cost.**

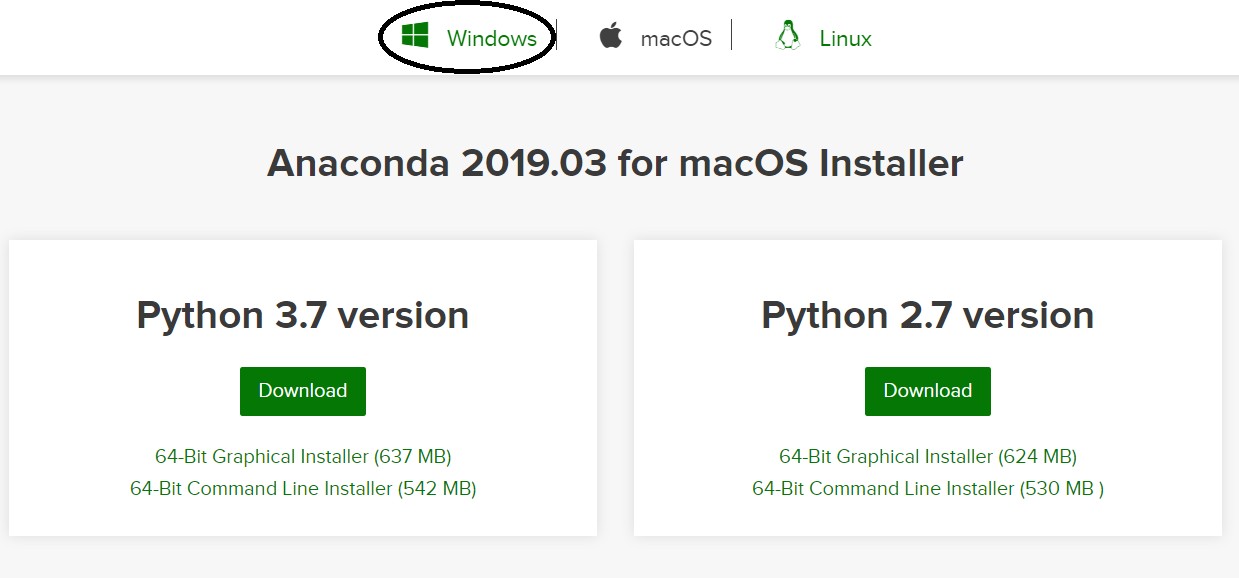
**Step 1: Install Anaconda**

If you already have Anaconda Installed, you can skip to **Step no 8** directly. Else follow below steps

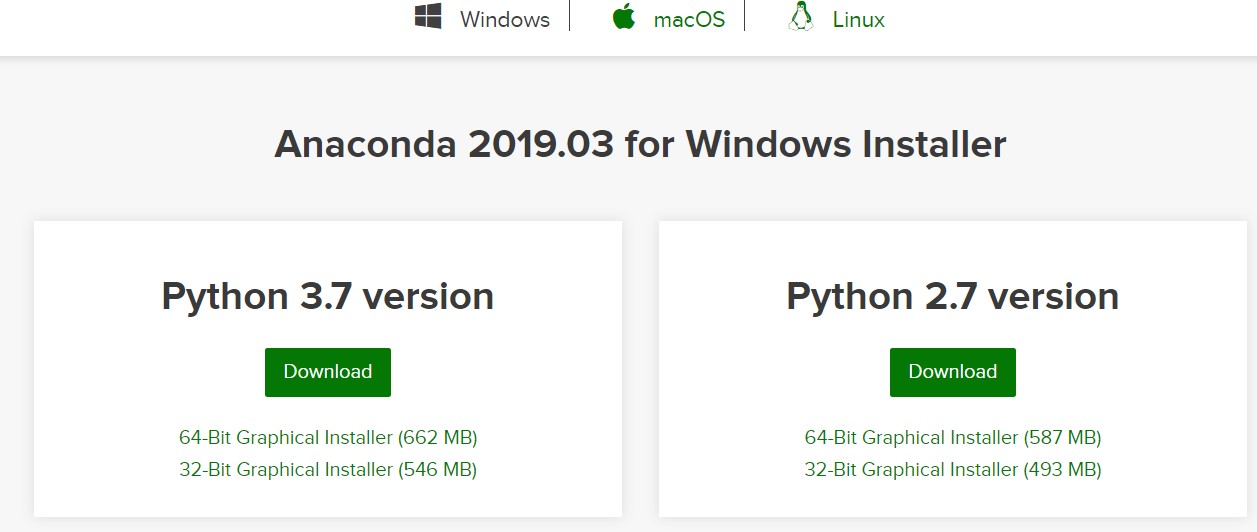
Open your browser and go to Anaconda website (<https://www.anaconda.com/distribution/>) to download and install Anaconda. You will see a page like this. Click on download.



**Step 2:** You will see that following page appears. By default, Anaconda shows you the download link for Mac operating system. If you have Mac, then you can click “64-Bit Graphical Installer” under Python 3.7 version to start downloading the file. In this computer, Windows is the operating system, so we will select Windows as shown below. If you have Linux as operating system, then you can select Linux option and download file in similar manner as Mac. Mac and Linux users can skip **Step 3 & 4**.



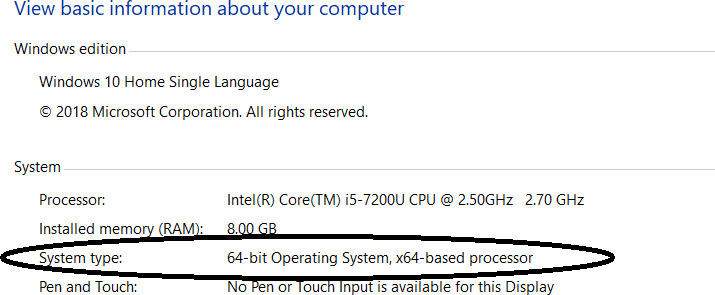
After clicking on Windows button, the following page will appear.



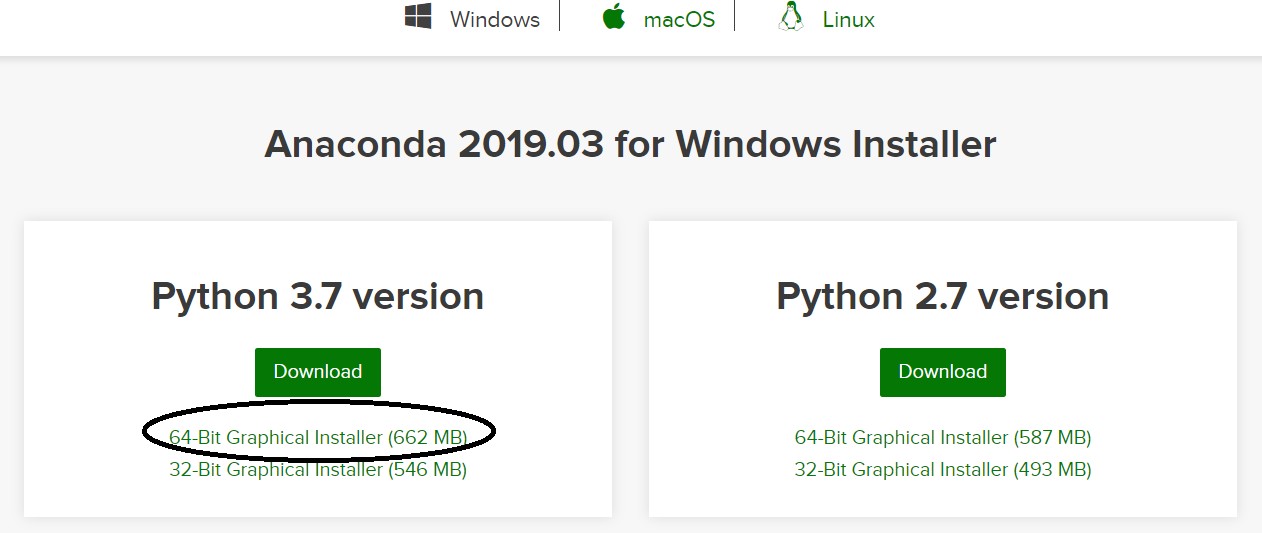
**Step 3:** You can see that there are two options for Windows: 64-Bit and 32-Bit. You need to find out whether your system is 64-Bit or 32-Bit and accordingly you need to select the file for your system. To do so, go to your desktop home screen, right click on ‘Computer’ icon, then select Properties.



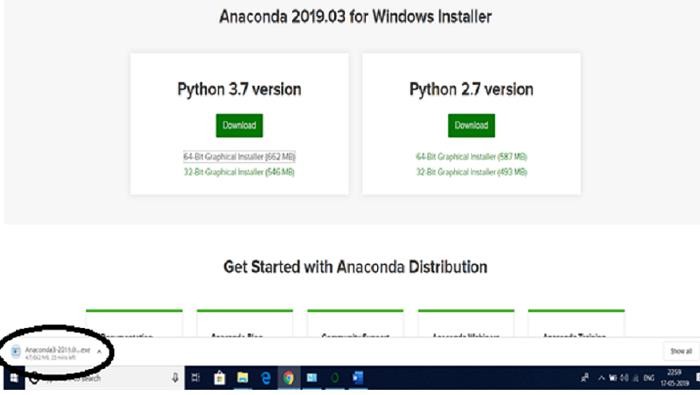
This will show you basic information about your system. Look for “System Type” as shown below and check whether it is 64-bit or 32-bit. For this computer, we see that Windows system type is 64 -bit.



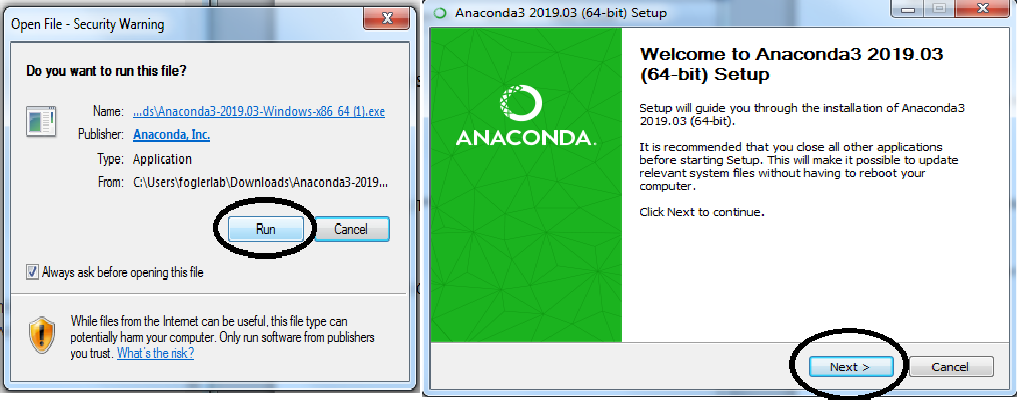
**Step 4:** Now, go back to your browser and then click “[64-Bit Graphical Installer (662 MB)](https://repo.anaconda.com/archive/Anaconda3-2019.03-Windows-x86_64.exe)” as this computer is 64 bit (as identified in Step 3)



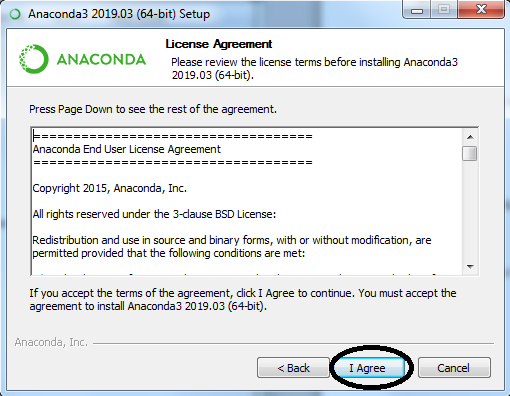
The installer will start downloading the file (this may take a while) and will appear in bottom left of your browser (if you are using google chrome) as shown below.



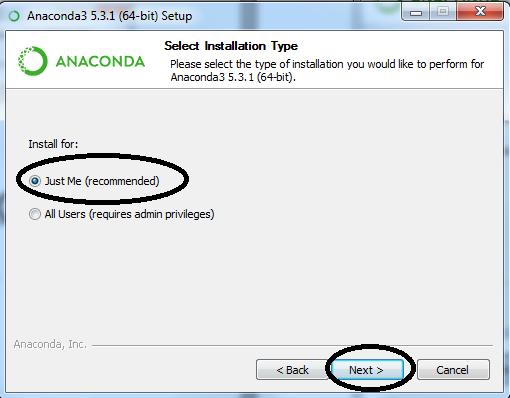
**Step 5:** When the file is completely downloaded, click on the file. You will see that following window appears. Click on ‘Run’, and then click ‘Next’ button.



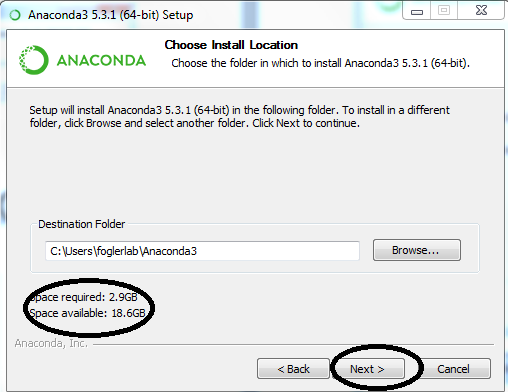
A new window will appear asking you to accept the terms of agreement, select “I Agree”.



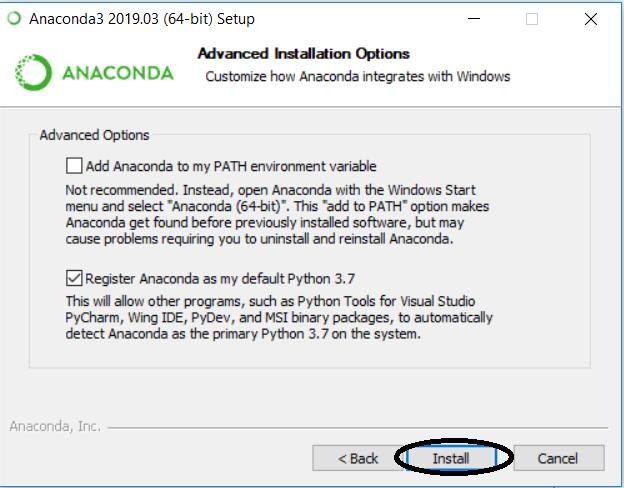
Select ‘Just Me’ which is recommended and then click Next.



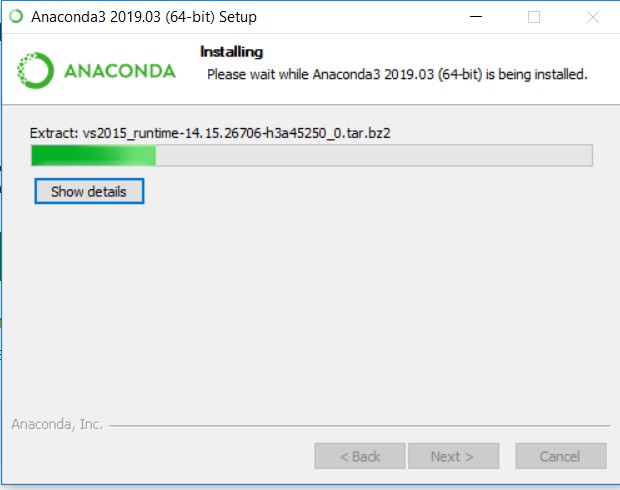
**Step 6:** Make sure you have the required free space for software installation. which you can check as shown below. Then click Next. (If you don’t have required space, then you need to delete some of your items to free the space)



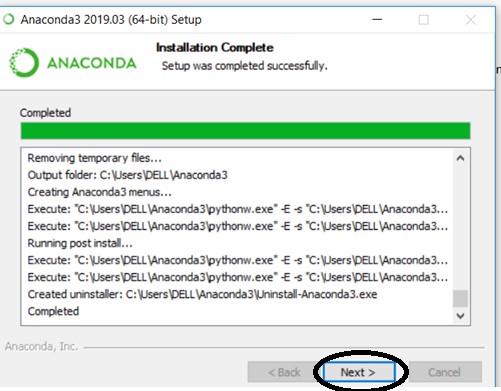
**Step 7:** You will see that following window appears. Click on Install.



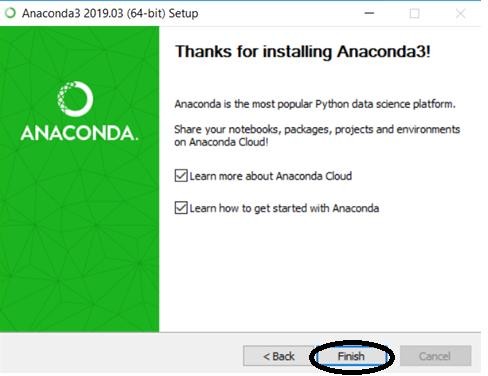
This will lead you to installation page showing the progress of installation. It will take some time for the software to get installed.



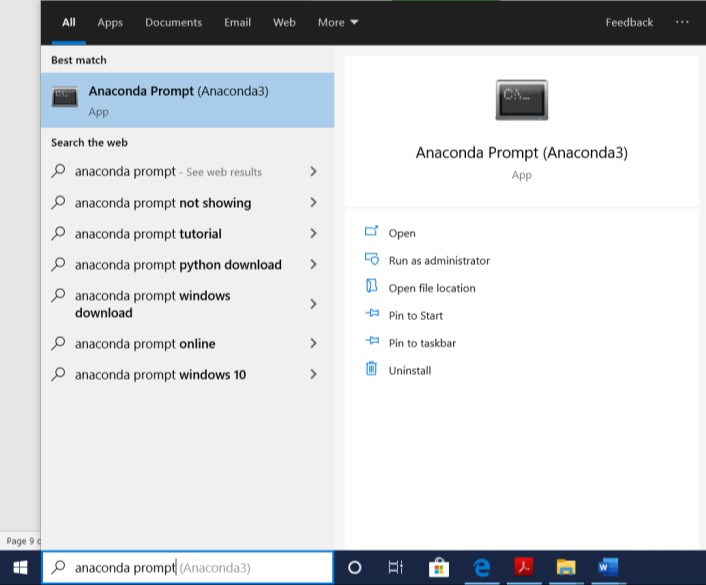
After all the files are extracted, the “Next” button will get enabled. Click on Next button



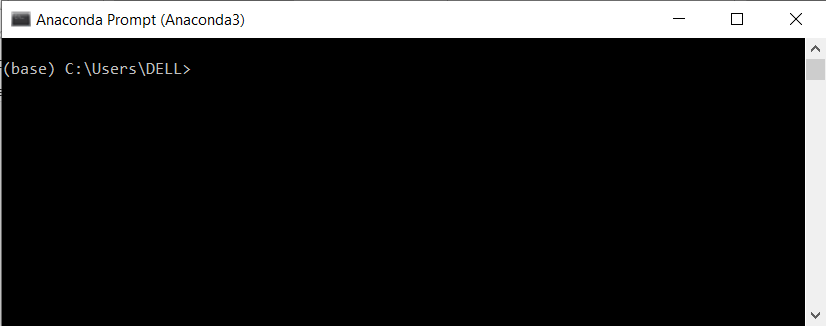
Then following window will appear. Click on Finish button to complete the installation. Now Anaconda has been installed on your computer.



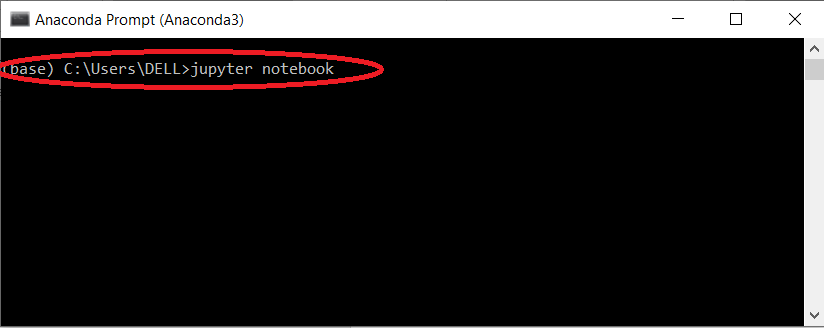
**Step 8:** Type ‘anaconda prompt’ in search box and click on the icon indicated below.



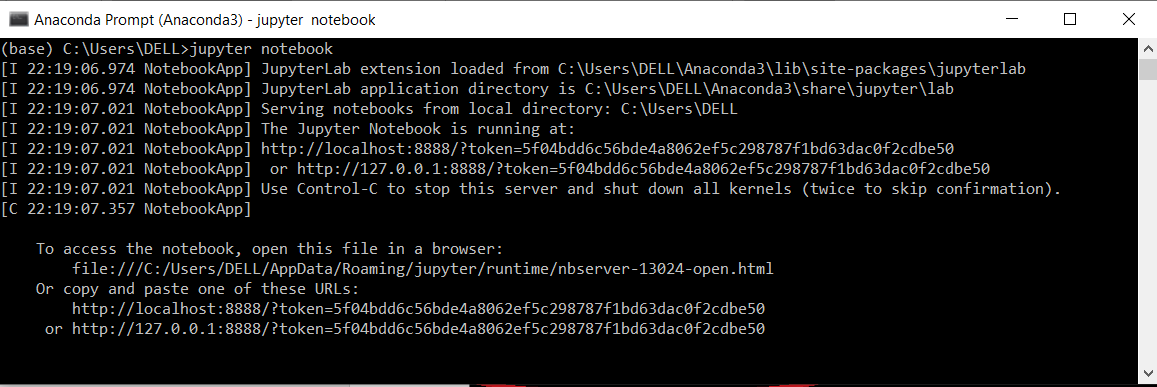
You will see that a command window opens. Just wait for few seconds until you see a file location (something like shown below)



**Step 9:** Type “jupyter notebook” in the command prompt and then Press Enter



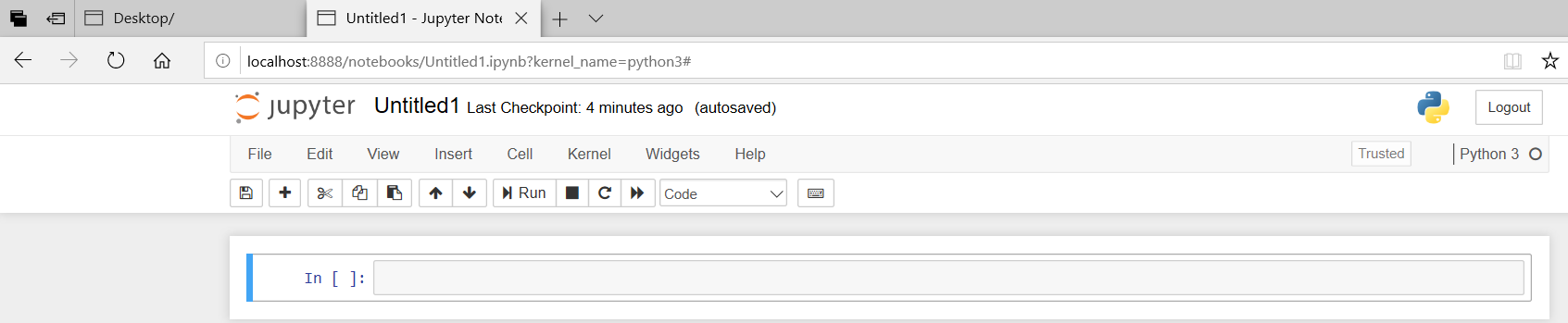
In few seconds, you will see that your command is executed as shown below.



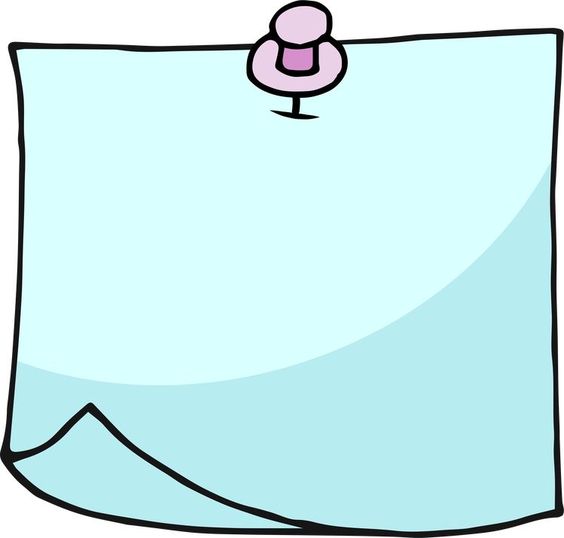
At the same time, you will also see that your browser opens showing Jupyter Notebook Interface. Click on “New” located at upper right corner if you wish to open Jupyter Notebook.



This will open Jupyter Notebook in an another tab as shown below

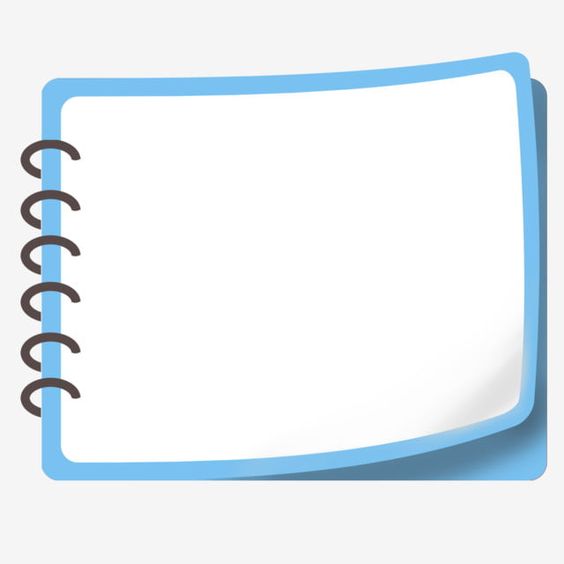


**Step 10:** Now you can start creating your own code or open an LEP

**Working with Pixels**

For our purposes, the distinction between matrices and arrays is not important, we don’t really care how the computer arranges our data in its memory. The important thing is that the computer stores values describing the pixels in images, as arrays.

As noted, in practice, real world images will typically be made up of a vast number of pixels, and each of these pixels will be one of potentially millions of colors. While we will deal with pictures of such complexity shortly, let’s start our exploration with 15 pixels in a 5 X 3 matrix with 2 colors and work our way up to that complex.



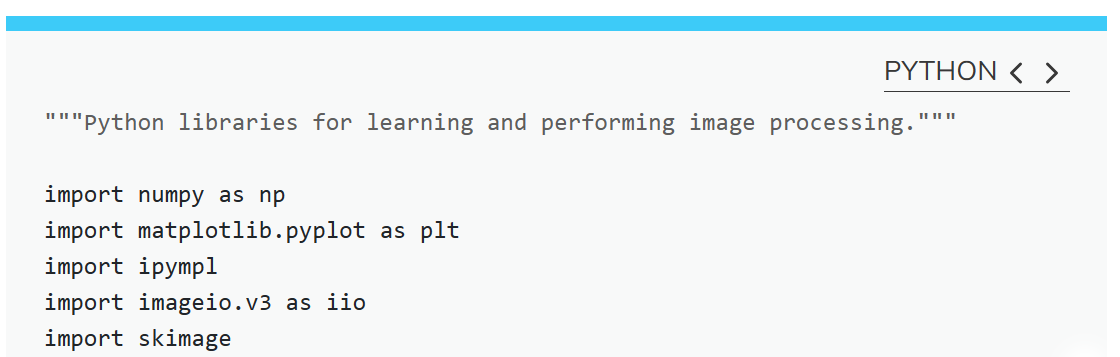
### **MATRICES, ARRAYS, IMAGES AND PIXELS**

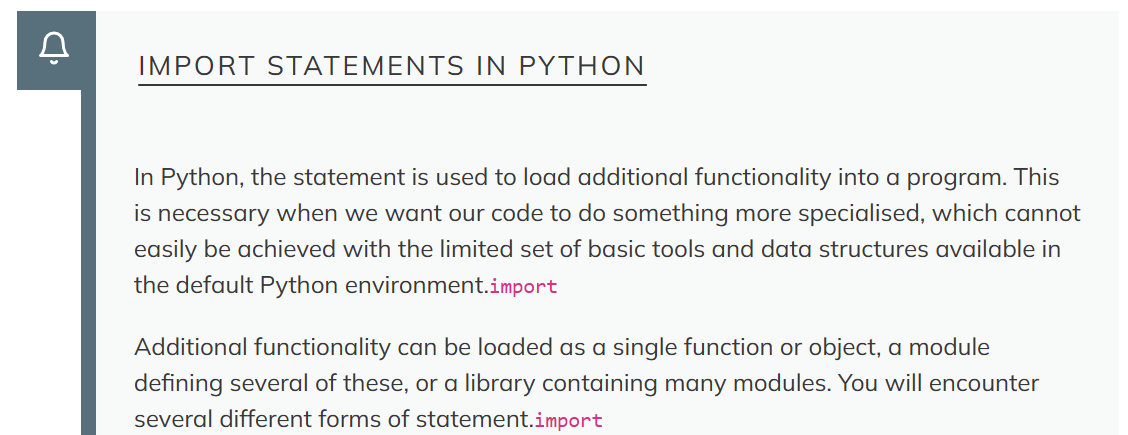
The **matrix** is mathematical concept - numbers evenly arranged in a rectangle. This can be a two-dimensional rectangle.

**Array** refers to a structure in the computer’s memory where data is stored in evenly spaced **elements**. This is strongly analogous to a matrix.

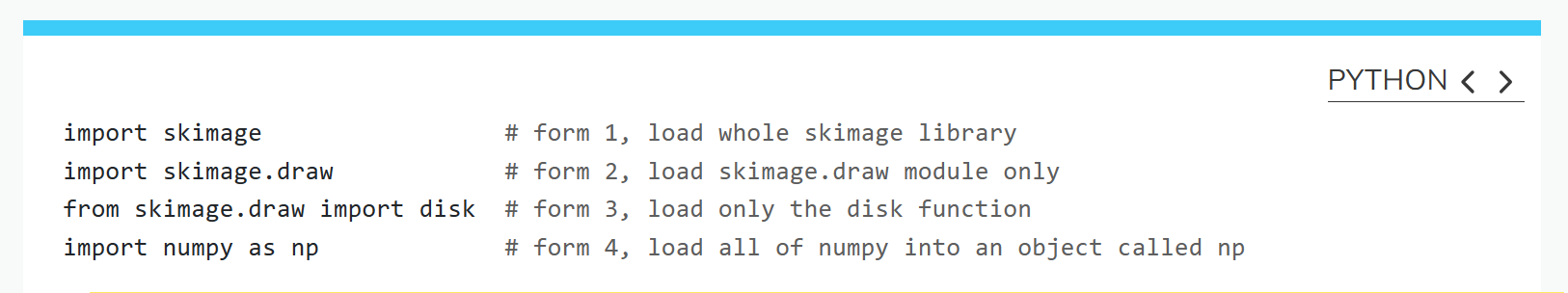
 A NumPy array is a **type** of variable (a simpler example of a type is an integer).

First, the necessary imports:

****



EXAMPLE



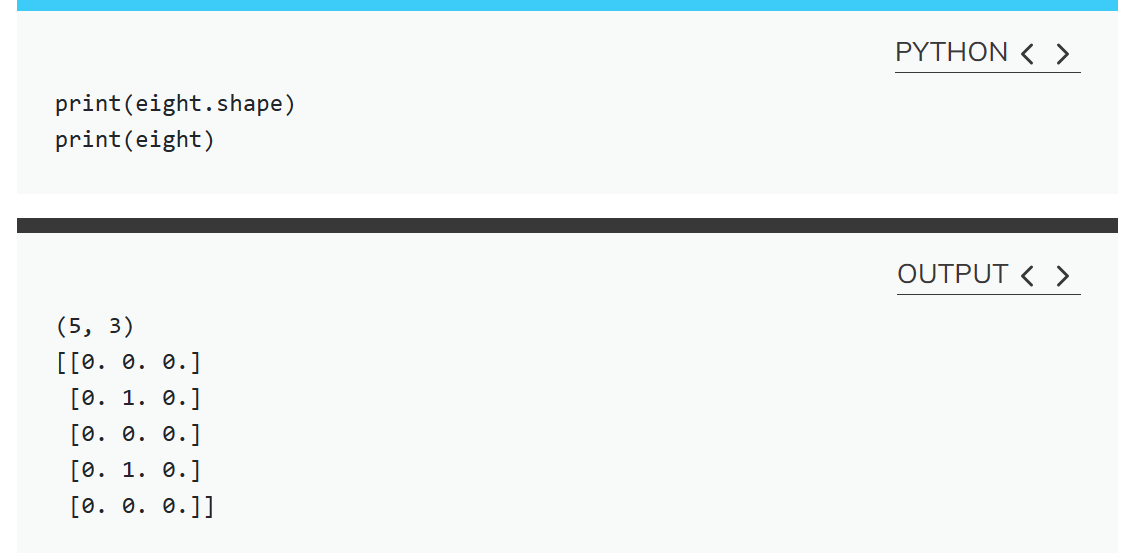
let's load our image data from disk using the imread function from the imageio. v3 module and display it using the imshow function from the matplotlib.pyplot module. imageio is a Python library for reading and writing image data. imageio. v3 is specifying that we want to use version 3 of imageio. This version has the benefit of supporting nD (multidimensional) image data natively .

A purple and yellow rectangle with white numbers

Description automatically generated

You might be thinking, “That does look vaguely like an eight, and I see two colours but how can that be only 15 pixels”. The display of the eight you see does use a lot more screen pixels to display our eight so large, but that does not mean there is information for all those screen pixels in the file. All those extra pixels are a consequence of our viewer creating additional pixels through interpolation. It could have just displayed it as a tiny image using only 15 screen pixels if the viewer was designed differently.

While many image file formats contain descriptive metadata that can be essential, the bulk of a picture file is just arrays of numeric information that, when interpreted according to a certain rule set, become recognizable as an image to us. Our image of an eight is no exception, and imageio.v3 stored that image data in an array of arrays making a 5 x 3 matrix of 15 pixels. We can demonstrate that by calling on the shape property of our image variable and see the matrix by printing our image variable to the screen.



Thus if we have tools that will allow us to manipulate these arrays of numbers, we can manipulate the image. The numpy library can be particularly useful here, so let’s try that out using numpy array slicing. Notice that the default behavior of the imshow function appended row and column numbers that will be helpful to us as we try to address individual or groups of pixels. First let’s load another copy of our eight, and then make it look like a zero.

To make it look like a zero, we need to change the number underlying the centremost pixel to be 1. With the help of those row and column headers, at this small scale we can determine the centre pixel is in row labeled 2 and column labeled 1. Using array slicing, we can then address and assign a new value to that position.



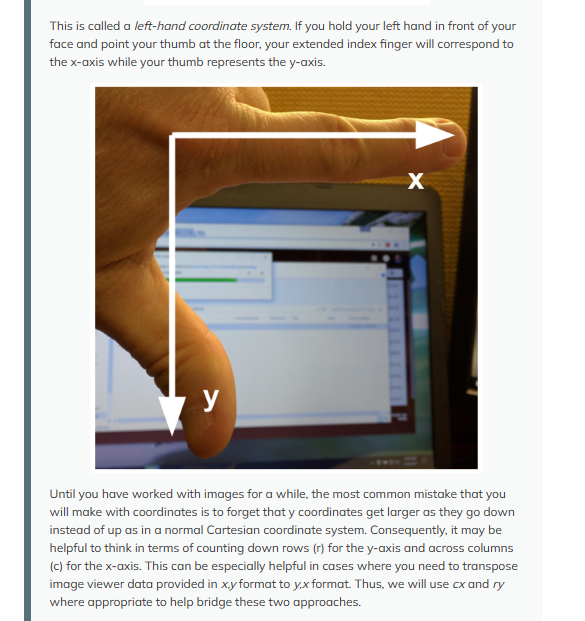
A close-up of a white board

Description automatically generatedA purple rectangle with a yellow rectangle in the middle

Description automatically generated

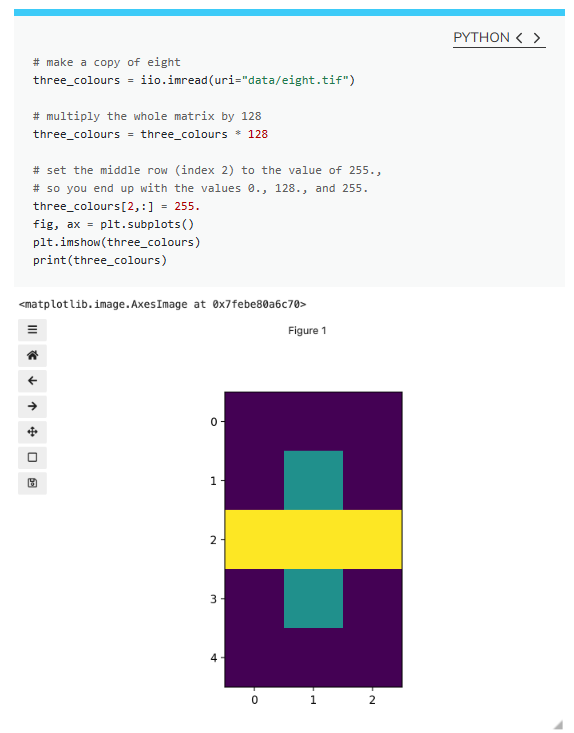
A screenshot of a computer

Description automatically generated

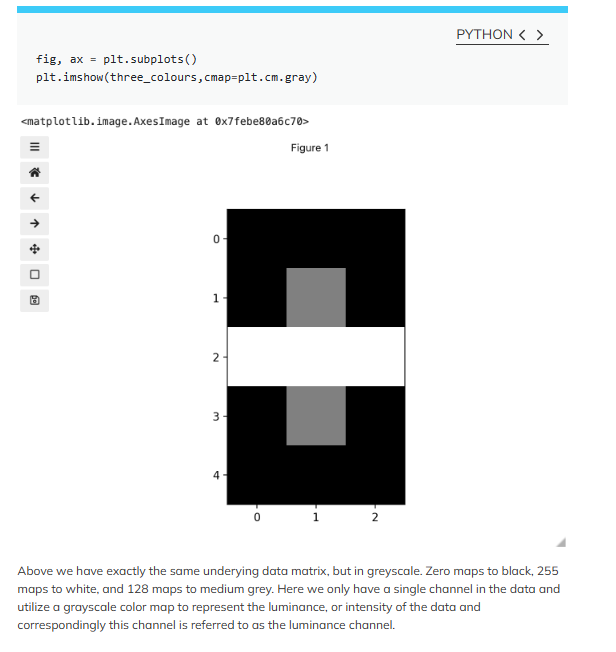


## More colors

Up to now, we only had a 2 colour matrix, but we can have more if we use other numbers or fractions. One common way is to use the numbers between 0 and 255 to allow for 256 different colours or 256 different levels of grey. Let’s try that out.

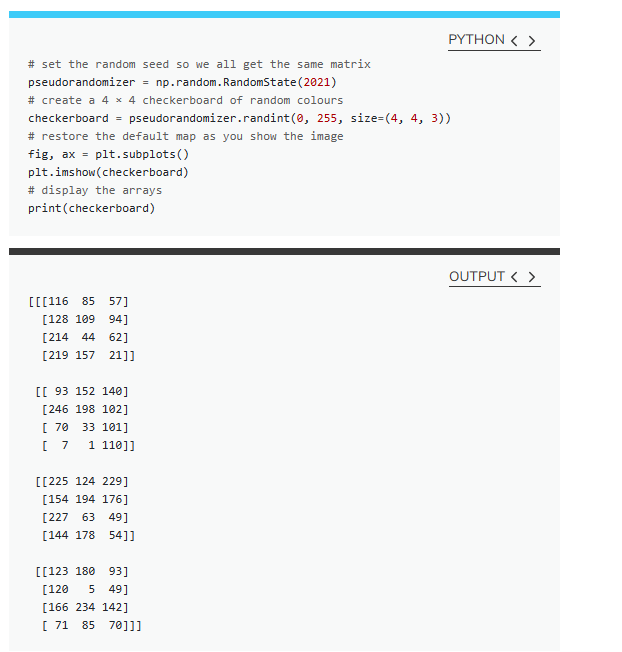


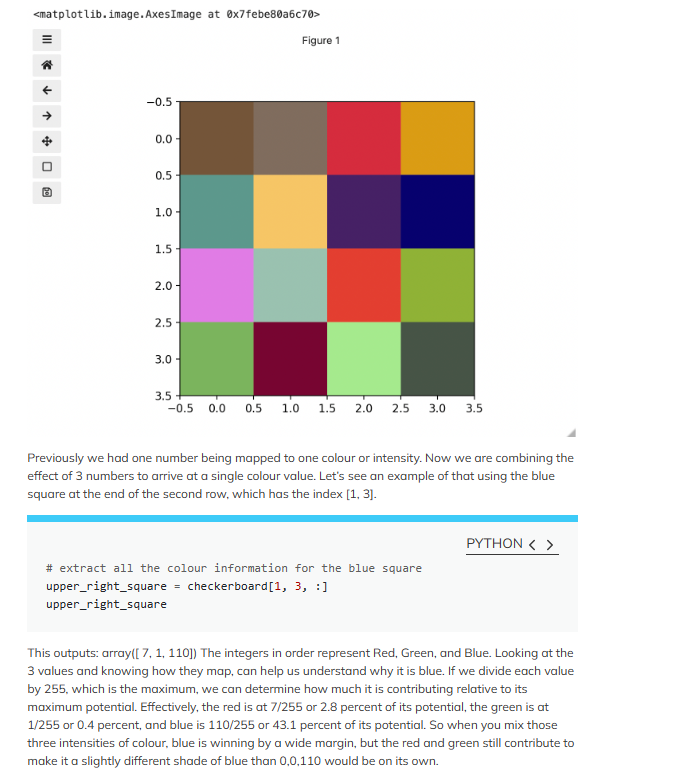
We now have 3 colours, but are they the three colours you expected? They all appear to be on a continuum of dark purple on the low end and yellow on the high end. This is a consequence of the default colour map (cmap) in this library. You can think of a colour map as an association or mapping of numbers to a specific colour. However, the goal here is not to have one number for every possible colour, but rather to have a continuum of colours that demonstrate relative intensity. In our specific case here for example, 255 or the highest intensity is mapped to yellow, and 0 or the lowest intensity is mapped to a dark purple. The best colour map for your data will vary and there are many options built in, but this default selection was not arbitrary. A lot of science went into making this the default due to its robustness when it comes to how the human mind interprets relative colour values, grey-scale printability, and colour-blind friendliness (You can read more about this default colour map in [a Matplotlib tutorial](https://matplotlib.org/stable/tutorials/colors/colormaps.html) and [an explanatory article by the authors](https://bids.github.io/colormap/)). Thus it is a good place to start, and you should change it only with purpose and forethought. For now, let’s see how you can do that using an alternative map you have likely seen before where it will be even easier to see it as a mapped continuum of intensities: greyscale.



Even More Colours

This is all well and good at this scale, but what happens when we instead have a picture of a natural landscape that contains millions of colours. Having a one to one mapping of number to colour like this would be inefficient and make adjustments and building tools to do so very difficult. Rather than larger numbers, the solution is to have more numbers in more dimensions. Storing the numbers in a multi-dimensional matrix where each colour or property like transparency is associated with its own dimension allows for individual contributions to a pixel to be adjusted independently. This ability to manipulate properties of groups of pixels separately will be key to certain techniques explored in later chapters of this lesson. To get started let’s see an example of how different dimensions of information combine to produce a set of pixels using a 4 X 4 matrix with 3 dimensions for the colours red, green, and blue. Rather than loading it from a file, we will generate this example using numpy.





**Your Task:**

### **CHANGING PIXEL VALUES**

Load another copy of eight named five, and then change the value of pixels so you have what looks like a 5 instead of an 8. Display the image and print out the matrix as well.

### **HINKING ABOUT RGBA COLOURS**

You are working on an image processing task using Python, and you have an RGBA image that you want to manipulate by changing its transparency (alpha channel) to create various transparency effects. Your goal is to load the original RGBA image, adjust the alpha channel values to achieve the desired transparency effect, and save the modified image.

**Here are the steps you need to follow:**

1. Load the original RGBA image using iio.imread().
2. Define a new alpha value that represents the desired transparency level (0 for fully transparent, 255 for fully opaque).
3. Create a new image with the same dimensions as the original.
4. Iterate through each pixel in the original image, preserving the RGB color values, and update the alpha channel value with the new alpha value.

**Write Python code to accomplish these tasks and create a program that allows users to input the original image path and the desired transparency level. The program should then load the image, apply the transparency effect, and save the modified image.**

Hint

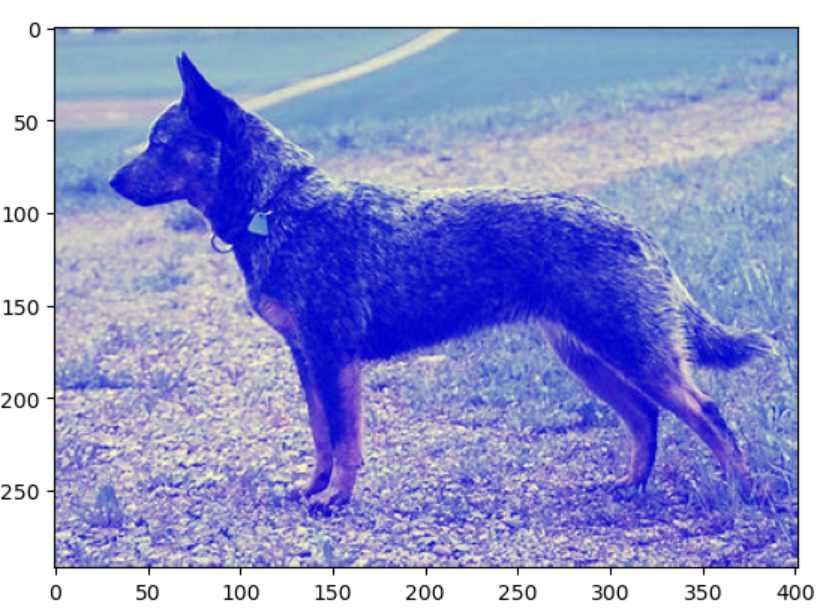
**Example Output:**

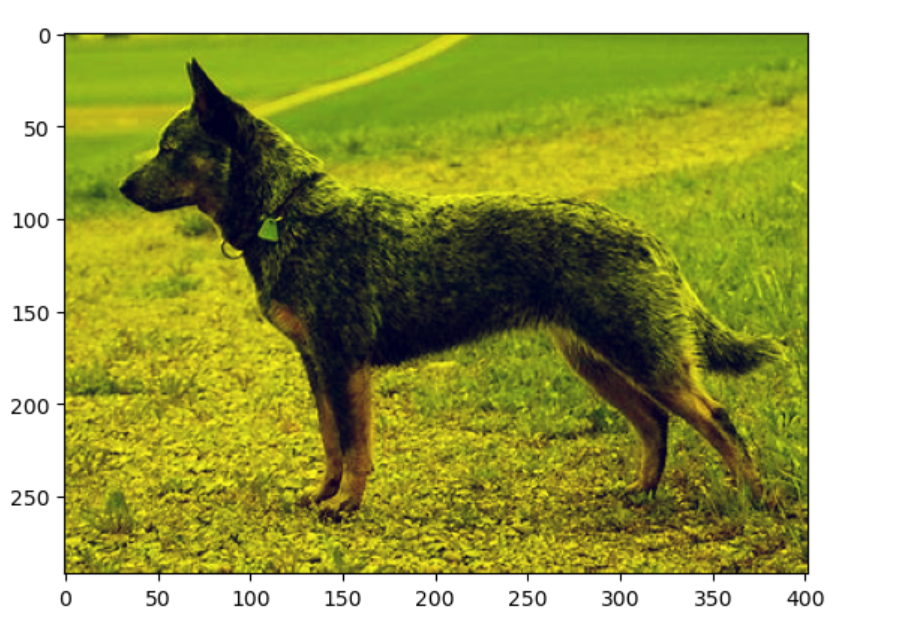


Original Image



Output image





**Assessment**

1. Each student will show all the above parts running as demo to the Lab Instructor **before leaving the lab.** Total marks for the lab is as follows

|  |  |
| --- | --- |
| Task 1 | Marks (demo + report) |
| 1 | 10 |
| 2 | 10 |
| Total | 20 |

1. Students will prepare a report in which they will submit the snapshots taken while they worked on each part. They will explain the figures to make sure that they understood what they did.

**References:**

[Image Processing with Python: Image Basics (fishtree-attempt.github.io)](https://fishtree-attempt.github.io/image-processing/02-image-basics.html#top)