Some of the steps in the video for this particular lesson have been updated, please see the lesson notes below for the corrected version.

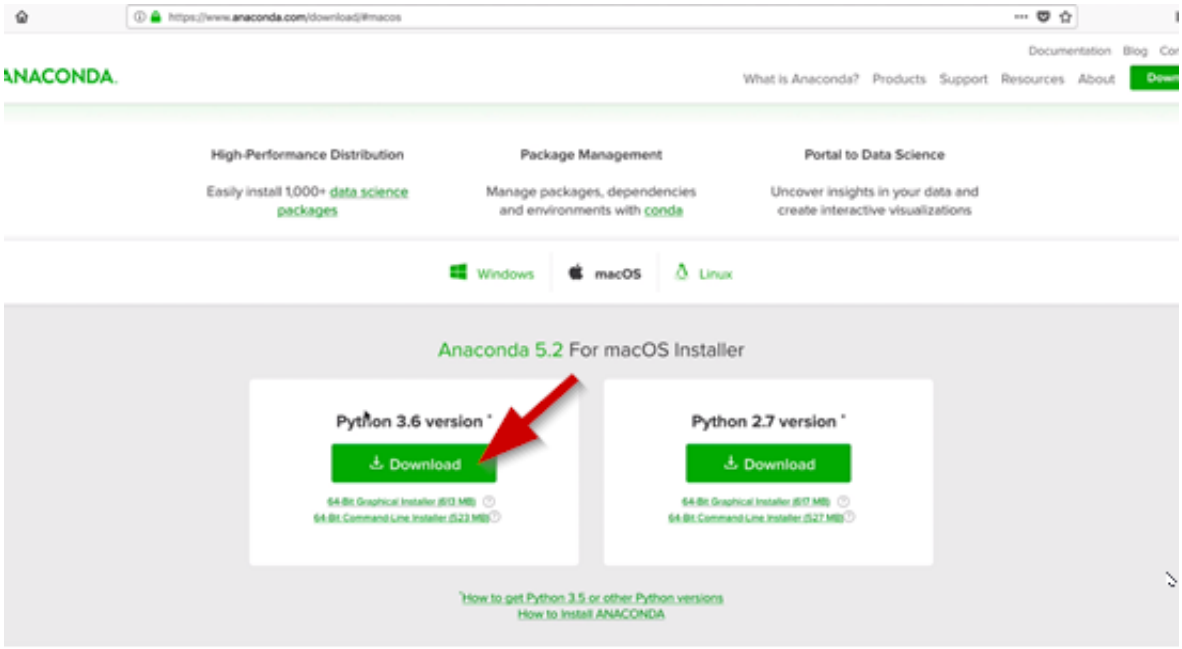
In this lesson, you will learn how to set up the development environment.

For managing our dependencies we are going to use this tool called **Anaconda**. Essentially you can use Anaconda to manage our dependencies and update them very easily.

You can download Anaconda from here: https://www.anaconda.com/

The direct link is here: Anaconda Download

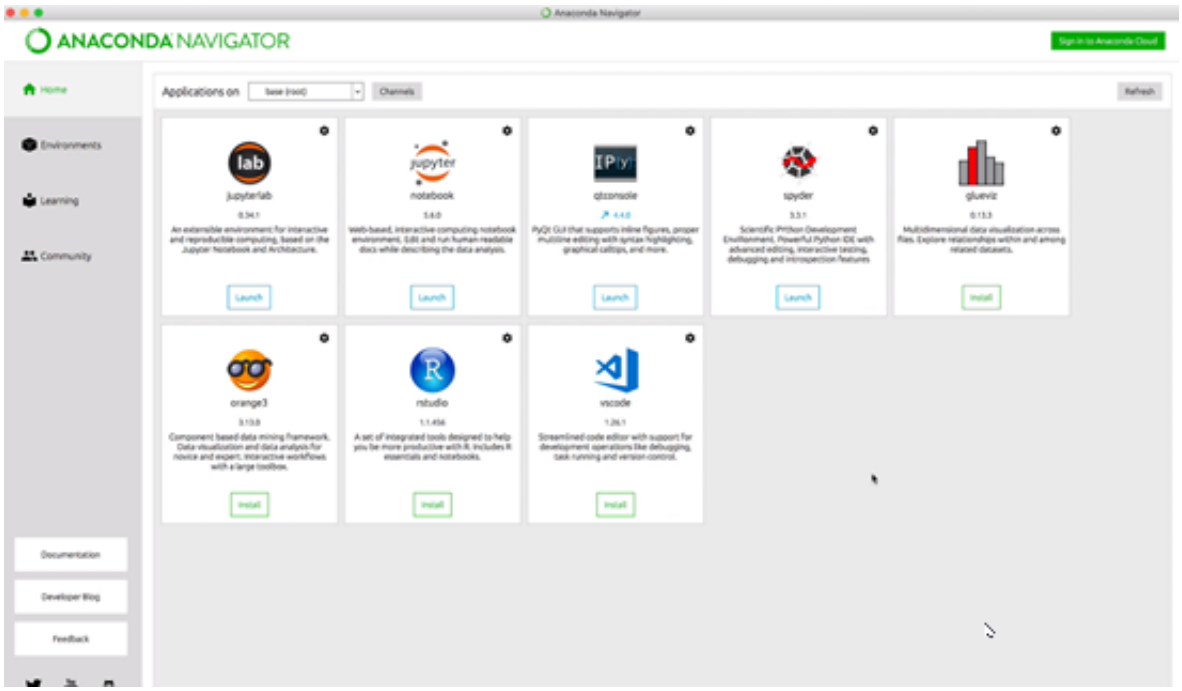
Choose the version of Anaconda you need based on the current operating system you are using.



Get Started

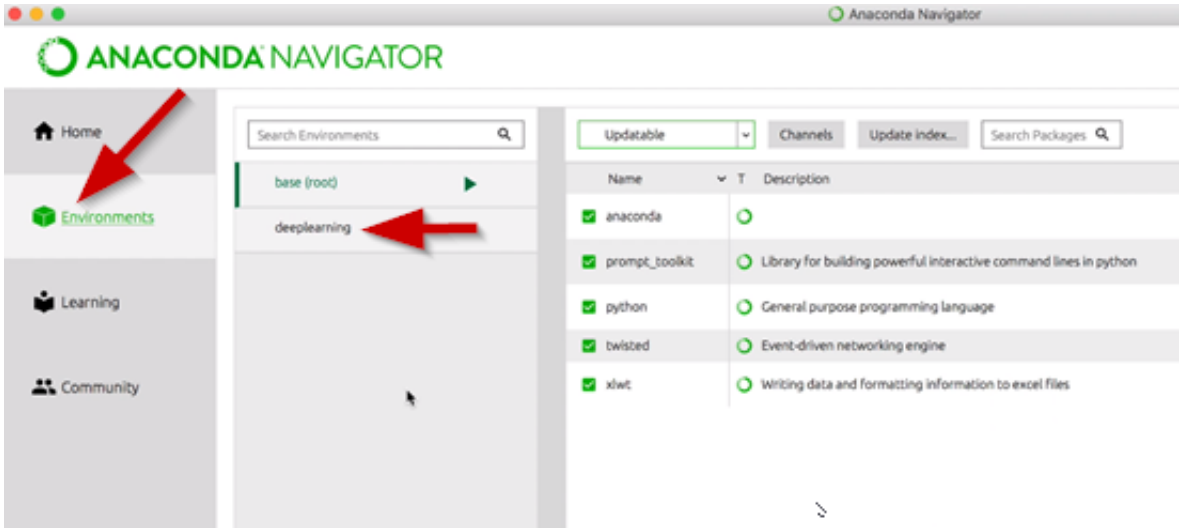
Once you have Anaconda downloaded go ahead and follow the setup wizard's instructions.

After installed, it should look like this:

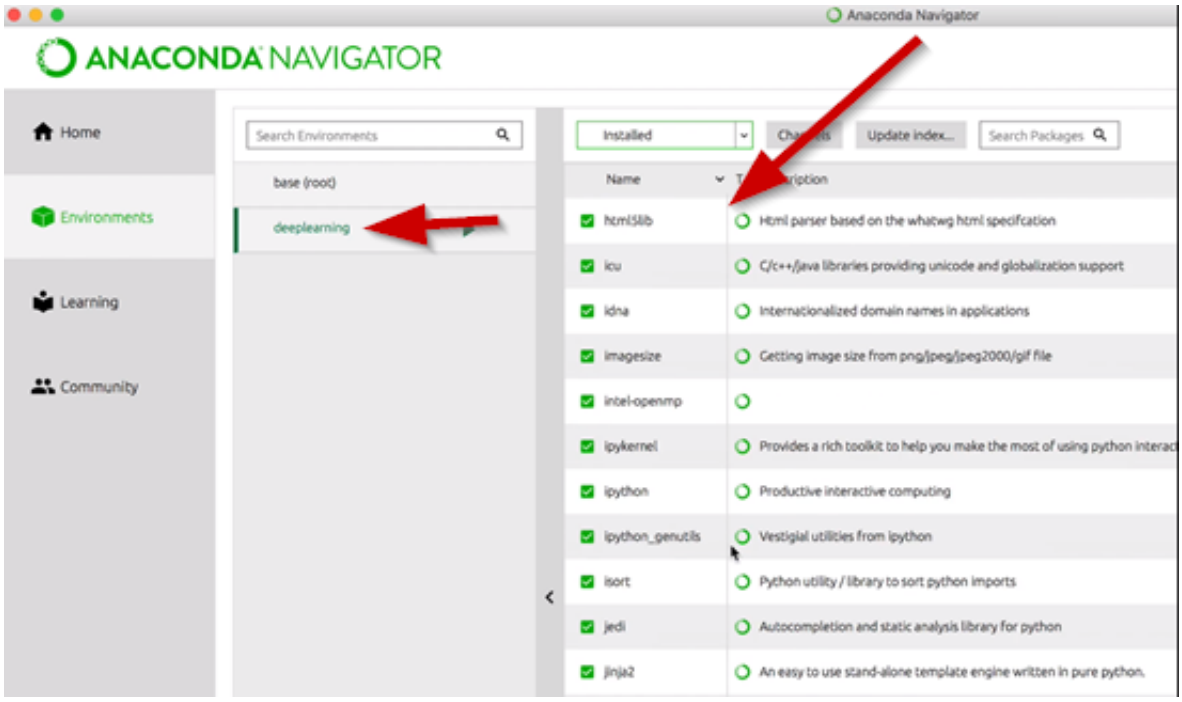


This already has all our dependencies bundled into these things called environments.

Select the **Environments** tab from the Anaconda menu:

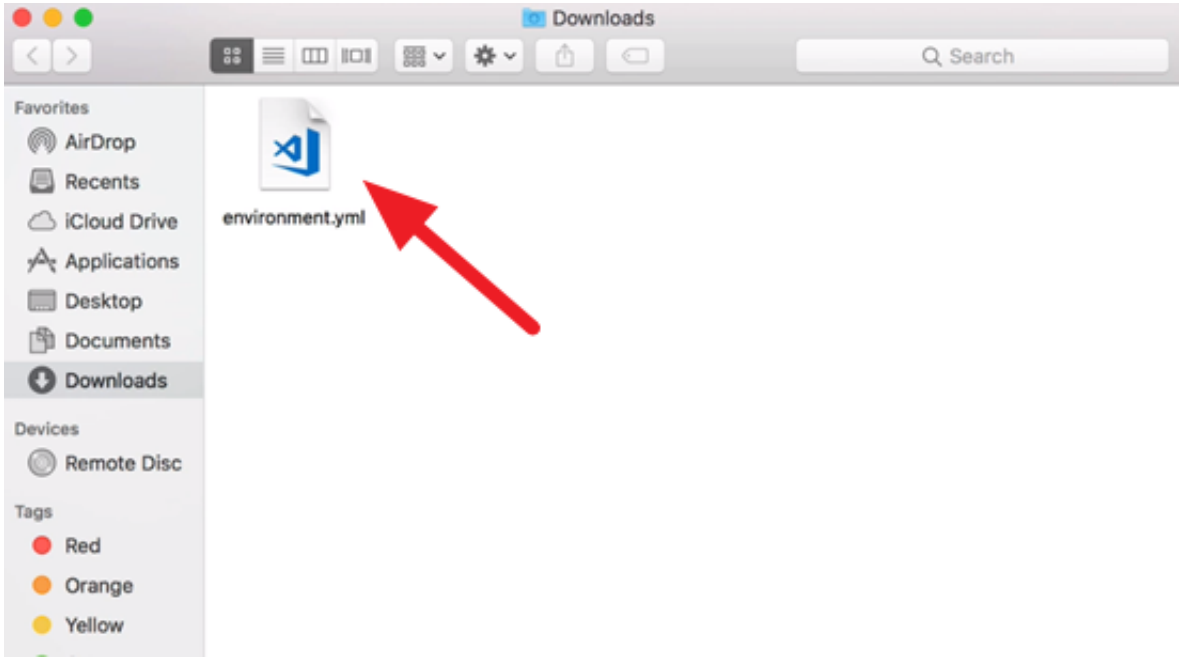


You can change your environment by selecting one of them in the menu. So if you clicked on the deeplearning one, you will then see all of its packages listed on the right.

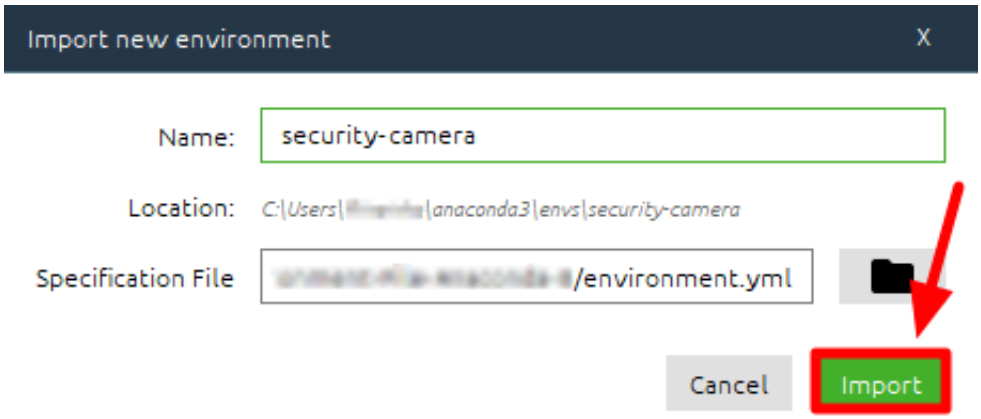


The following step has been updated, and differs from the video:

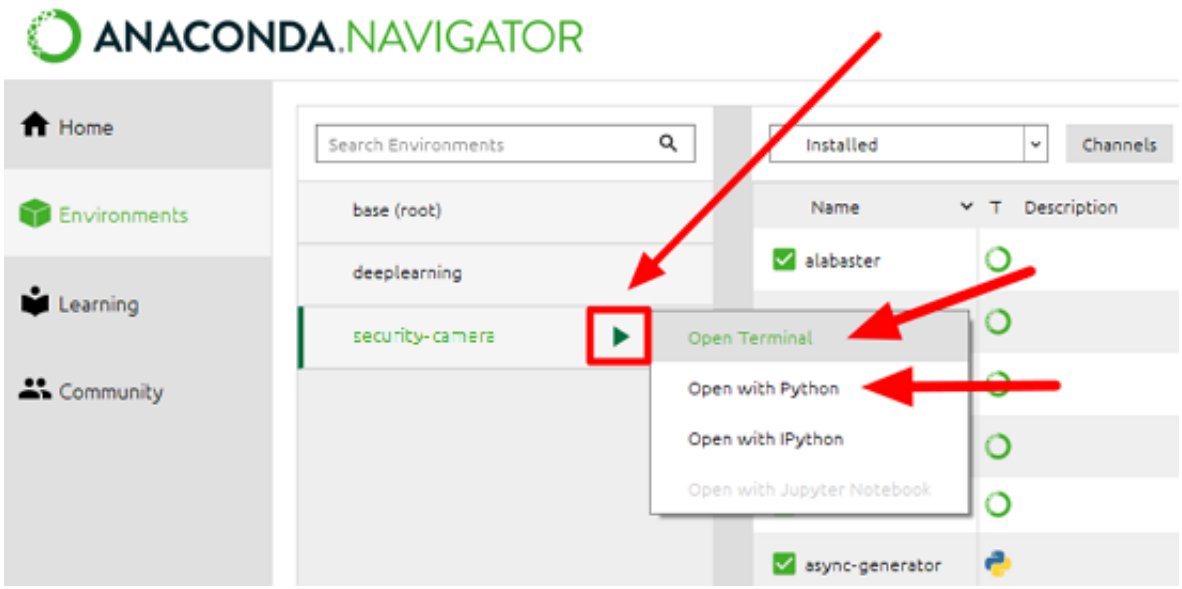
The environment we're going to use can be **downloaded** from the **Course Home page**. After you download it, you will need to **extract** its contents:



All you need to do now is go to Anaconda and **import** the **environment**. It'll pull up a window for you to select the environment file you now have:



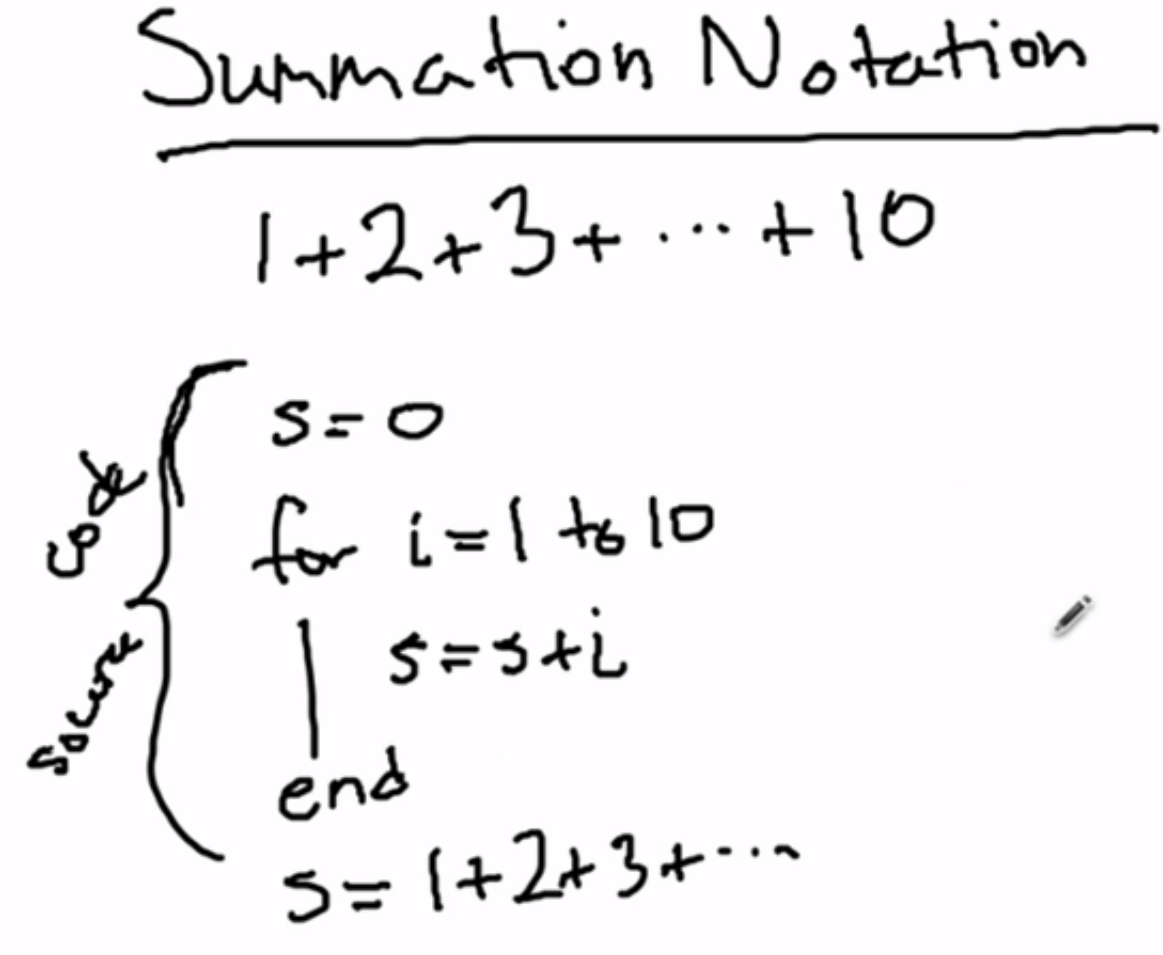
After the environment is done loading, you will see it listed along with the other available environments. If you **click** on the **green arrow** icon next to the imported environment, you will pull up a **drop-down menu** and from it you can **open a terminal** or a Python interpreter:



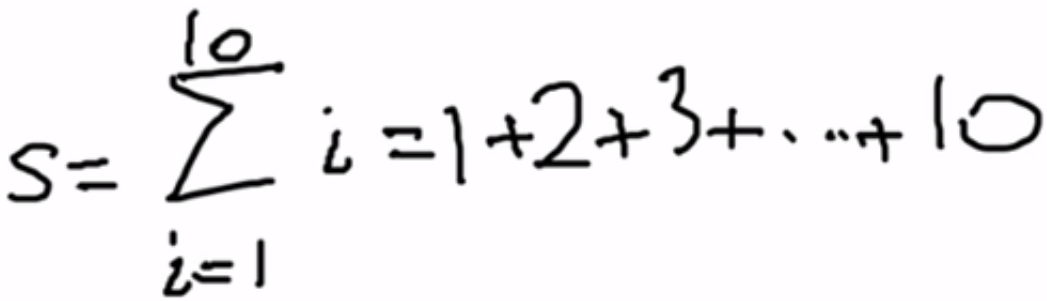
In this lesson we will discuss a new mathematical notation that will help us in the future with concepts especially with image similarity.

For example, what if we wanted to sum up the numbers from 1-10? So this would be the sum of all the numbers between one and ten, including one and ten.

So we could come up with some source code using a programming language to do this computation using a for loop.



We can use **Summation Notation** from mathematics to do this computation:



This notation acts like a for loop, where we keep accumulating a value or keep adding a value, which

is why it's called a sum or summation.

We will be using summation later on when we are discussing image similarity.

Another example for summation notation:

$$S' = \sum_{j=5}^{15} j+1 = (5+1)+(6+1)+(7+1)+\cdots+(15+1)$$

Let's now apply this too images in specifics.

For example, we have an image that is 3X3.

$$A = \begin{bmatrix} 0 & 100 & 50 \\ 0 & 60 & 60 \\ 0 & 200 & 50 \end{bmatrix}$$

Why we need summation notation is because a very common thing to do is take the sum of all the pixels in an image. So we can start by doing the rows first:

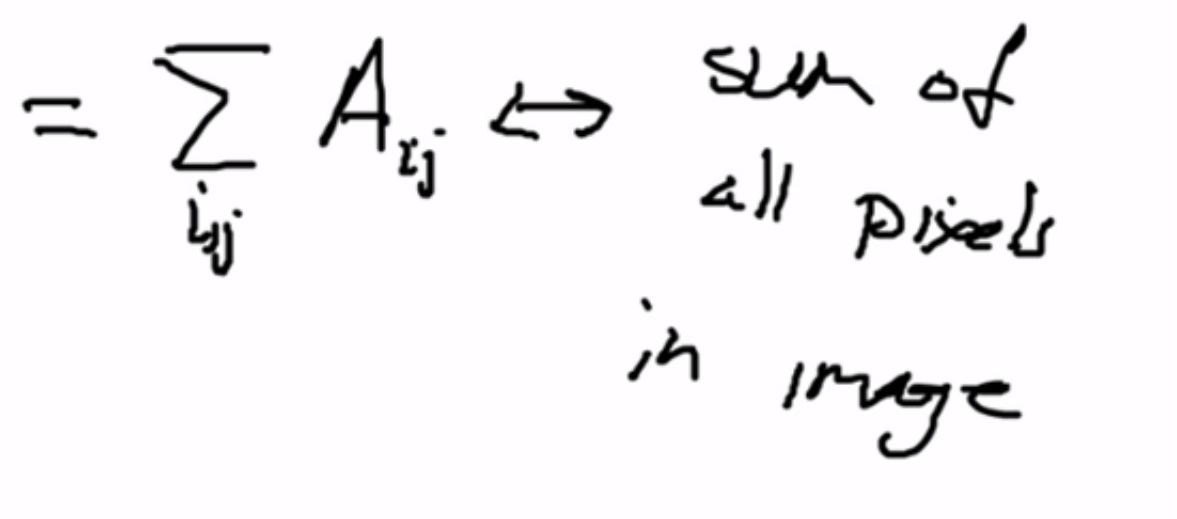
$$S = (0+100+50)+(0+60+60)+(0+200+50)$$

We can then make this a little easier:

$$= (A_{11}+A_{12}+A_{13})+(A_{21}+A_{22}+A_{23})+(A_{31}+A_{32}+A_{33})$$

$$= \sum_{i=1}^{3} A_{1i} + \sum_{j=1}^{3} A_{2j} + \sum_{k=1}^{3} A_{3k}$$

And as a shorthand we can do this:



In this video you will see a demonstration on how we can perform the same kind of summation operations over matrices in Numpy so that you can become more accustomed to it.

See the code below:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np

A = np.ones((3, 3))

s = 0
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        s += A [i,j]
print(s)
```

This is the long way around it all, but numpy being a scientific computing library and all, is really optimized for doing a lot of neat things.

But, we do need to know what functions to use.

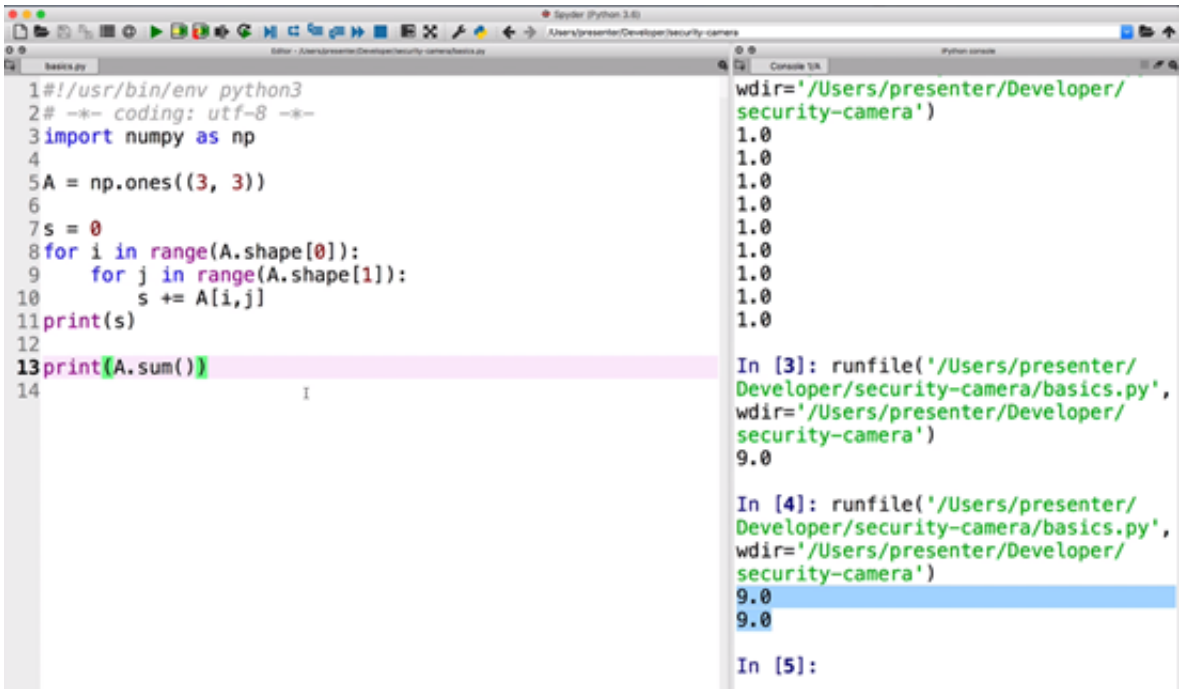
So for summing all of the values in a matrix, there is a function that's simply called sum.

So we can call sum on our numpy matrix A, and that will print out the same value.

See the code below:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np

A = np.ones((3, 3))

s = 0
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        s += A [i,j]
print(s)
print(A.sum())
```

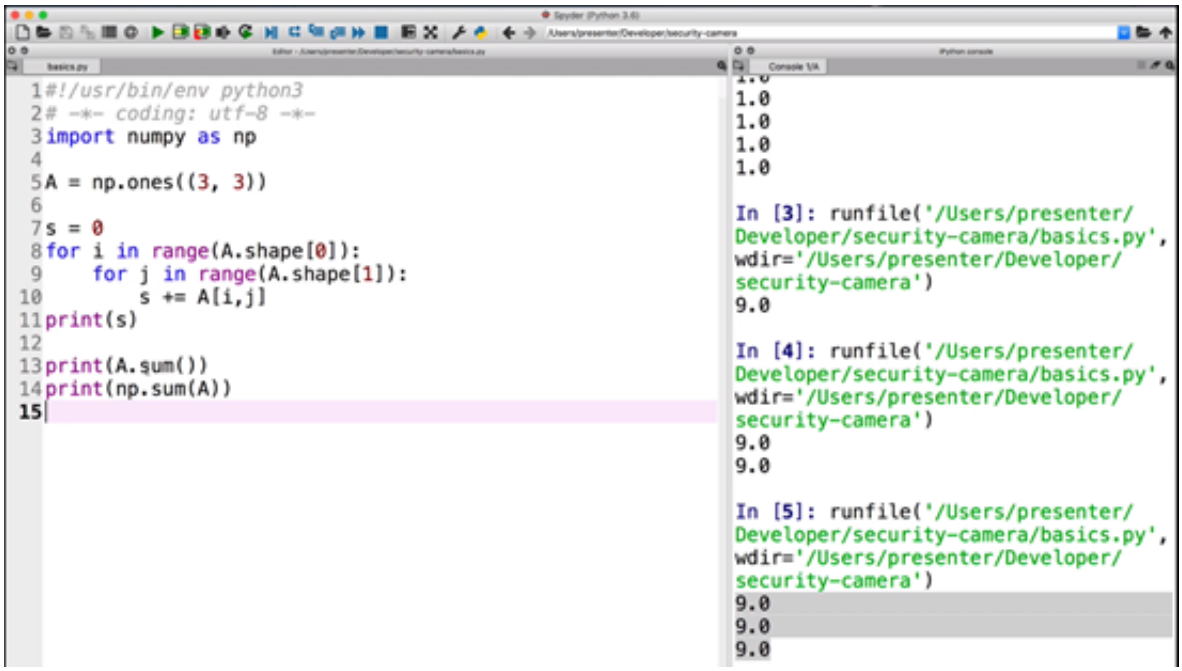
When this code is ran it will print out the same value for both:



And we can also use this numpy function, see the code below:

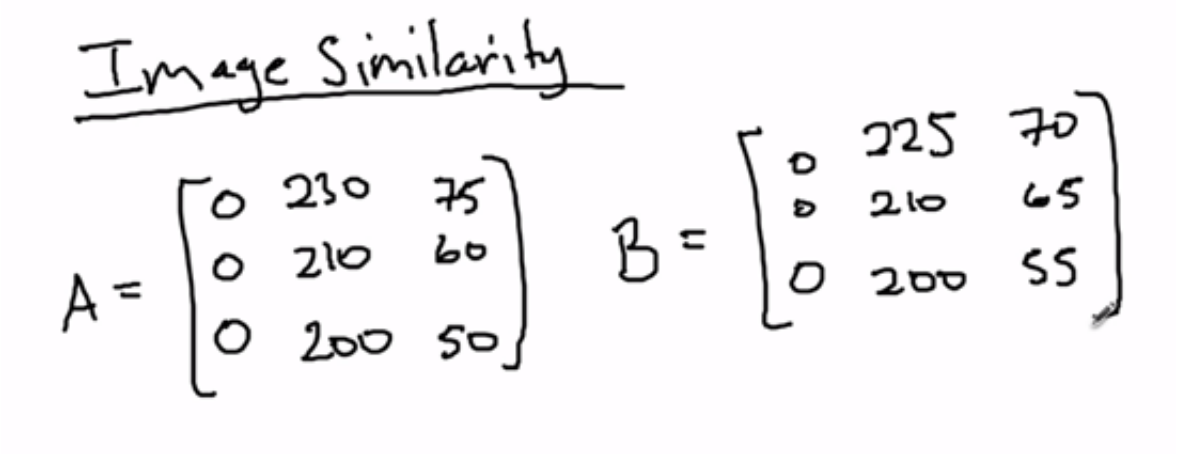
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np

A = np.ones((3, 3))

s = 0
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        s += A [i,j]
print(s)

print(A.sum())
print(np.sum(A))
```

And again we get the same value again for all three functions.



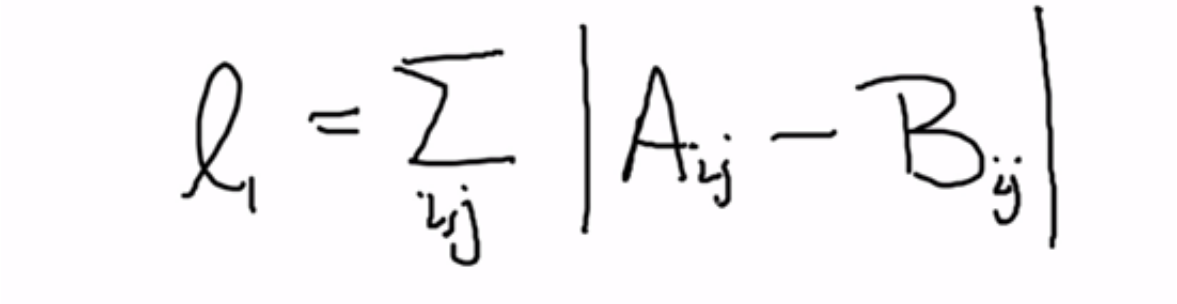
In this lesson we will discuss different metrics of image similarity. As humans we can easily differentiate between images. A computer only has access to these pixel values, so the question is how can we determine difference using pixel values in the computer's perspective?

So let's suppose we have two matrices:

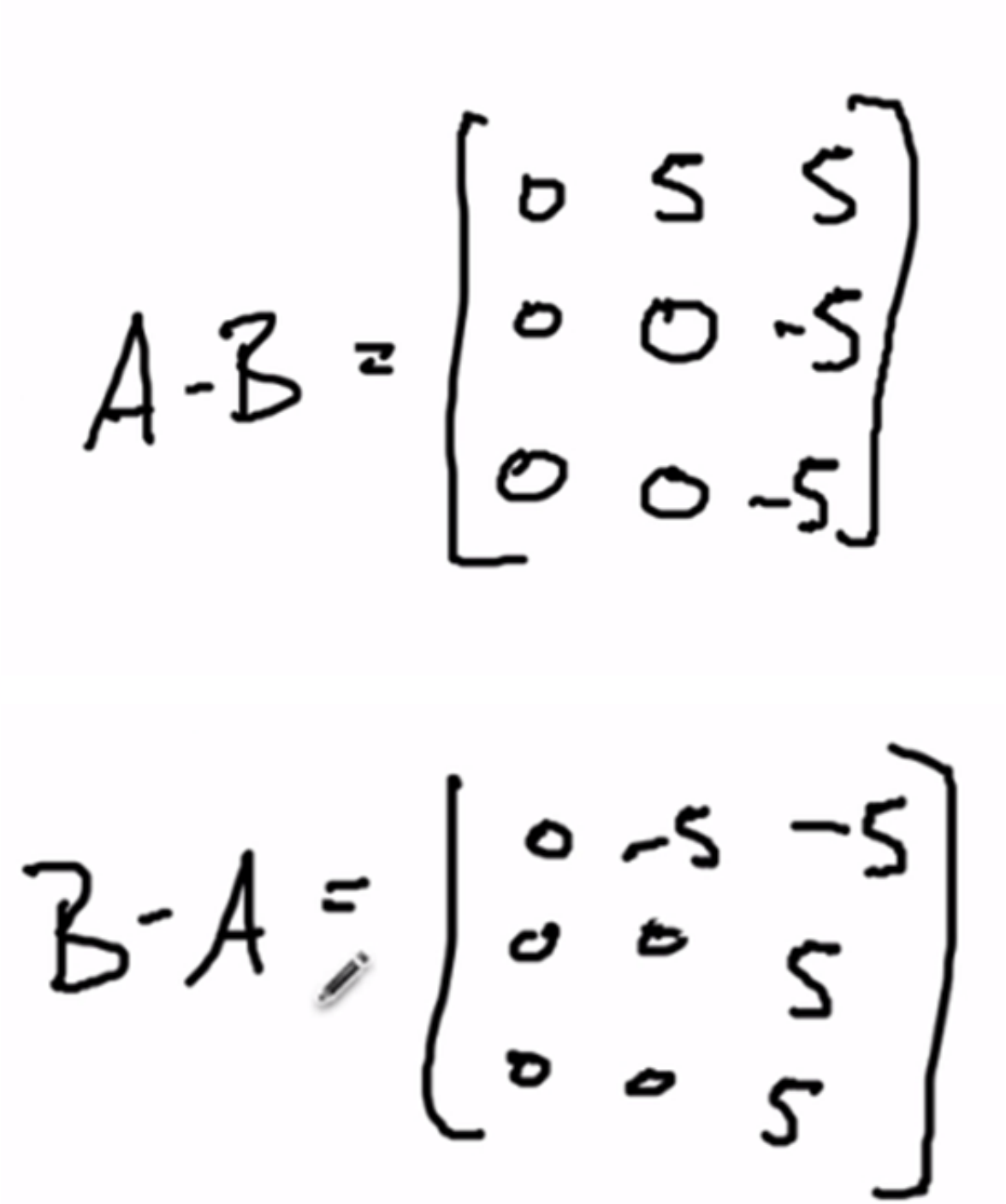


So how do we know that these two images are not the same?

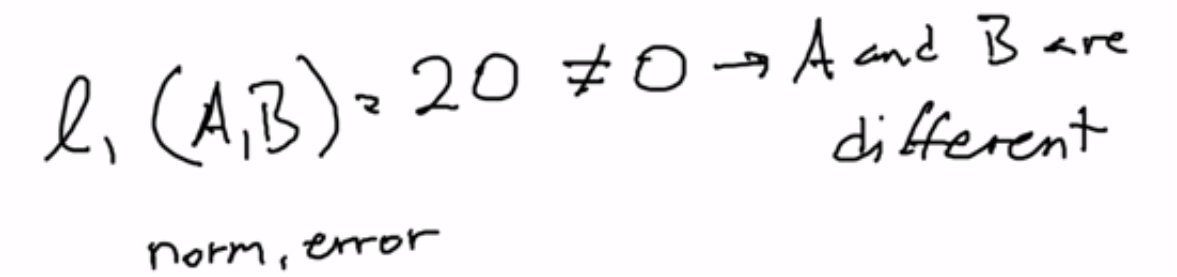
Well if they were identical images they would have the same pixel values in the corresponding coordinates. Meaning there is no difference in the pixel values.



Its important always account for **absolute value with image similarity.**



L1 Norm or error is used to compare the difference between two images:

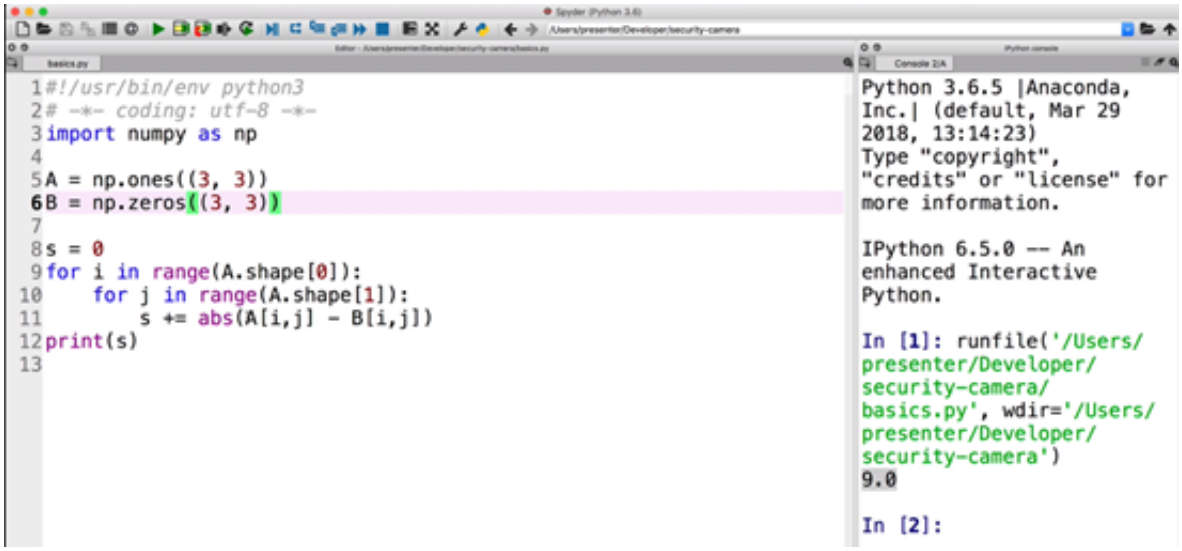


In this lesson you will be shown how to compute the sum of the absolute difference using numpy.

See the code below:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np

A = np.ones((3,3))
B = np.zeros((3,3)))

s = 0
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        s += abs (A[i, j] - B[i,j])
print(s)
```

When the code is ran we get the value 9 printed out.

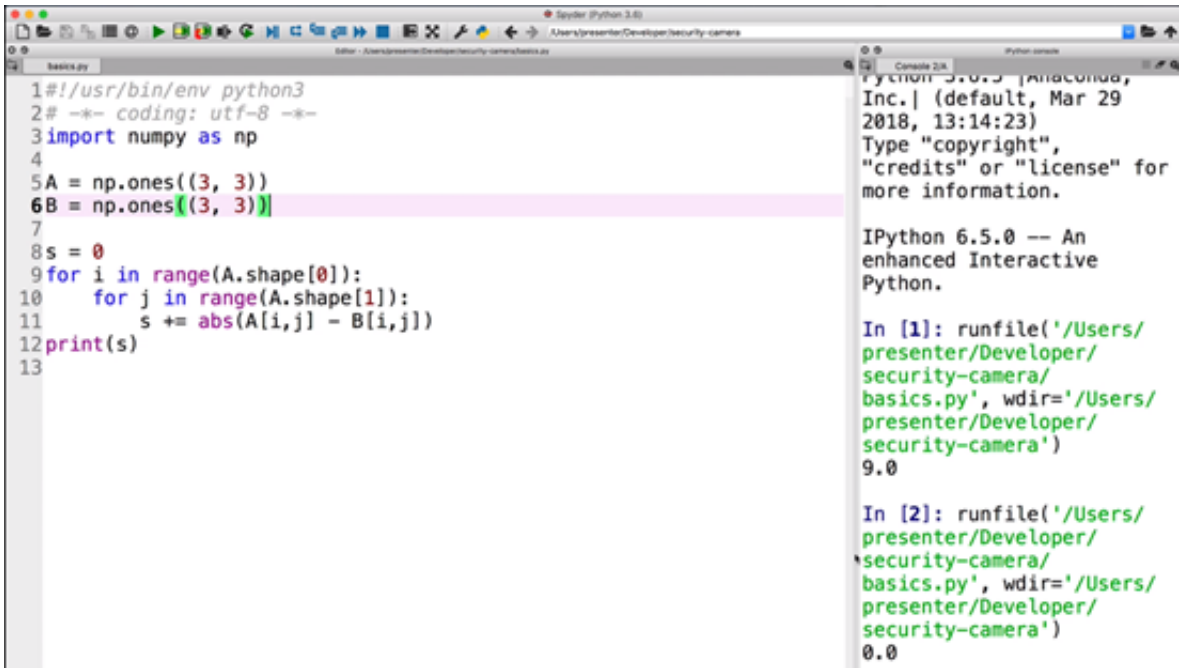


We are going to replace matrix B with ones now instead of the zeros.

See the code below:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np

A = np.ones((3,3))
B = np.ones((3,3)))

s = 0
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        s += abs (A[i, j] - B[i,j])
print(s)
```


When the code is ran we get the value 0 printed out.

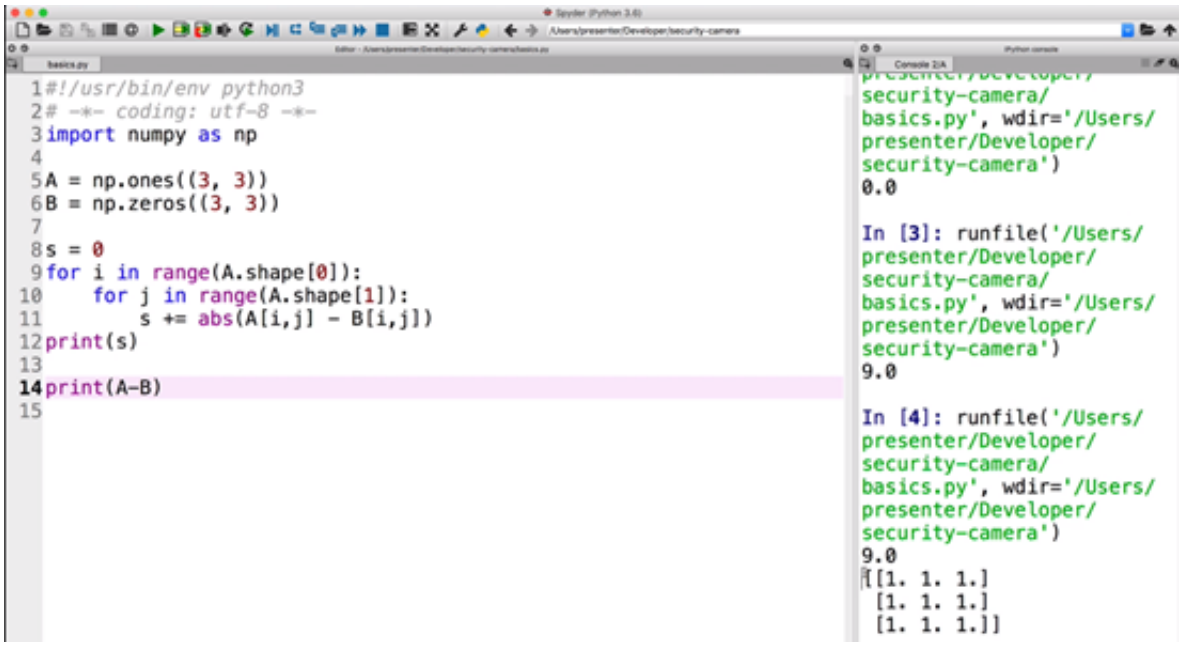


These matrices are identical.

Now we will use a numpy function, see the code below:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np

A = np.ones((3,3))
B = np.zeros((3,3)))

s = 0
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        s += abs (A[i, j] - B[i,j])
print(s)

print(A-B)
```

When this code is ran we get a matrix of ones printed out:



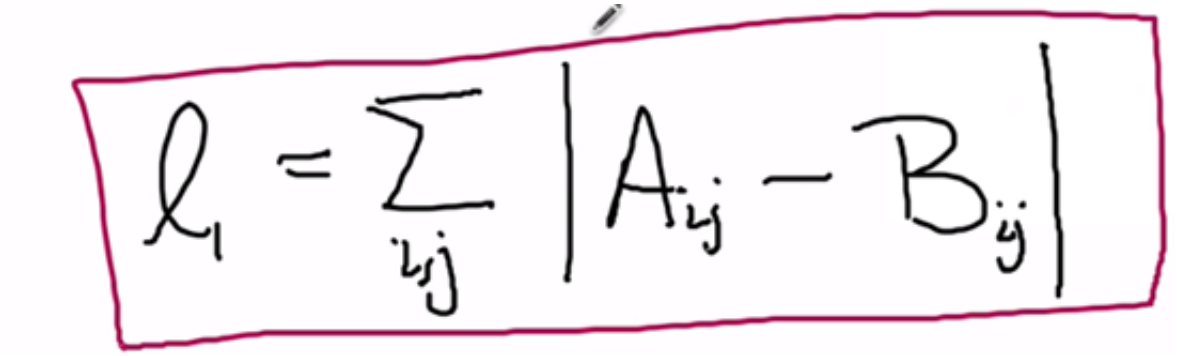
See the code below:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np

A = np.ones((3,3))
B = np.zeros((3,3)))

s = 0
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        s += abs (A[i, j] - B[i,j])
print(s)

print(abs(A-B).sum())
```


In this lesson we will improve upon the L1 Norm.

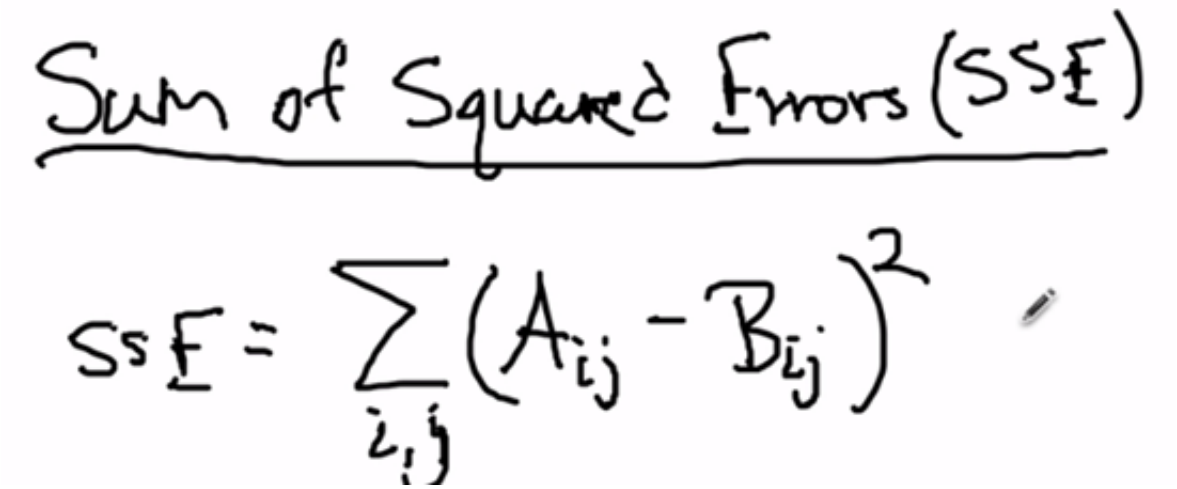
The L1 Norm can be improved upon a little bit, and what we can do is replace the absolute value.

Here is the example of the formula we used earlier for the L1 Norm:



So what we can do is **replace the absolute value with a squared**, and the result is what we call **sum of squared errors, or SSE.**

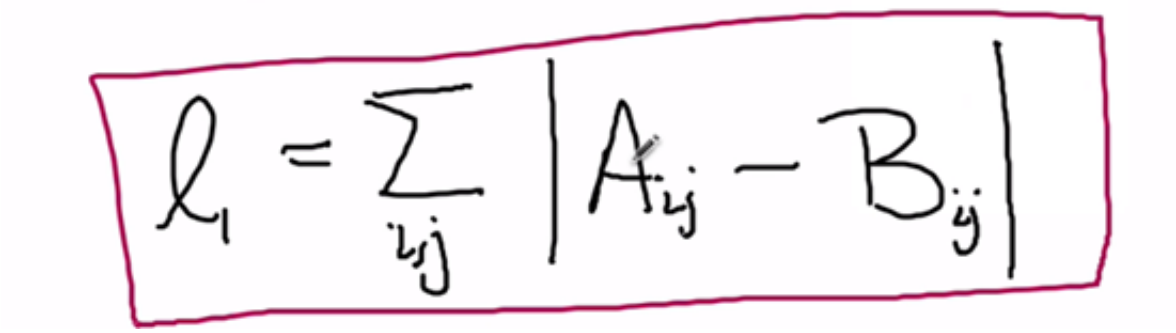
So therefore the **SSE is equal to Sum over IJ, I J is all the pixels then AIJ minus BIJ and instead of taking the absolute value we are actually going to square it.**



Squaring has all the same benefits as taking absolute value.

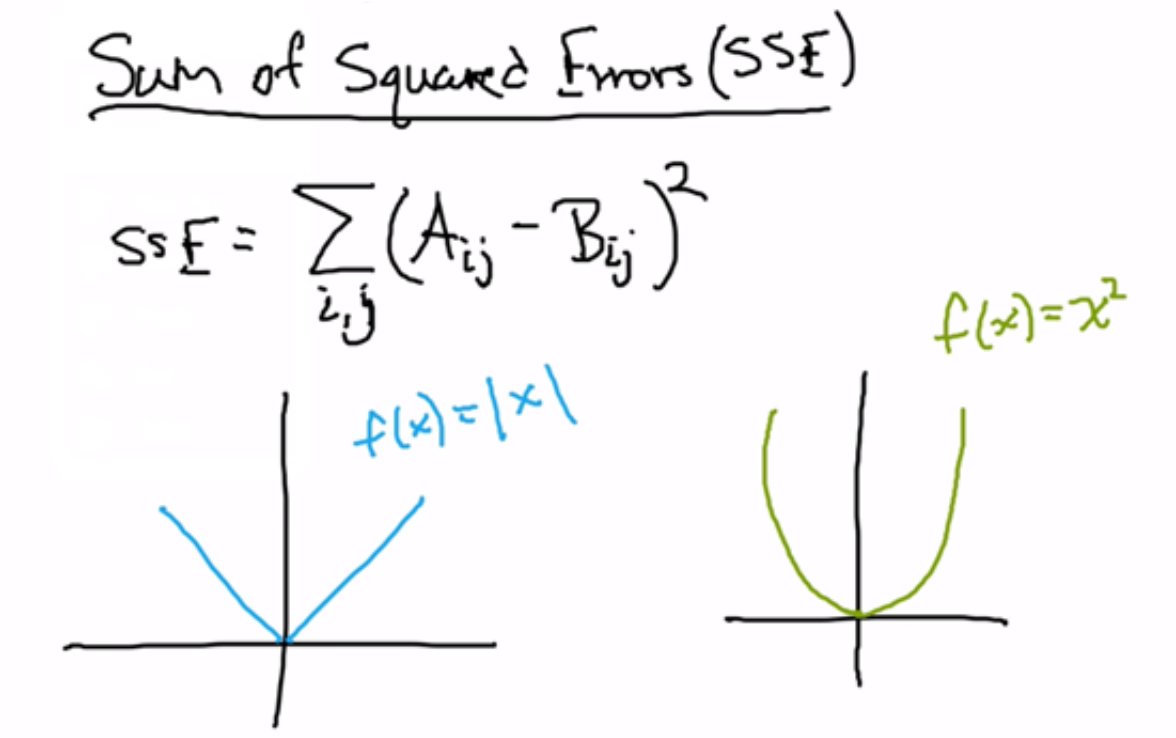
We need to take absolute value in the first place because if we do not our error could equal zero, and if it was zero that would mean the two images were the same, but we saw in that particualr case in the previous lesson that wasn't the good way to compute this by just taking the pure difference.

So that is why we have this absolute value:



So an error of zero means that there's no error and that the images are the same. If anything greater than that then there's some error and these images are different in some way.

Squared will also make all the numbers positive.

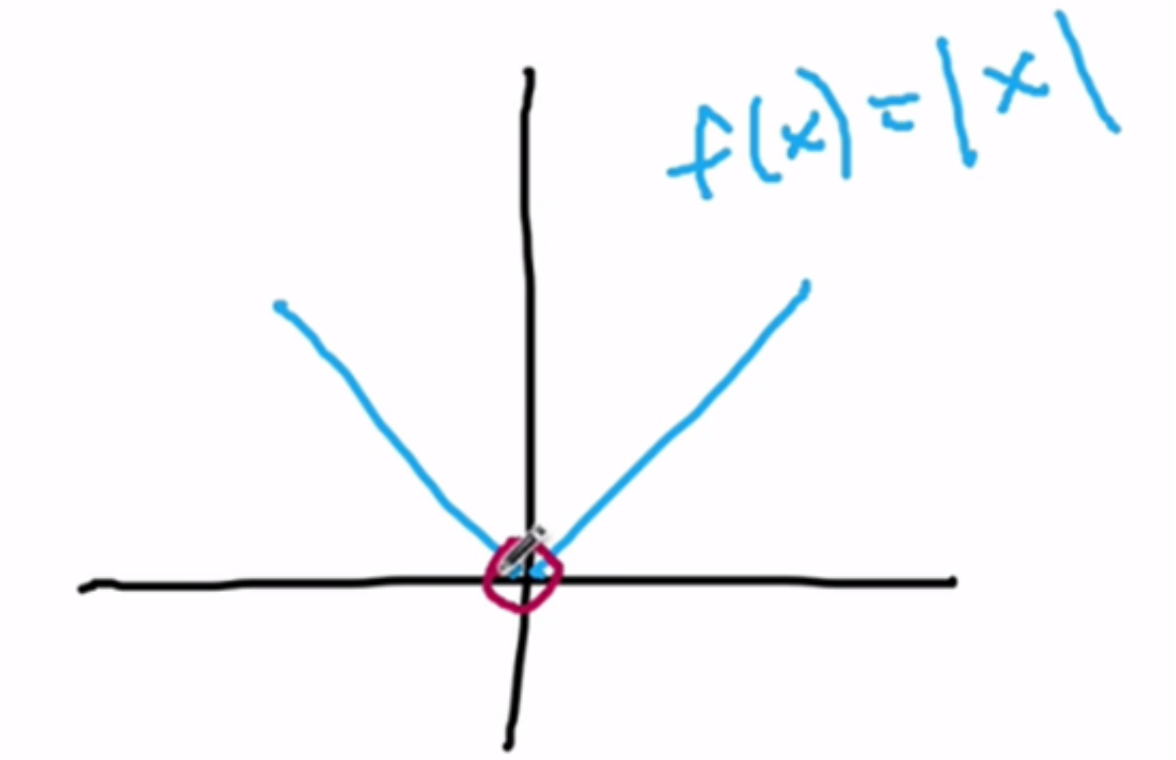


So with using theses graphs above to show that F of X and G of X, there is no way to get a negative number, basically, with this, absolute value is always positive and squared is also always positive.

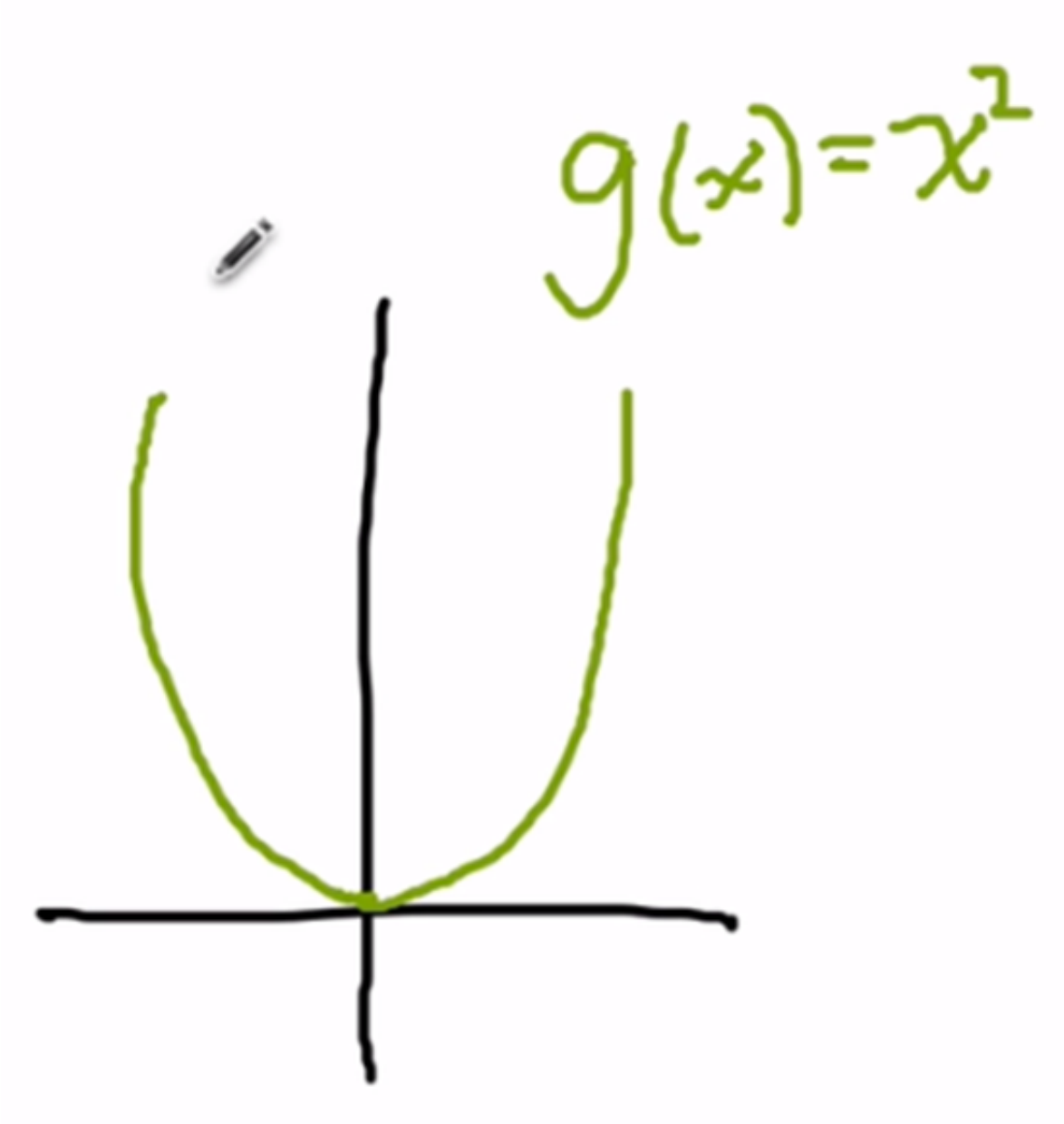
But, there is no way to get a negative number, so we are safe in either of these cases. So we can substitute squared in for absolute value and then if we were to compute this, this norm error would actually be different. It would satisfy the condition that if these two images were the same then you would get zero. If they were different you would get something that's not zero and we can't have a negative number with using the graphs.

We use squared instead of absolute value because the short answer for this is calculus.

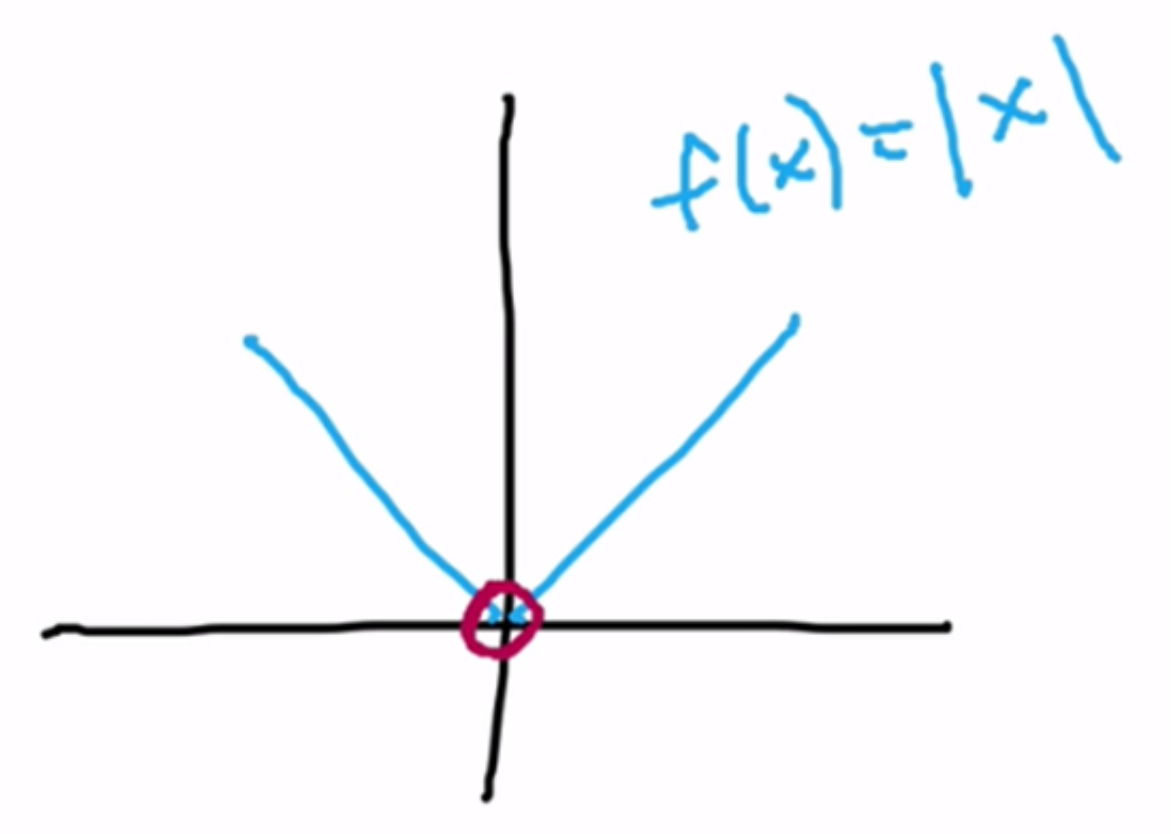
So X squared is a smooth function. Absolute value isn't a smooth function because we have this point here:



And you may now be asking why does it matter if it’s pointy or smooth, and it makes a big difference when you get into the calculus of it. Because this squared function is nice and smooth:



With any sort of calculus you need to do, the derivative of this turns out to be really nice. Where the derivative of this is not so nice:



So its not easy to work with, so usually when we're dealing with this kind of error, we generally prefer squared over absolute value. Smooth functions have a lot of nice properties that pointy functions like absolute value do not.

Specifically in terms of calculus, we won't actually get into any of the calculus, because that is beyond the scope of this course. The reason that we use squared is because this sum of squared error isn't just used for image processing, it's used for so many other things.

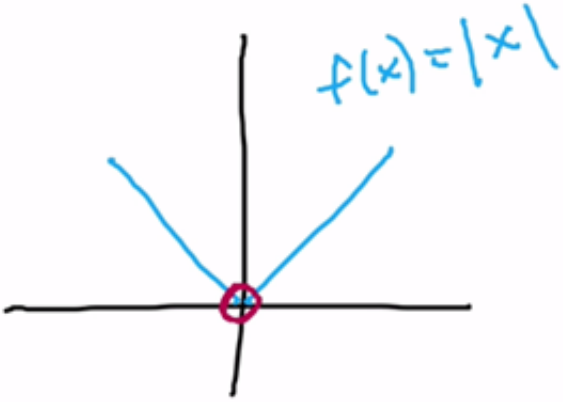
Things like you know, artificial intelligence and machine learning like to use something of this form, except these are really not matrices.

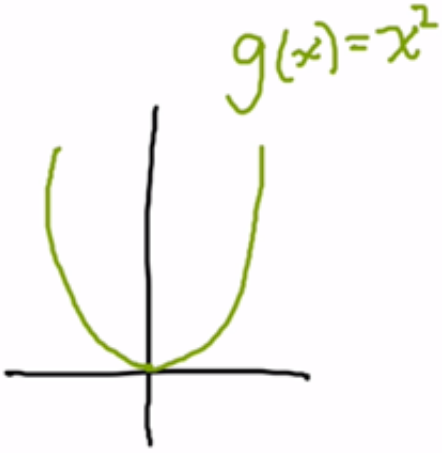
Because we prefer squared when it comes to absolute value because when we are trying to figure out how to update the, like the weights and the biases of a neural network, it turns out this is much easier to use then absolute value, and so there's a ton of different applications for this, like A.I. machine learning, and statistics.

There is another reason why squared is preferred to be used versus absolute value and that is because the errors tend to be exaggerated. So if there's a large error between two, then the squared is gonna exaggerate that a lot, because when you are squaring that value it becomes even larger. So squared also has a kind of added benefit of exaggerating differences, but **in general we prefer squared over absolute value, and that's why we have this thing called Sum of Squared Errors or SSE, and the formula looks like this:**

Sum of Squared Errors (SSE)

$$SSE = \sum_{i,j} (A_{ij} - B_{ij})^2$$





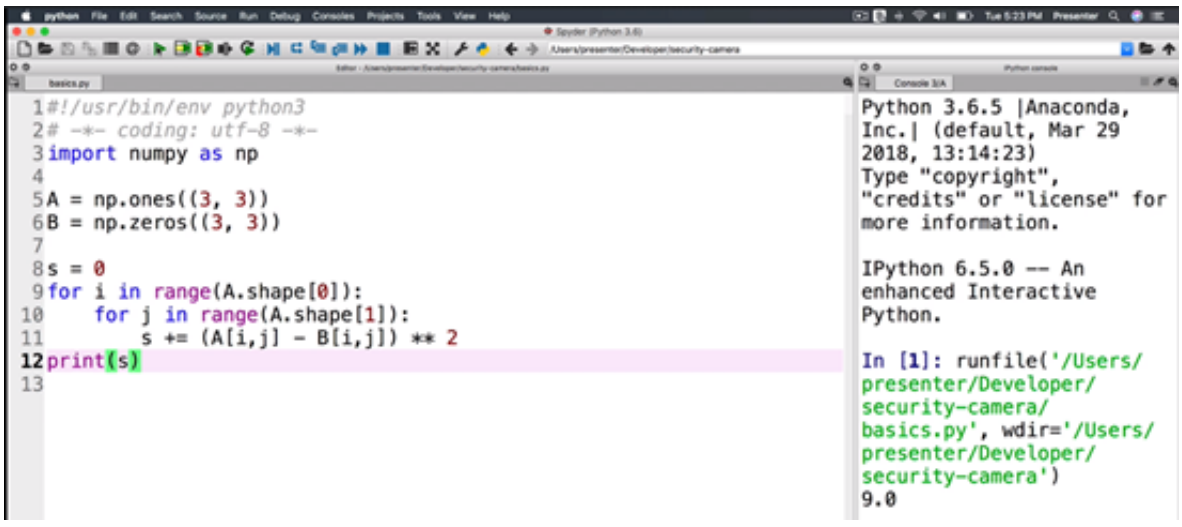
In this lesson you will be shown how to **compute the sum of squared errors using the long way and then the shorter way which uses numpy.**

See the code below for how to compute the sum of absolute errors, we will then modify it later to compute the sum of squared errors:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np

A = np.ones((3, 3))
B = np.zeros((3, 3))

s = 0
for i in range (A.shape[0]):
    for j in range(A.shape[1]):
        s += (A[i,j] - B[i,j]) ** 2
print(s)
```

Before we run this code lets go over it a little bit. All of the values of A are going to be one, and all of the values in B are going to be zero. So one minus zero is one. When the value of one is squared then you will get one, and there is nine of them, because we are doing this nine times. So the sum we expect to get again is nine.

Run the code, and the result printed out is 9.0.



We can compute the Sum of Squared Errors using the long way and then there is a shorter way we can do this as well, using numpy.

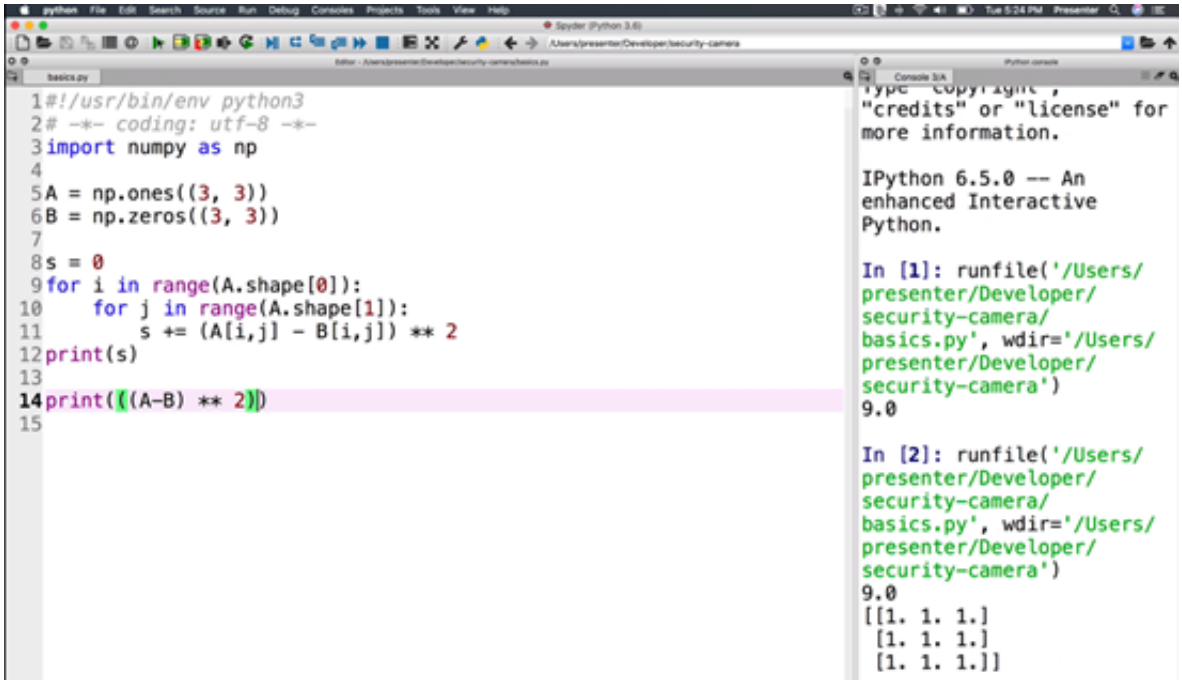
See the code below:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np

A = np.ones((3, 3))
B = np.zeros((3, 3))
```



```
s = 0
for i in range (A.shape[0]):
    for j in range(A.shape[1]):
        s += (A[i,j] - B[i,j]) ** 2
print(s)

print(((A-B) ** 2))
```

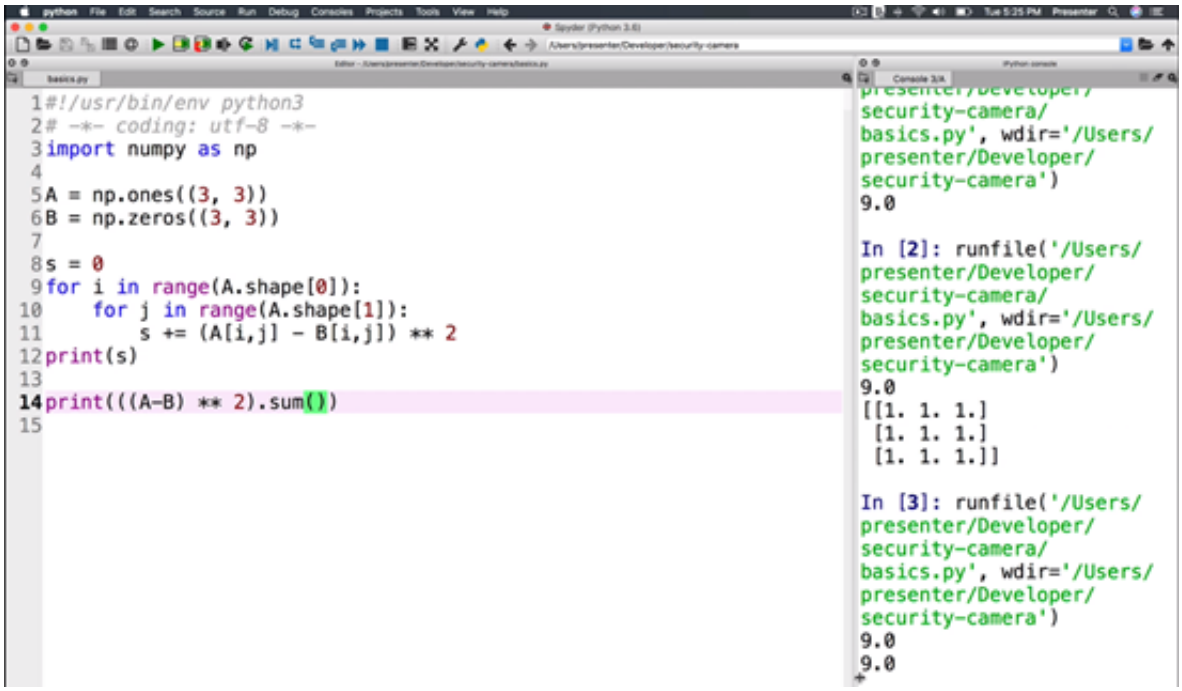
If this **code is ran, we will get all ones.**



See the code below:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np

A = np.ones((3, 3))
B = np.zeros((3, 3))

s = 0
for i in range (A.shape[0]):
    for j in range(A.shape[1]):
        s += (A[i,j] - B[i,j]) ** 2
print(s)

print(((A-B) ** 2).sum())
```

When this code is ran we get 9.0 again printed out.

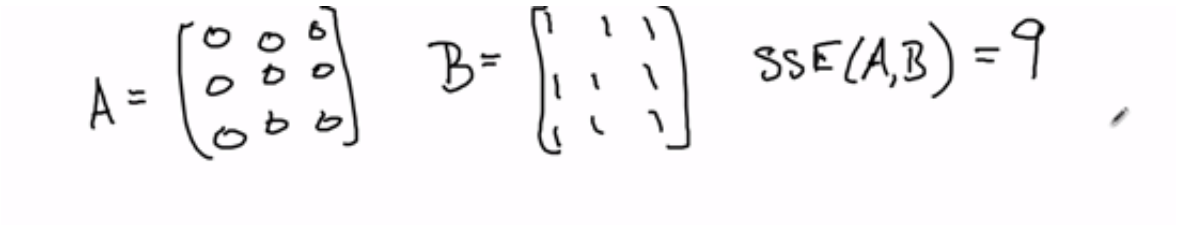


So this is another alternative way that we can compute the Sum of Squared Errors, the SSE, using numpy.

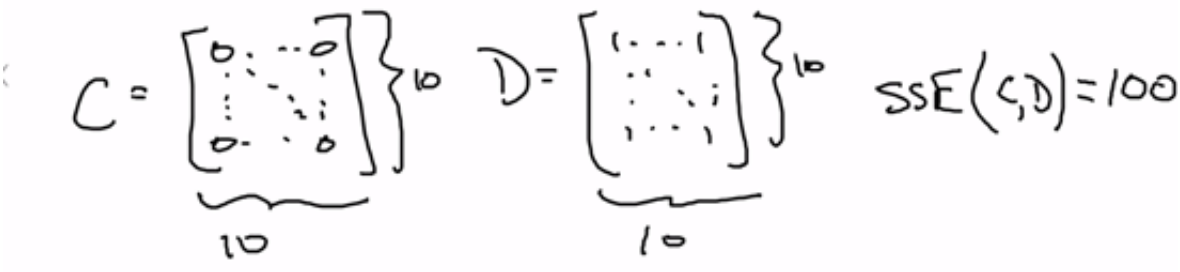
In this lesson we'll discuss about an improvement that we can make on sum squared error that will convert it into mean squared error.

You can infer from this title of mean squared error what kind of changes that will be made.

First, lets illustrate why we need this change, so Sum of Squared Error is good but there's one big problem with it, here is an example:

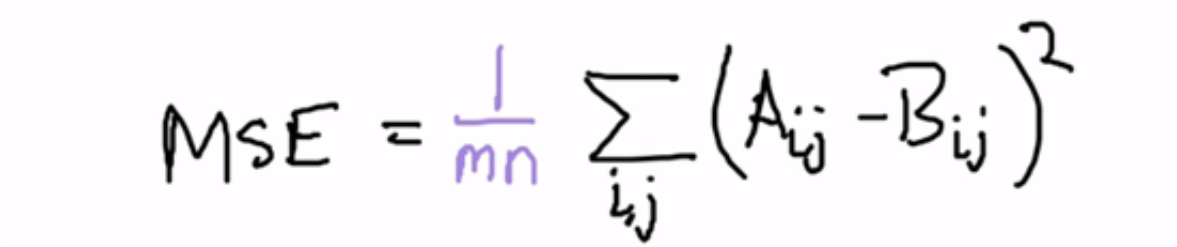


And if we made this matrix larger, where matrix C had zeros and was 10 rows and 10 columns, then matrix D had all ones and was 10 rows and 10 columns:



So all four of these matrices above have the same content, the only difference is that matrix C and D are just larger matrices than A and B. So now the Sum of Squared Error goes through the roof. This isn't a good representation of error. It shouldn't matter if two images are the same, and if they were shrunk or grown, they should remain the same. **This is why we need Mean Squared Error.**

Mean Squared Error or MSE is the same of Sum Squared Error, except there is one thing we need and what we need to do is **add the M and the N, where the M is the number of rows and N is the number of columns:**

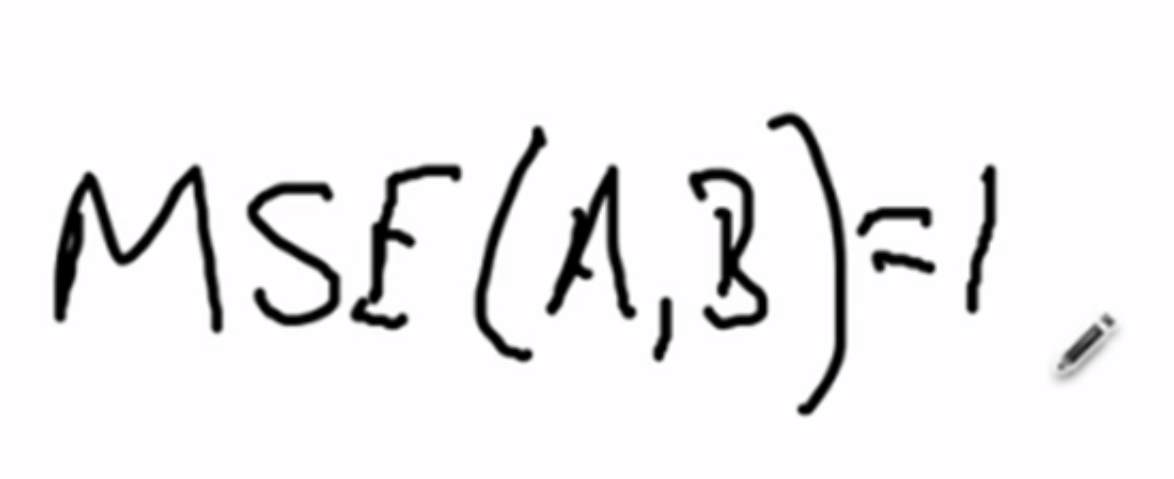


So what we are doing here is averaging the error.

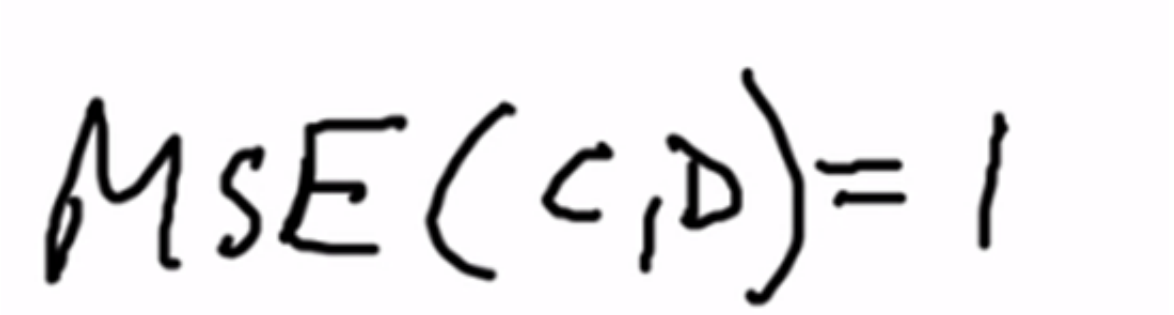
The term in purple in the screenshot above gets you the total number of pixels in the image A and in the image B.

So what we are doing here is we are basically just averaging the total error divided by how many pixels this error occurs over ans so we get the average of the sum squared error. This is called Mean Squared Error because we are taking the mean, which is just another term for saying average. In some cases this is better then Sum of Squared Error.

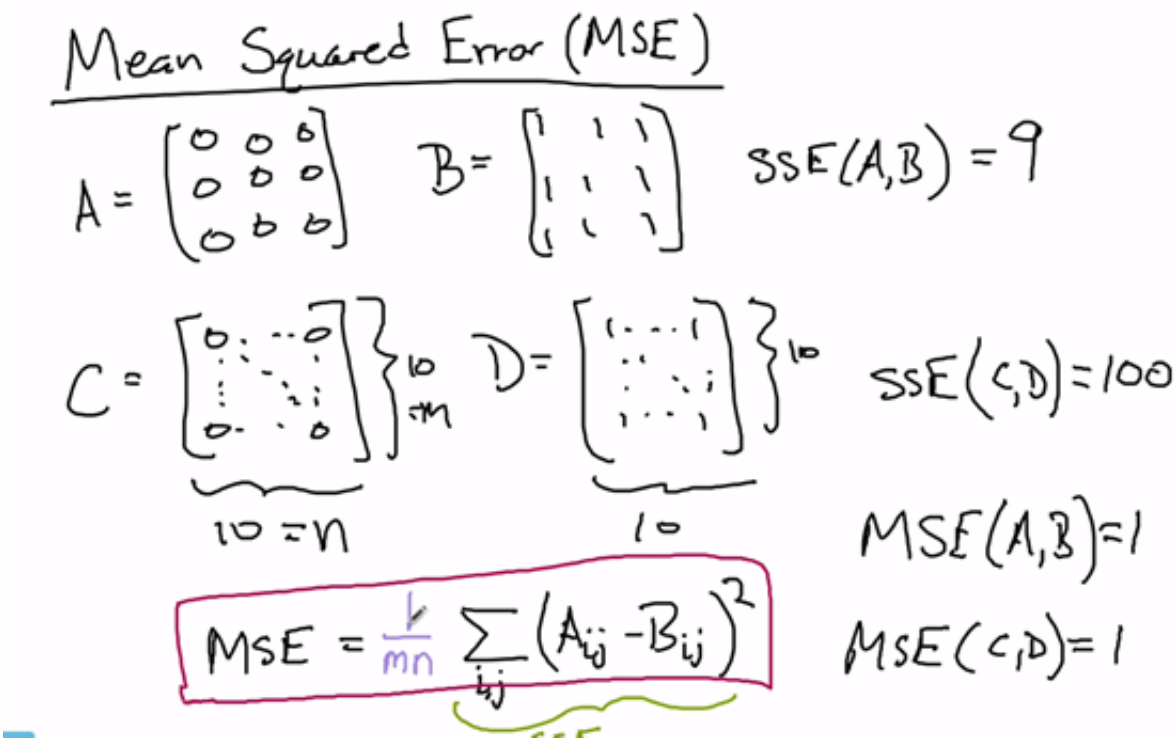
Let's go back to Matrix A and B and let's compute the mean squared error for these matrices:



Let's go back to Matrix C and D and let's compute the mean squared error for these matrices:



In almost all cases we always prefer Mean Squared Error over Sum of Squared Error.

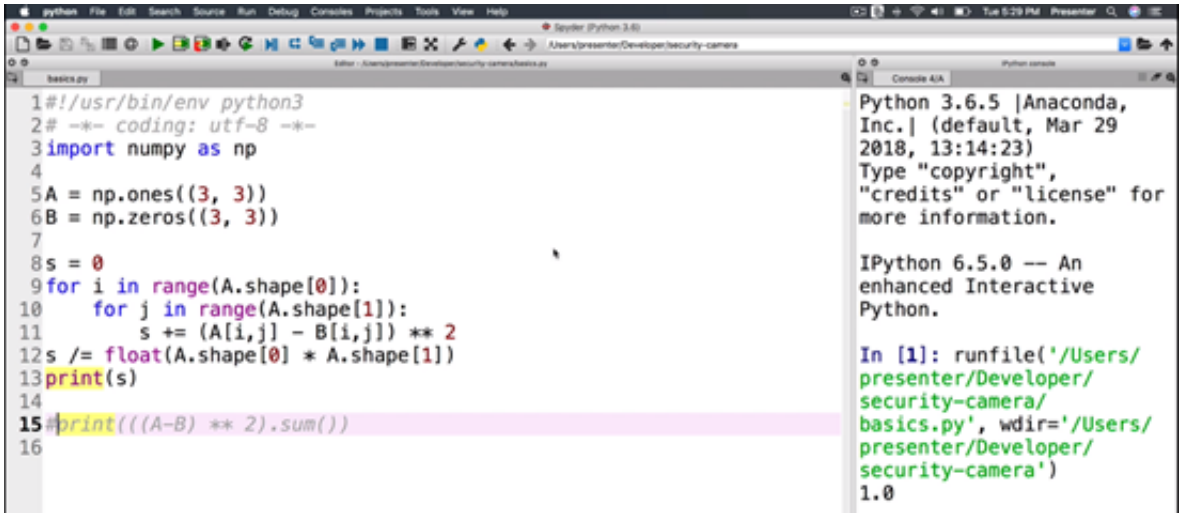


In this lesson you will learn how to compute the mean squared error using numpy and modifying it from the sum of squared error is also going to be quite simple to do. The sum of squared error will take the sum across all of these, or take the difference between the matrices and squares them and takes the sum. **Instead of taking the sum, also we wanna take the mean. In order to convert this from an SSE to an MSE, we simply divide by the total number of elements we have.**

See the code below:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np

A = np.ones((3, 3))
B = np.zeros((3, 3))

s = 0
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        s += (A[i, j] - B[i, j]) ** 2
s /= float(A.shape[0] * A.shape[1])

print(s)

#print(((A-B) **2).sum())
```

When this code is ran you will get 1.0 printed out.



This is correctly computing the MSE, and this is the longer way around this.

There is a shorter way around this using numpy, which equates to just changing a single function call.

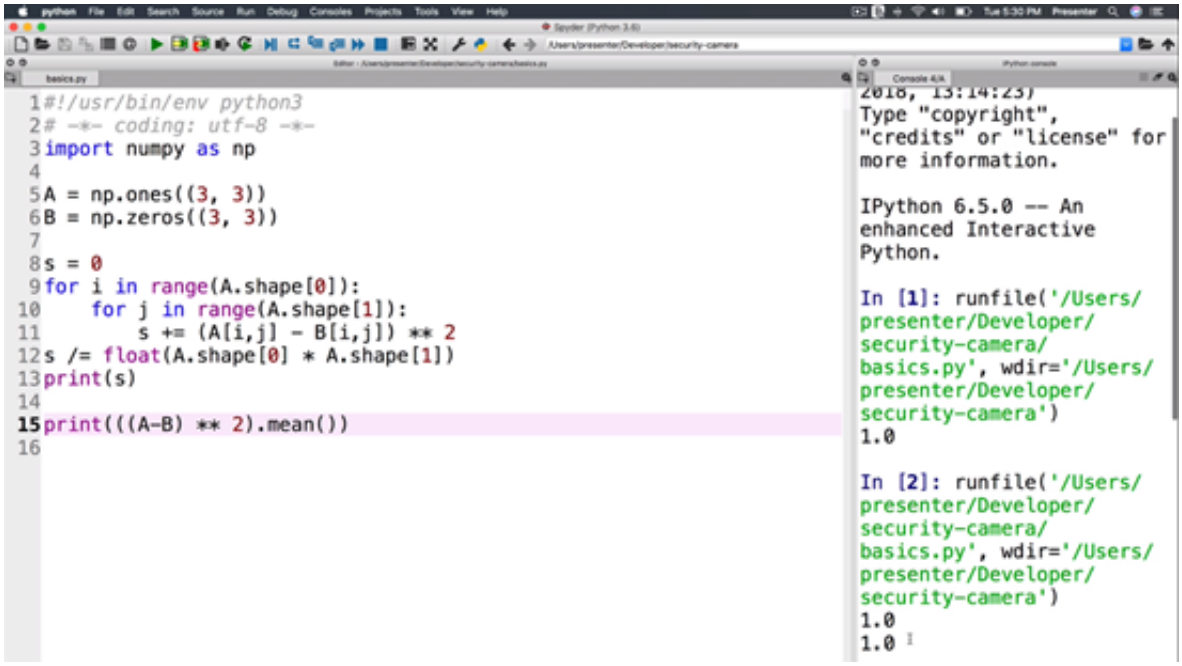
See the code below:

```
#!/usr/bin/env python3
```



```
# -*- coding: utf-8 -*-

import numpy as np

A = np.ones((3, 3))
B = np.zeros((3, 3))

s = 0
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        s += (A[i, j] - B[i, j]) ** 2
s /= float(A.shape[0] * A.shape[1])

print(s)

print(((A-B) **2).mean())
```

When this code is ran we will get the result 1.0 printed out.



In this lesson we will discuss a different approach to image-similarity called **structural similarity(SSIM).** A Mean Squared Error is a really good measure of error difference, but the issue with mean squared error is that it looks at each pixel individually and independently. This is different to how humans perceive images because we don't look at two images and look at all the pixels of the images and compare them.

We look at the images in a more holistic sense. This is what structural similarity is trying to capture.

So instead of treating it pixel-independently structural similarity actually looks at groups of pixels to try to better determine if two images are different or not.

MSE and SSE look at pixels individually. **SSIM looks at groups of pixels, and this is better than looking at pixels individually because then all of those small changes in noise and variation don't tend to affect groups of pixels than they do with individual pixels.**

As it turns out that some structural similarity is actually a product of three other kinds of measures.

$$SSIM(A,B) = l(A,B) \cdot c(A,B) \cdot s(A,B)$$

The L stands for luminance, C is for contrast, and S is for structure.

$$SSIM(A,B) = l(A,B) \cdot c(A,B) \cdot s(A,B)$$

luminance contrast structure

When you look at groups of pixels, then you get a better representation of the image and a better representation subsequently of the difference between images when you look at groups instead of just one individual pixel.

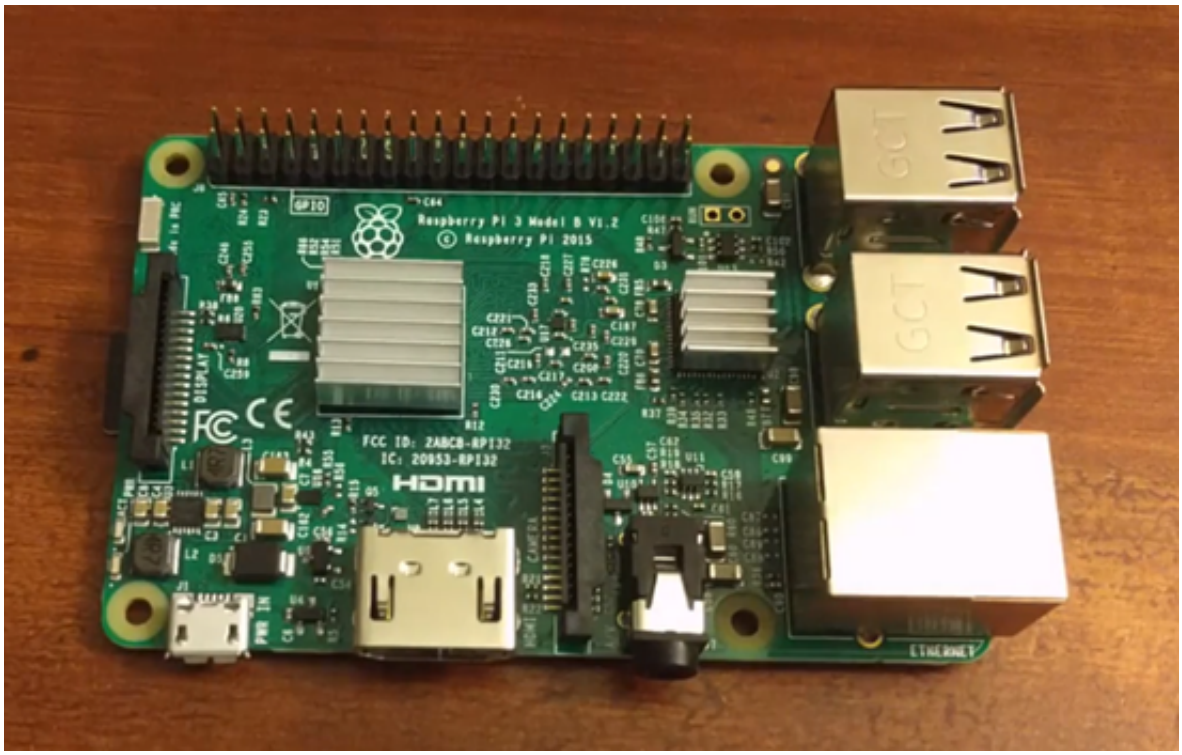
The good thing about structural similarity and its much different than MSE, or SSE, is that it actually has a lower and upper bound.

$$-1 \leq SSIM \leq 1$$

perfect imperfect perfect

In this lesson we will be looking at the **Raspberry Pi.**

Here it is:

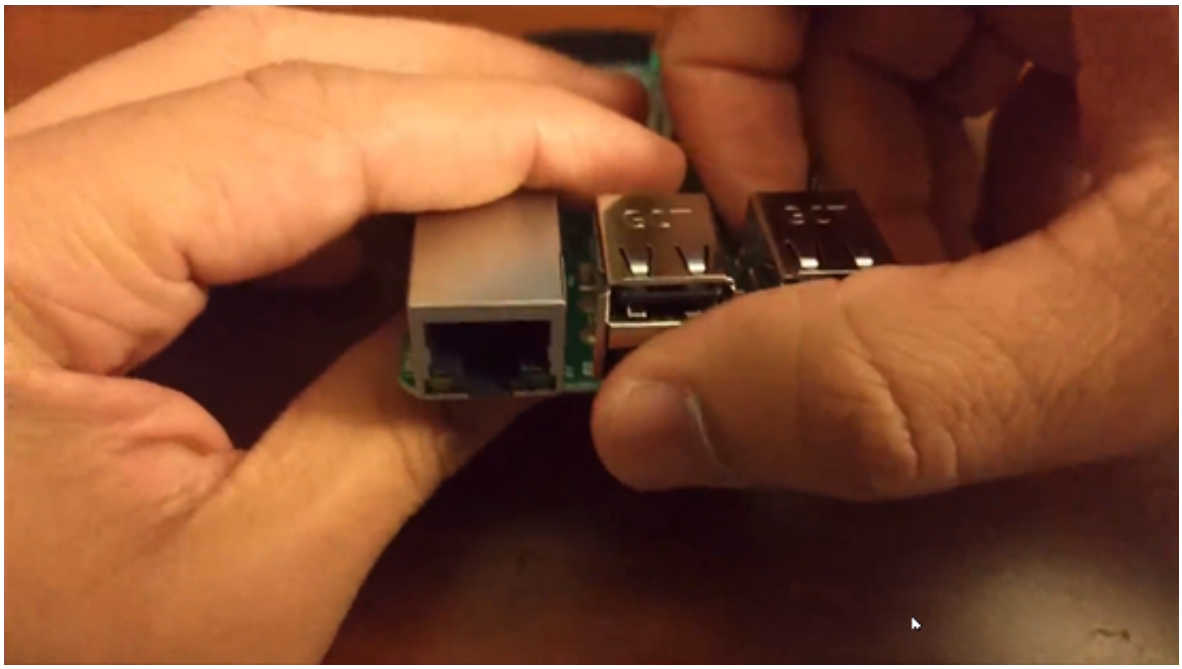


It is basically a micro computer, and its a full computer. You can write documents on it, manage spreadsheets, play games, stream videos, and do pretty much anything else you can do with a regular computer. Only its much smaller.

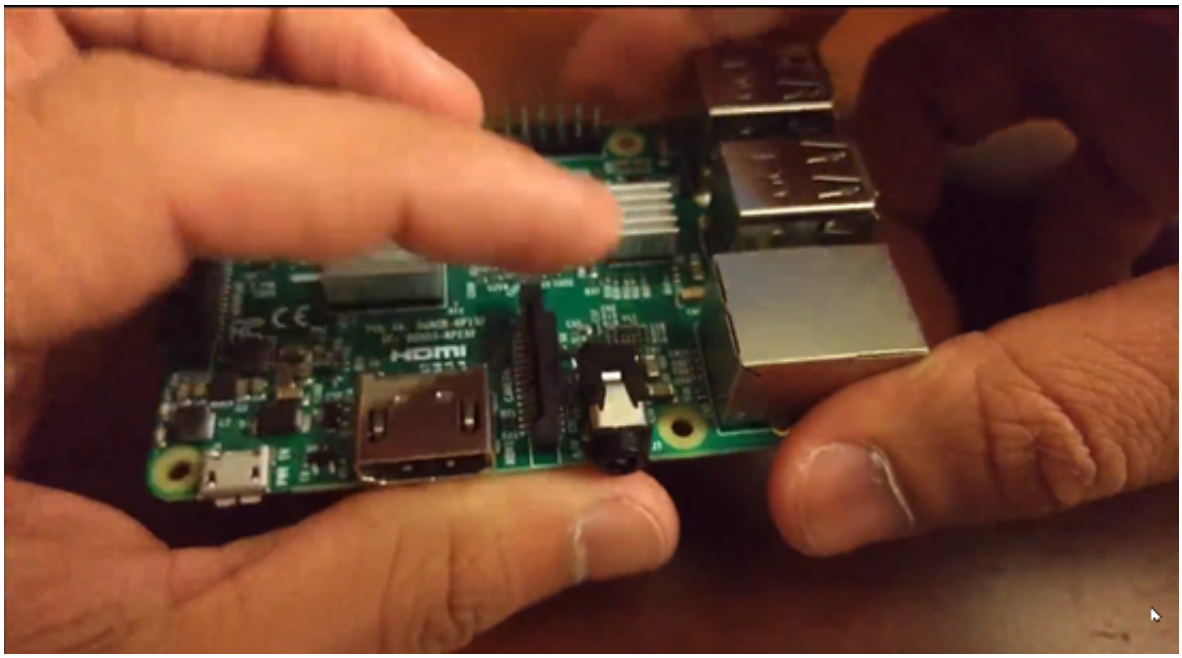
This a very small computer, its about credit card size.

You have **access to hardware components through these general-purpose input/output or GPI.**

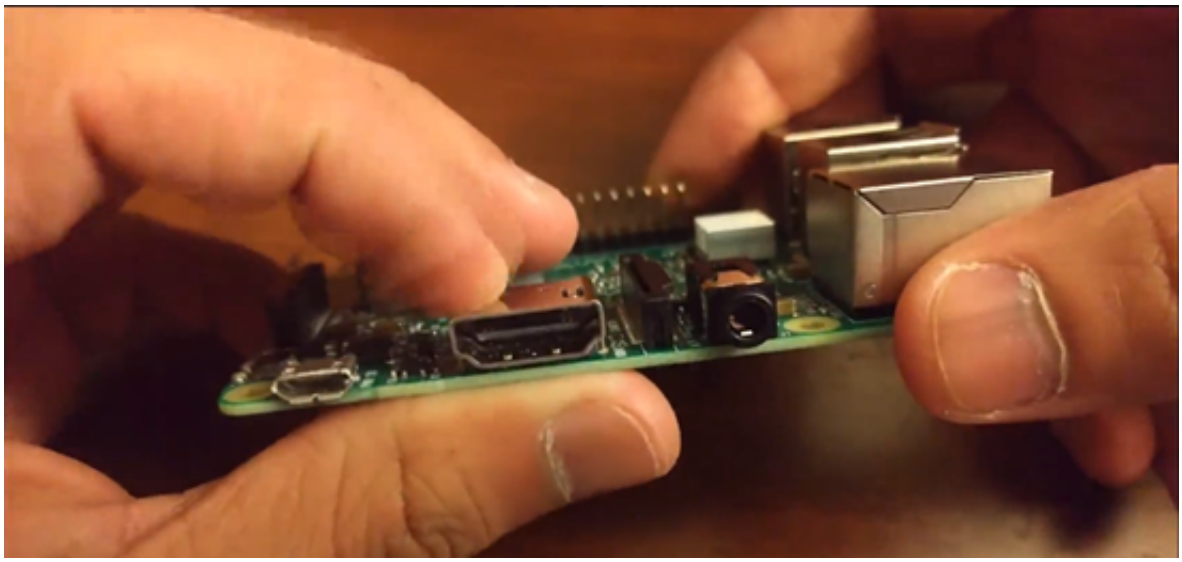
In addition to those pins we get **four USB ports, we get the Ethernet that you see there:**



When flipped on its side you will see there is **audio out:**



There is an **HDMI out.**



The micro usb is the actual power supply system:

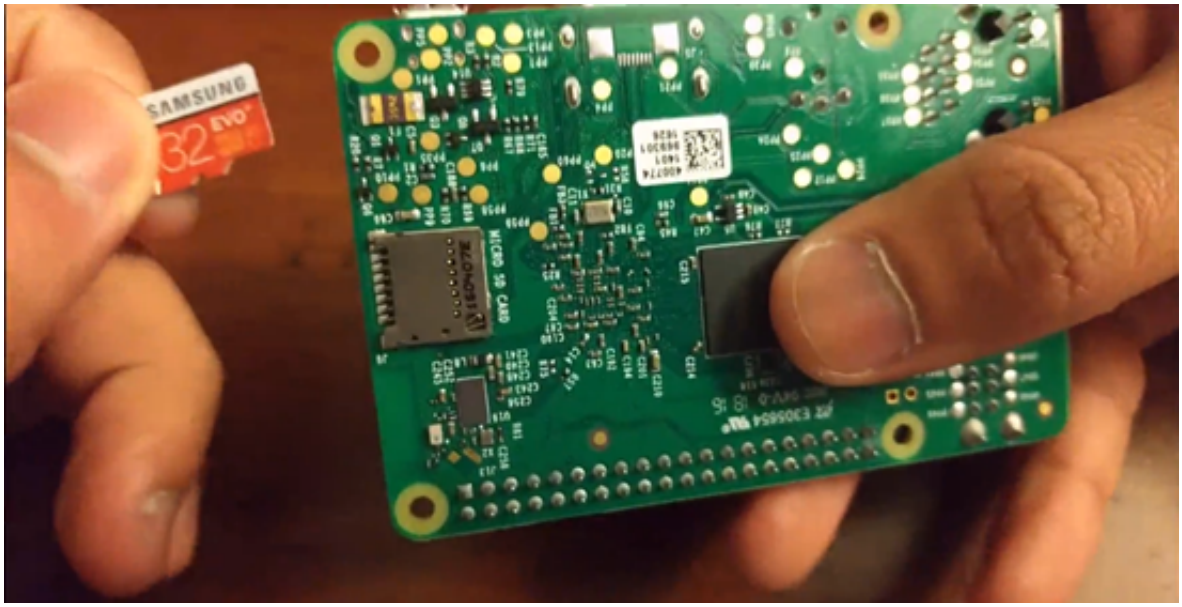
There is a **ribbon for display on the side:**

Then there is a **ribbon for a camera, here:**

The main processor of the Raspberry Pi, is actually kind of hidden under the heat sync here:

The heat sync just basically dissipates excess heat that generates from running the Raspberry Pi. It dissipates that heat so that we don't actually end up damaging the PI. The newer Pi3, which is the one here in this lesson have built-in wifi and bluetooth so that we don't have to actually buy any adapters.

There is a slot for micro SD, and you will need one of these cards as well:



There are actual instructions on how to upload the Raspberry Pi's operating system on their official website.

Here is the link to the website: https://www.raspberrypi.org/help/faqs/

Here is the direct link: Raspberry PI

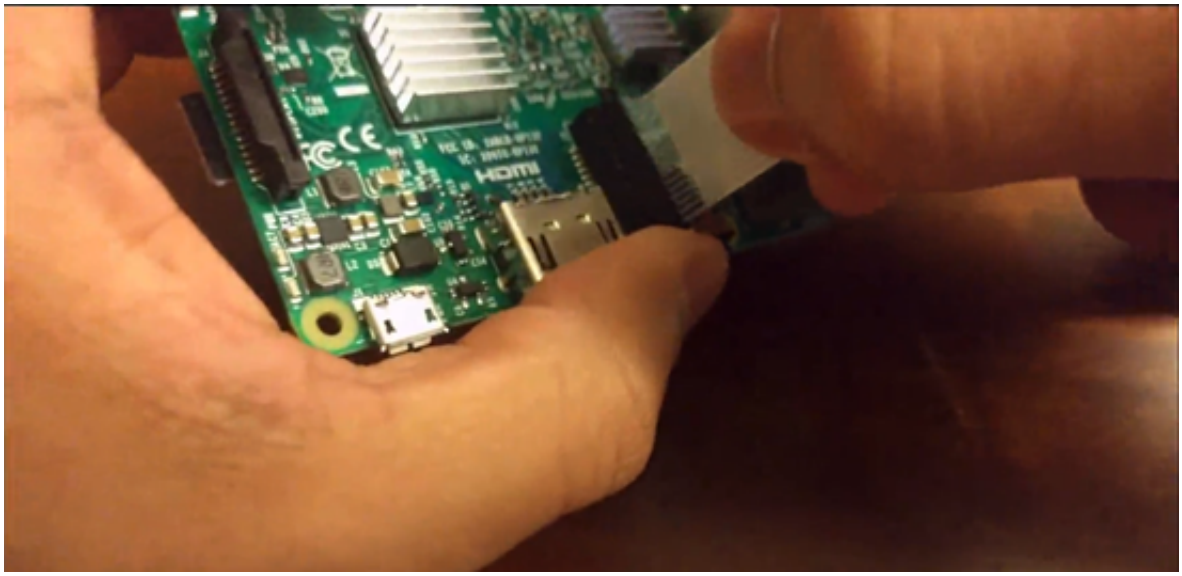
Once you have all this stuff we can plug in the keyboard and mouse and connect it to your monitor, it will automatically turn on and run when you plug in the power. A reminder the power is a micro usb port.

Here is the camera module to the Raspberry Pi:

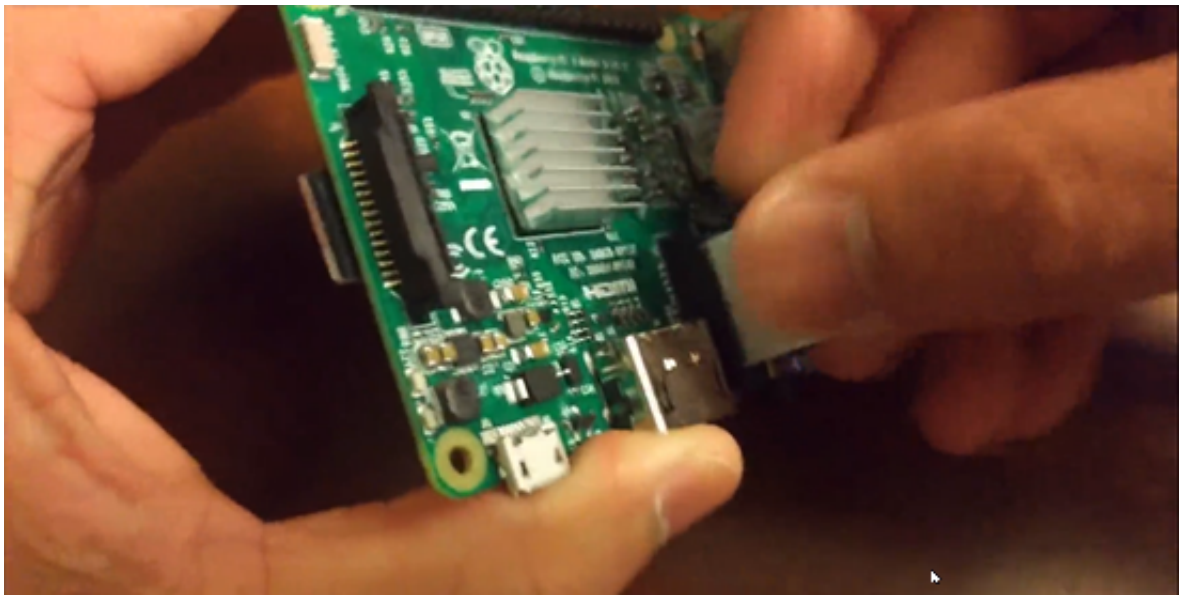
Instead of using usb, it actually uses a ribbon to transfer the captured image to the PI.

To get this actually working you can open up that port here:

You open it up, and put the ribbon in:



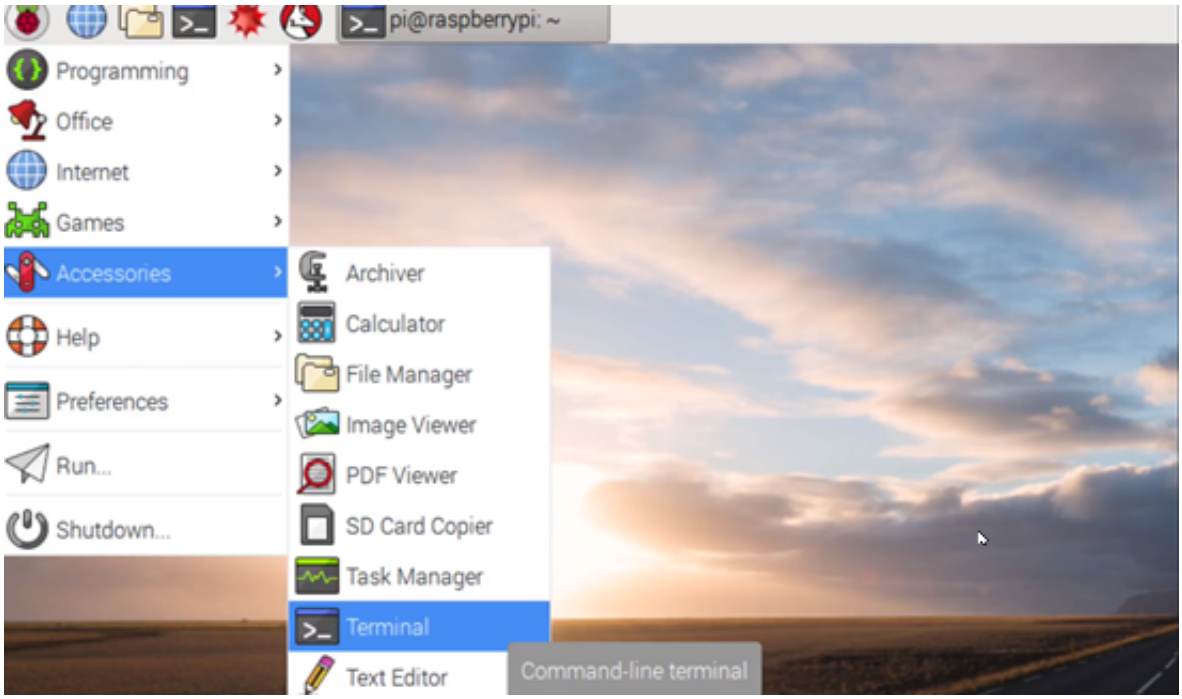
See the pins on the left, make sure the shiny part of the ribbon actually lines up, it’s pointing to the left, so that it actually makes connections with that.



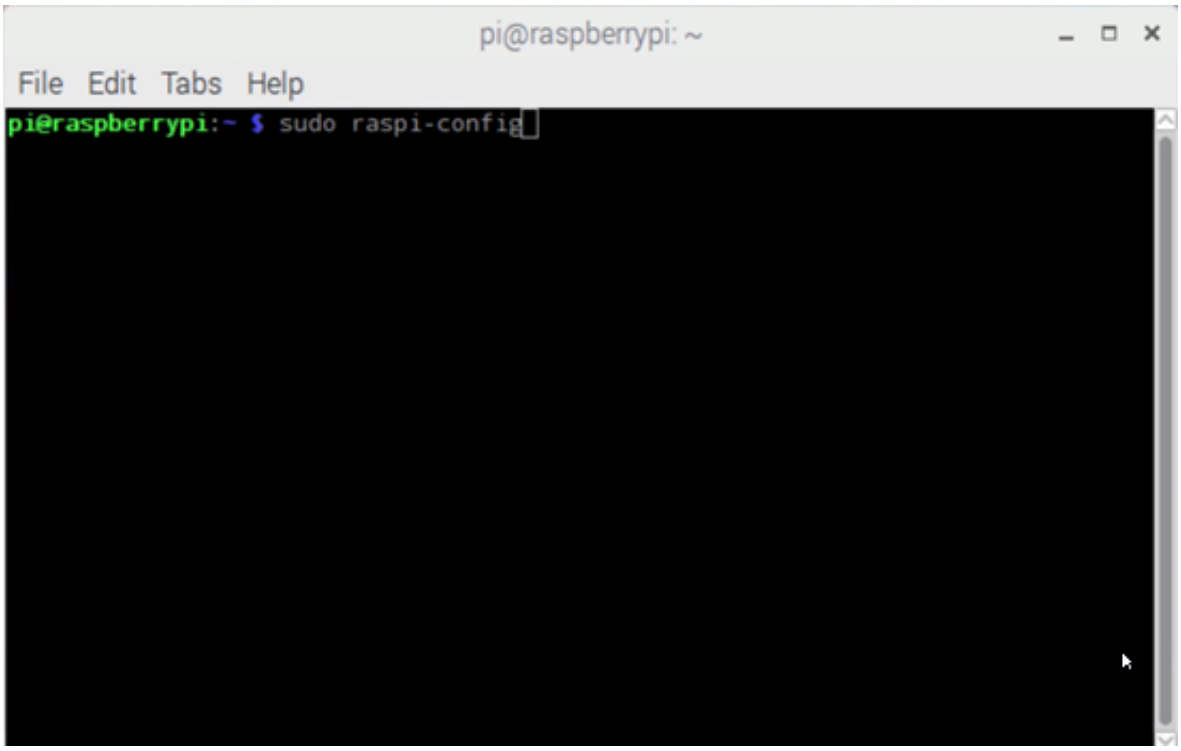
When the camera port is closed we can actually lock it in.

That is it really for hardware regarding the Raspberry Pi and the camera module.

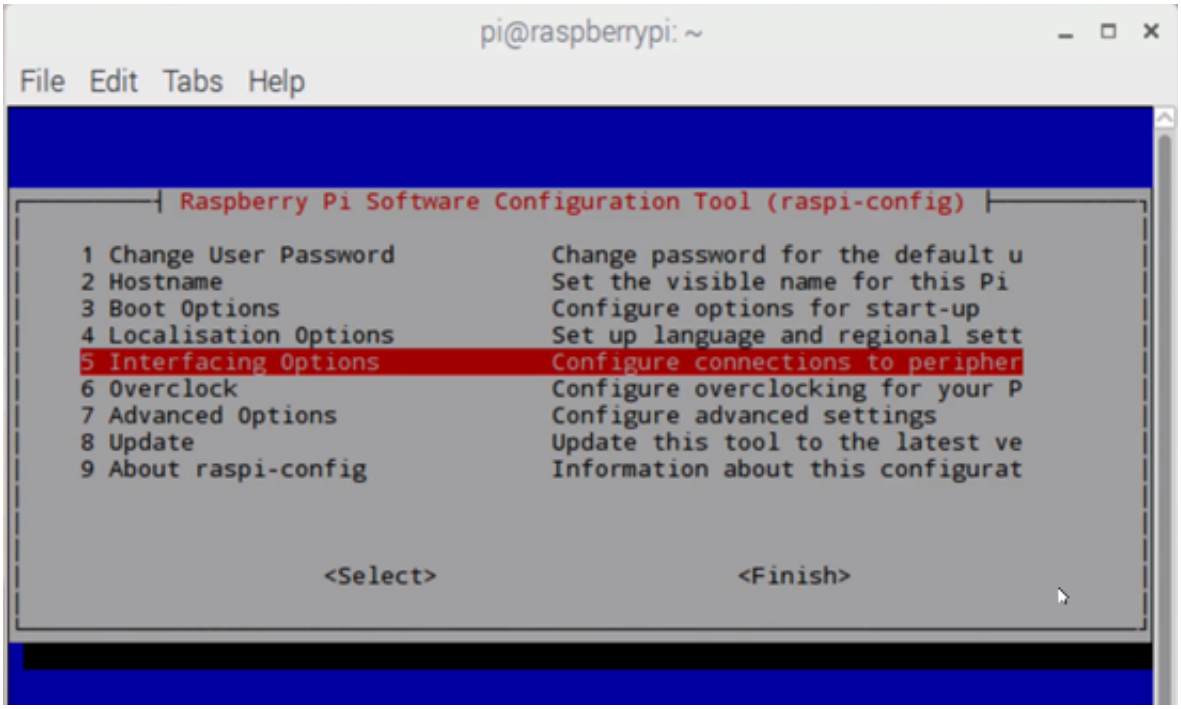
There are some things we need to software side though. When we are dealing with the software side, once you have the Raspberry Pi displaying on your monitor and the display:



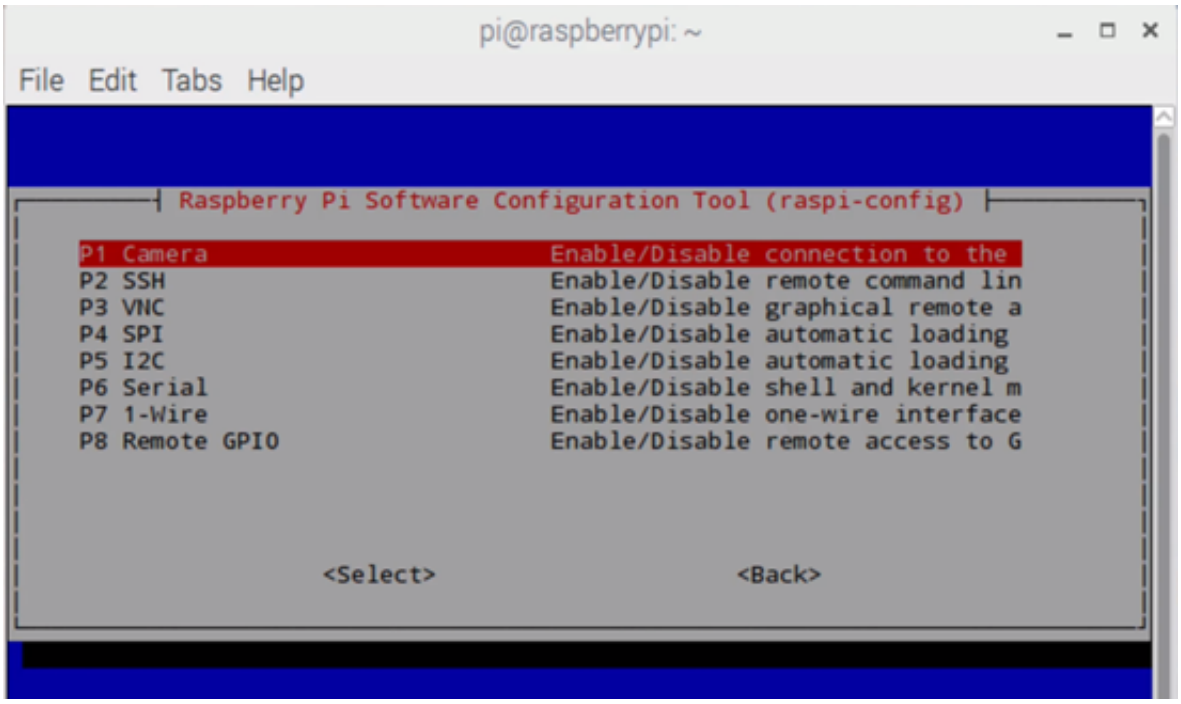
Find a terminal window and type in **“SUDO raspi-config”**



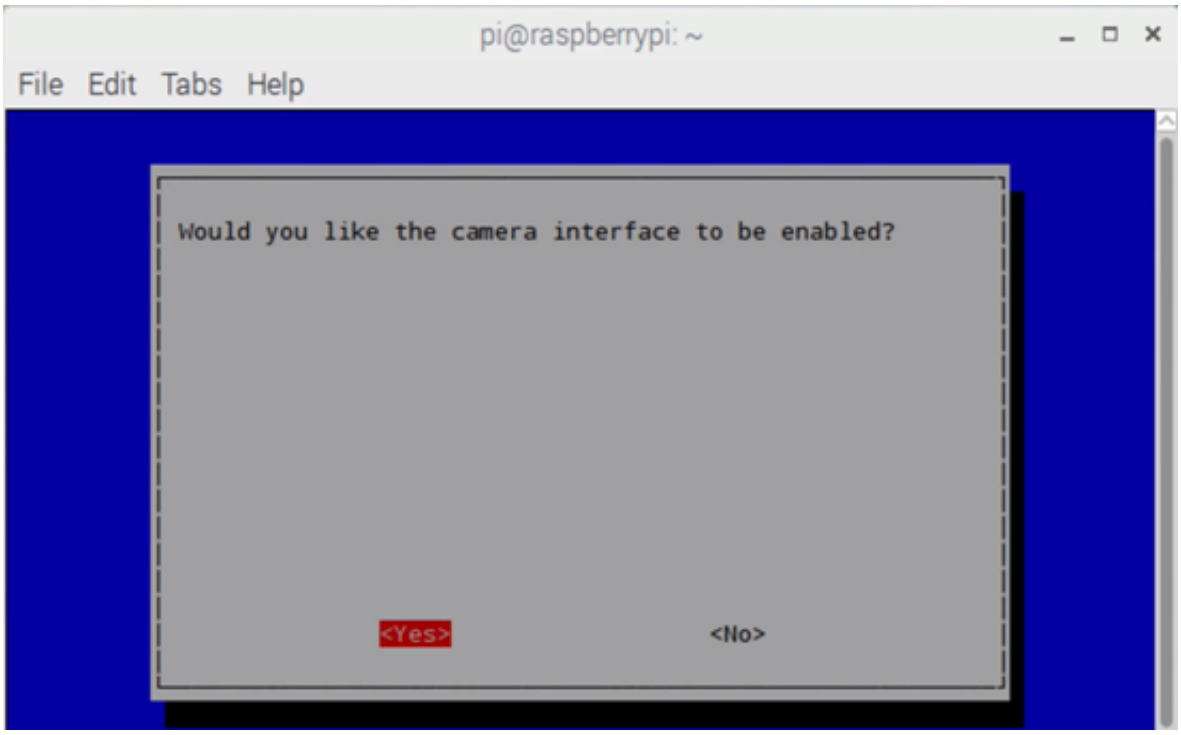
You will probably have to enter your password for the PI,



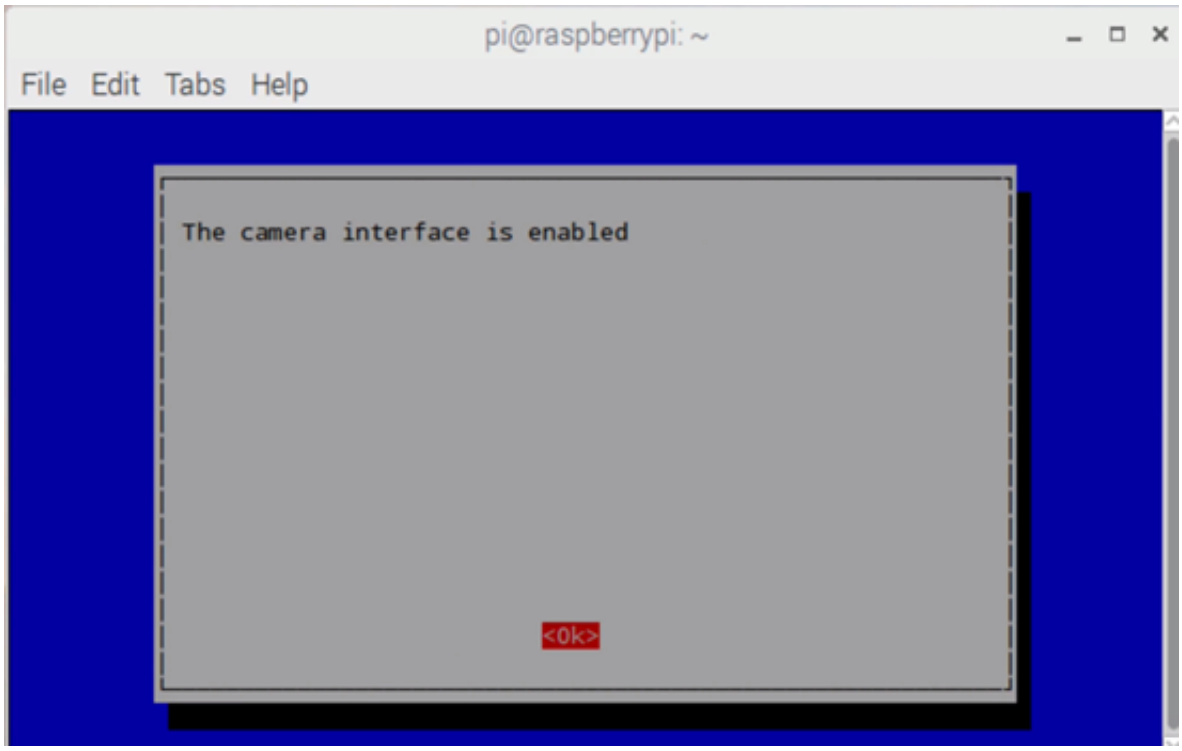
And if you notice, if you look at this image here, **one of the entries is Enable Camera:**



You can use the arrow keys to select that option, and then hit the Enter key and that will enable the camera:



Then you can reboot the Raspberry Pi actually after you select the OK prompt:



After the PI is rebooted everything should be good to go!

You do not need a Raspberry Pi for this project, you don't need to go out and buy one, all of the stuff we will be working on in the upcoming lessons will actually function just fine on like a webcam or something.

This is a way to introduce you to a Raspberry Pi.

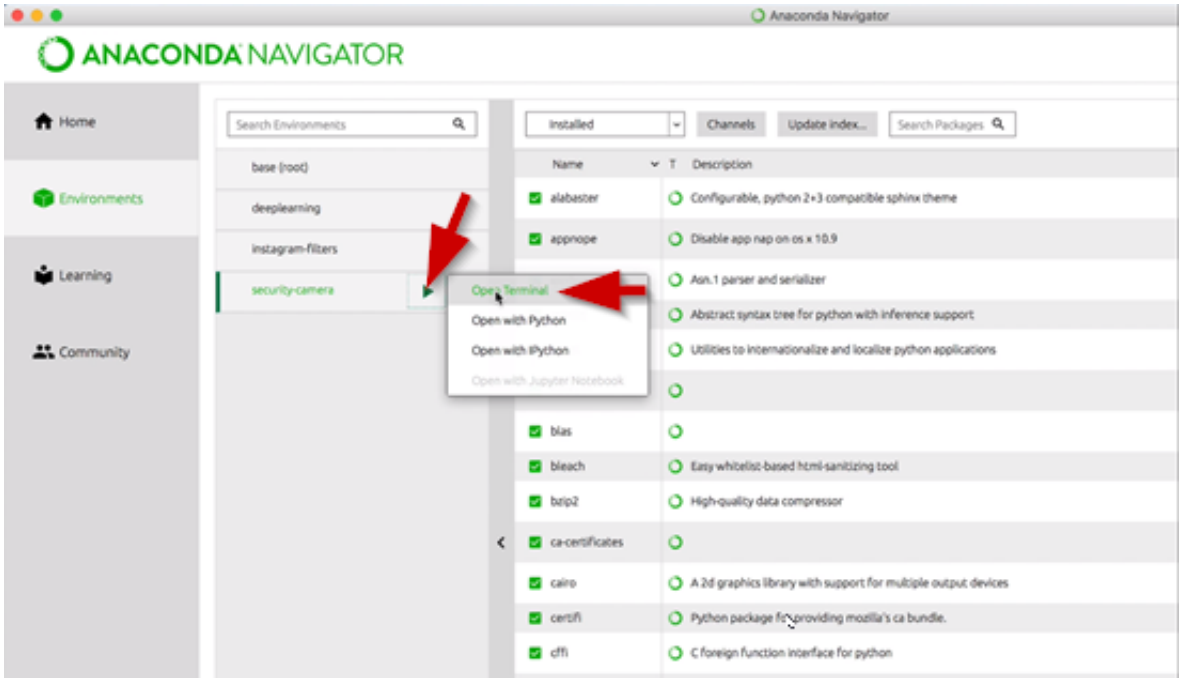
In this lesson we will be setting up our developer environment and installing the Twilio package, along with moving over some project files from the downloaded zip file with all the source code so that we can get started building our security camera.

So the first thing we need to do is setup our security camera environment with Anaconda. This has already been covered in the first lesson of this course, **there is a .zip file you can download from the course home page that contains the environment.yml file, you can unzip this and then import it in the Anaconda Environment.**

The next thing we need to do is install Twilio. There is no way to do this directly through the Anaconda Navigator so we will have to use a Python installer called PIP. In order to install Twilio it's going to be just a single line that will need to be ran in the terminal.

S**o begin by opening Anaconda, go to environments, and make sure you have the correct environment set up. The one you that you just imported should be called security camera.**

Click on the green arrow and click Open Terminal.



This will pull up the terminal:

The correct environment is loaded. You can check this by looking at the name that comes before this **“bash-3.2$”** This should say security camera in parentheses to the left. So this creates a terminal and it loads all of our environment dependencies into it as well.

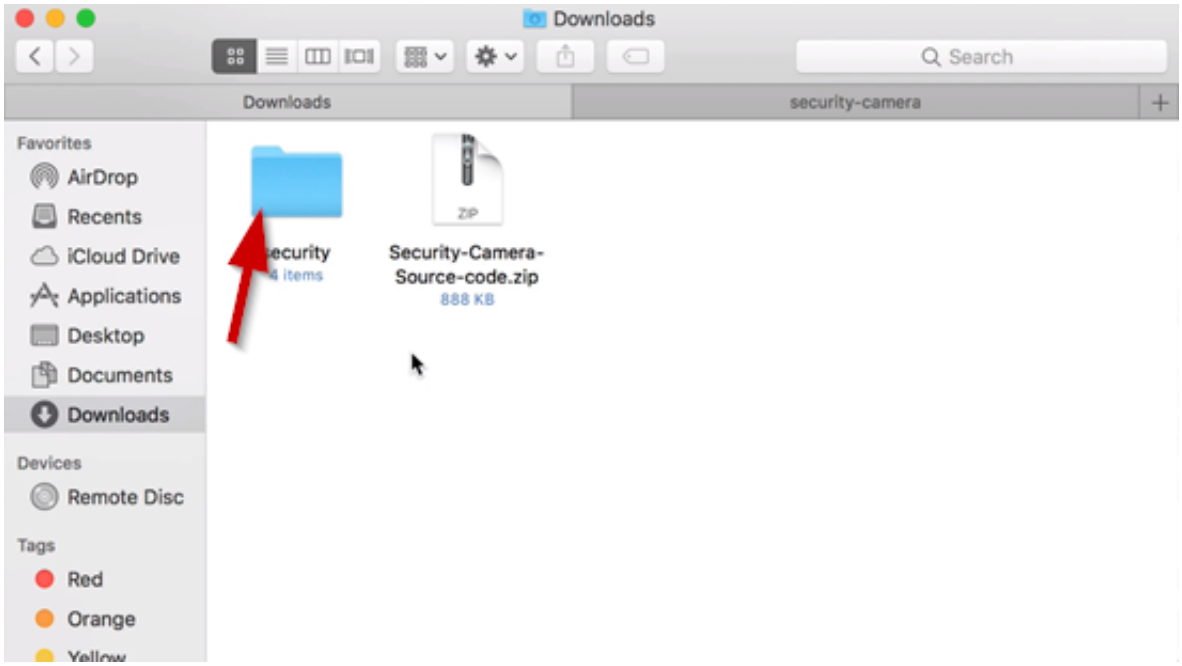
So to install Twilio we just have to run one command. It’s called **“pip install twilio”**

Pip is a package manager tool for Python, it comes built-in with Anaconda, and it has a list of repositories that we can download Python packages from, and then this is going to be loaded into our security camera environment. So when we do pip install twilio, it’ll download the twilio SDK from the correct repository and then install it in just this environment. So **hit the enter key** and you will see the installation begin:

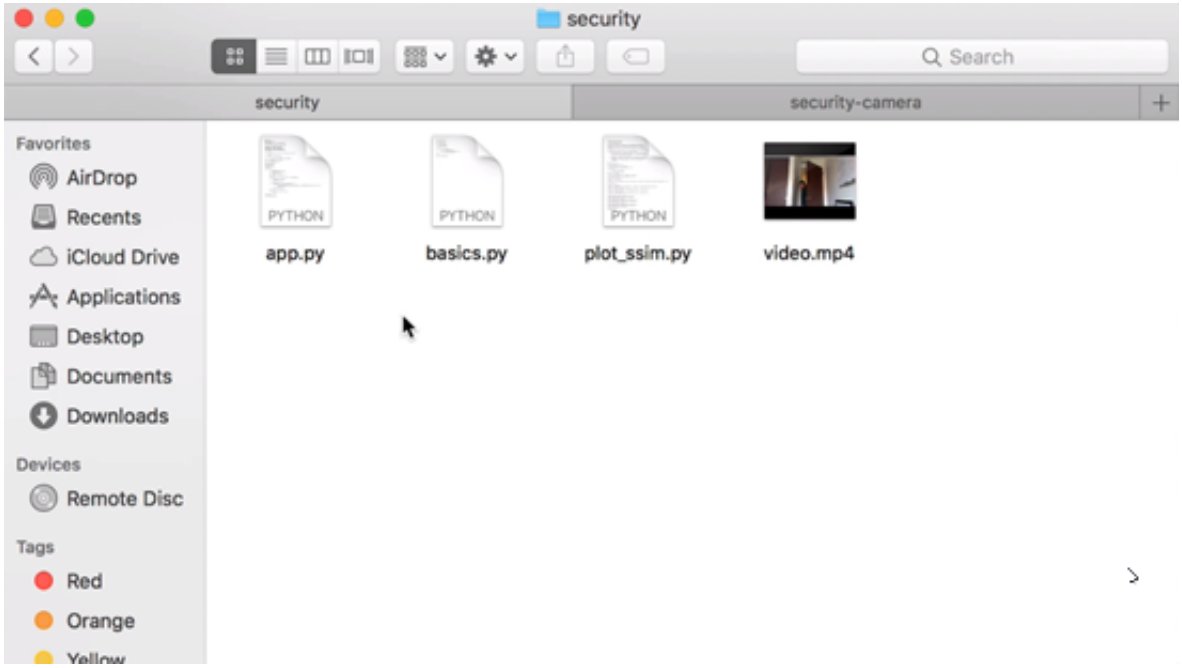

```
presenter — a.tool — pip install twilio — 80×24
thon3.6/site-packages (6.16.3)
Requirement already satisfied: pysocks; python_version >= "3.0" in ./anaconda3/e
nvs/security-camera/lib/python3.6/site-packages (from twilio) (1.6.8)
Requirement already satisfied: pytz in ./anaconda3/envs/security-camera/lib/pyth
on3.6/site-packages (from twilio) (2018.5)
Requirement already satisfied: PyJWT>=1.4.2 in ./anaconda3/envs/security-camera/
lib/python3.6/site-packages (from twilio) (1.6.4)
Requirement already satisfied: requests>=2.0.0; python_version >= "3.0" in ./ana
conda3/envs/security-camera/lib/python3.6/site-packages (from twilio) (2.19.1)
Requirement already satisfied: six in ./anaconda3/envs/security-camera/lib/pytho
n3.6/site-packages (from twilio) (1.11.0)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in ./anaconda3/envs/securit
y-camera/lib/python3.6/site-packages (from requests>=2.0.0; python_version >= "3
.0"->twilio) (3.0.4)
Requirement already satisfied: urllib3<1.24,>=1.21.1 in ./anaconda3/envs/securit
y-camera/lib/python3.6/site-packages (from requests>=2.0.0; python_version >= "3
.0"->twilio) (1.23)
Requirement already satisfied: idna<2.8,>=2.5 in ./anaconda3/envs/security-camer
a/lib/python3.6/site-packages (from requests>=2.0.0; python_version >= "3.0"->tw
ilio) (2.7)
Requirement already satisfied: certifi>=2017.4.17 in ./anaconda3/envs/security-c
amera/lib/python3.6/site-packages (from requests>=2.0.0; python_version >= "3.0"
->twilio) (2018.8.24)
```

Once twilio is installed, we have to **copy over some files from the zip file that has all of the source code into our project directory, and this is because for our security camera, although we're going to be using a webcam for if you're trying to do some kind of development on a computer that doesn't have a webcam or to kind of supplant the webcam, there is an MP4 file and we're going to load, we are going to be using for our development purposes.** We are just going to be prototyping on the video, and then I am going to explain how we can move on to use an actual webcam.

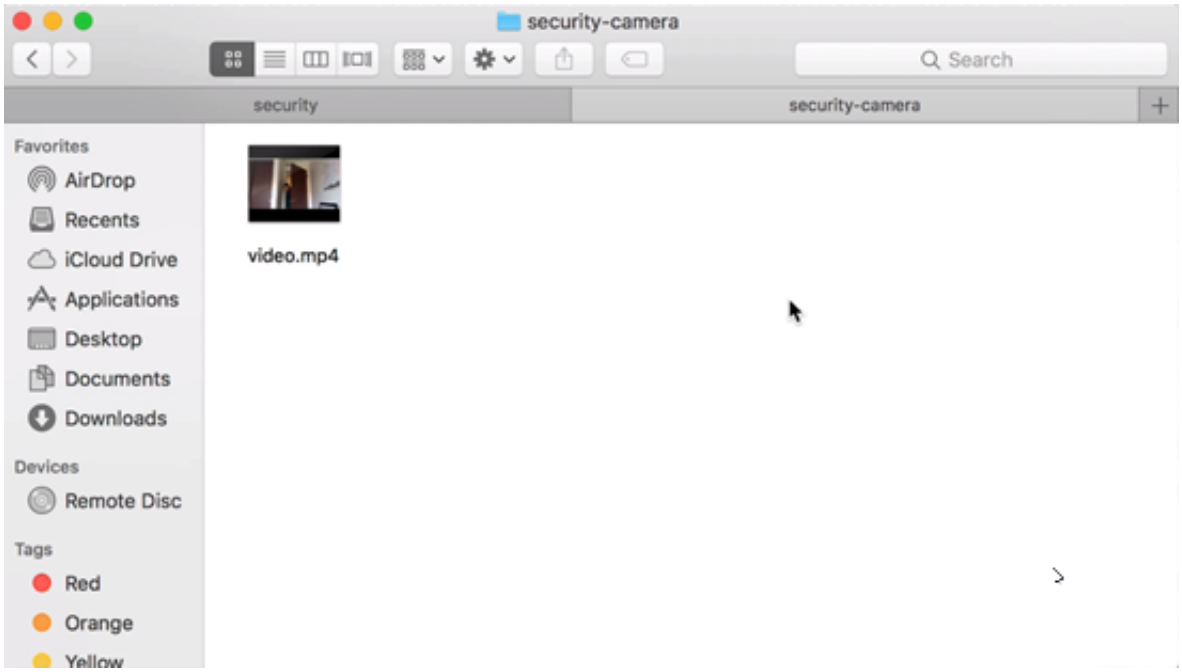
First, let's move the video over into our project directory because we are going to mostly be using that. If you **navigate to where you have your downloads, there should be a source code zip file that contains all of the source code in the project file. So unzip the Security-Camera-Source-Code file.**



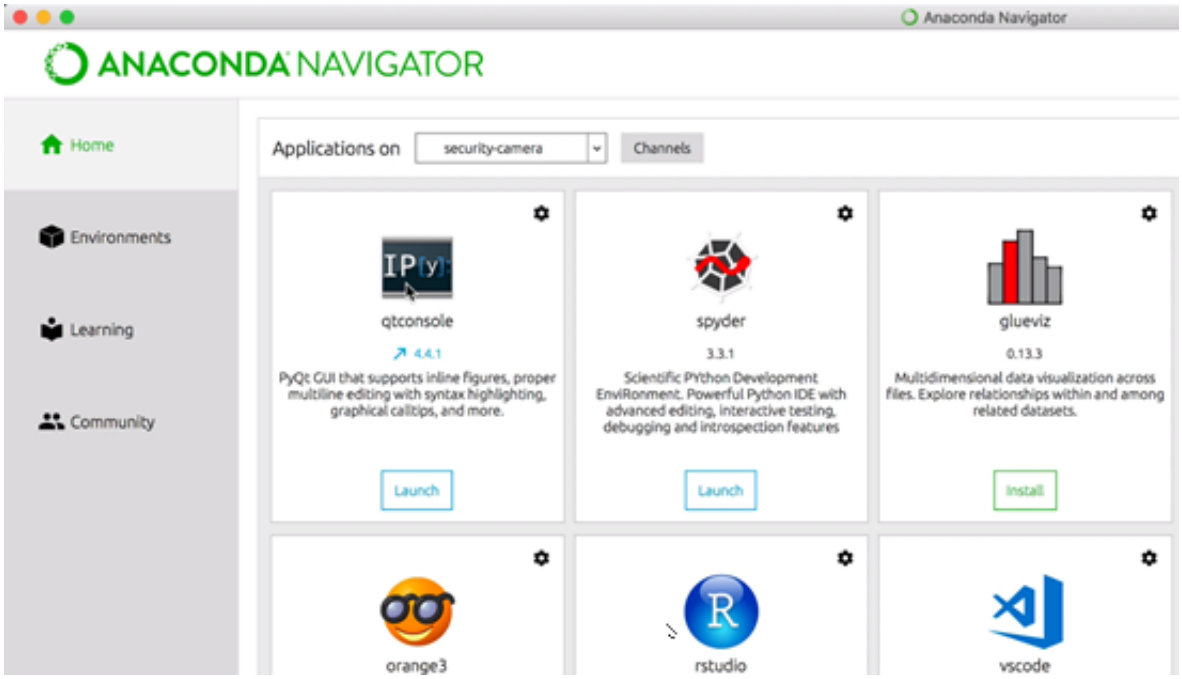
Inside the Security folder there will be four files:



What we are concerned with right now though is the **video.mp4 file. We want to copy that file, and paste the video.mp4 file into the security-camera project directory:**

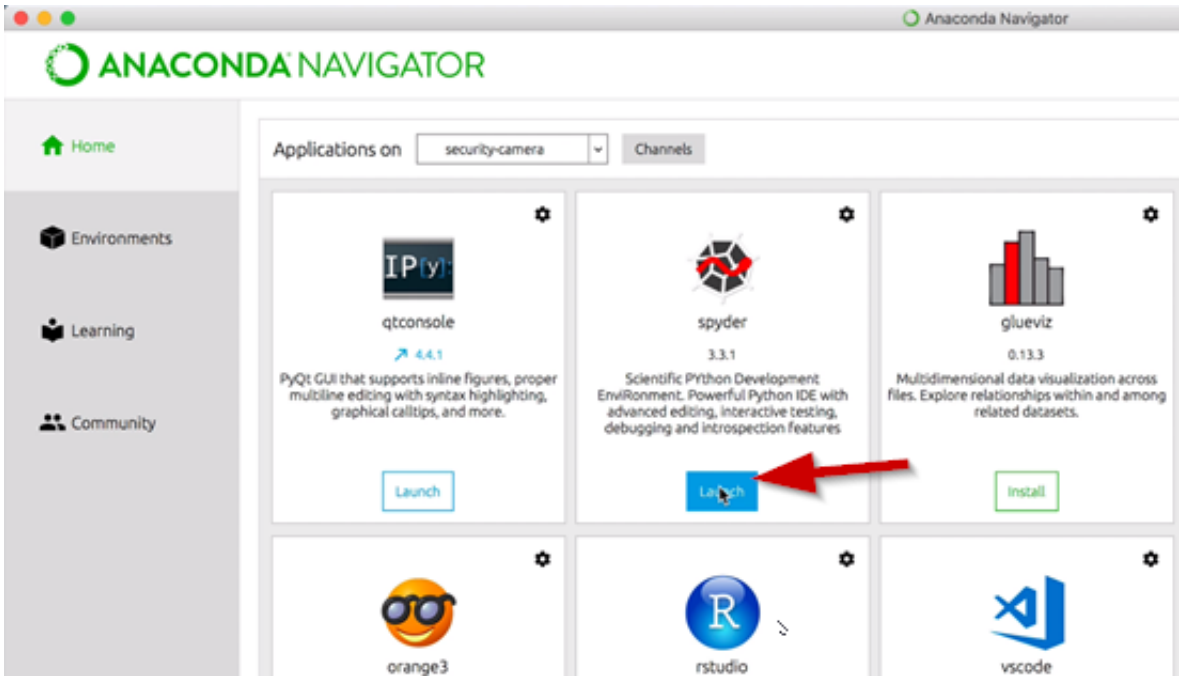


Now that we have the prerequisites installed let's get started building our app. So we are going to open up the Anaconda Navigator:

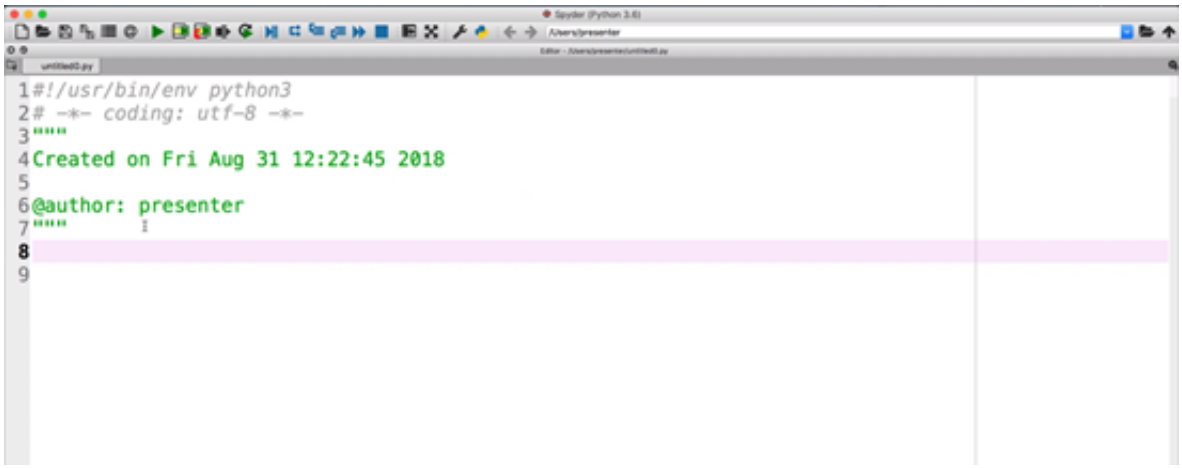


Make sure you are in the right environment. Which should be **"security-camera."**

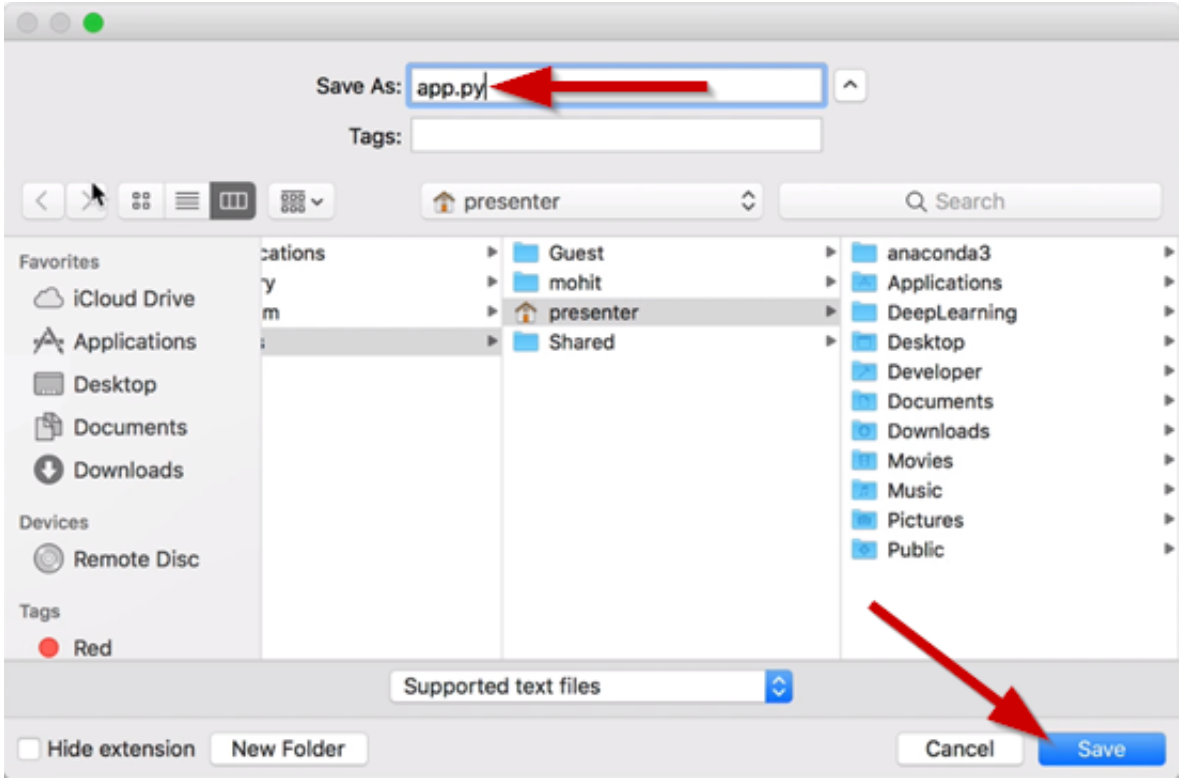
Now we want to launch the Spyder IDE, so just click the blue Launch button:



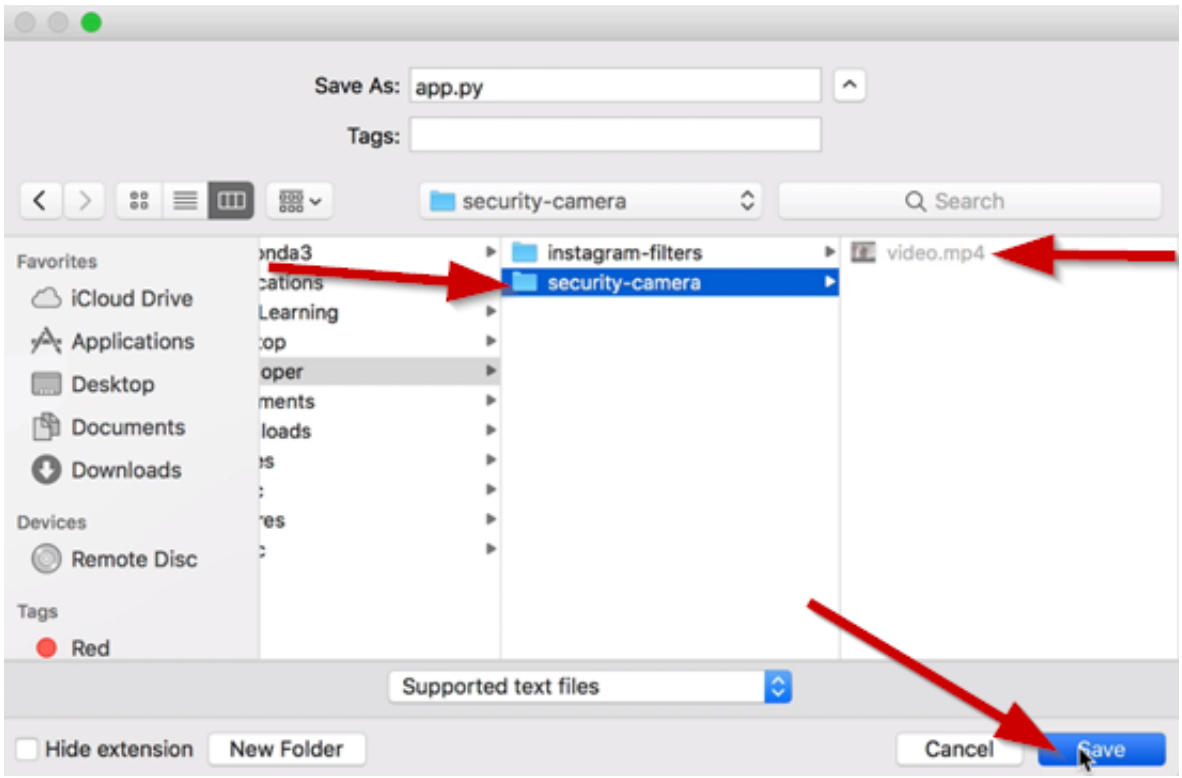
You should then be greeted with a blank file like this:



You can **delete the comment in green, and save this as "app.py"**



Save it in the same folder as the video.



So what we will be doing now is learning how to use the video capture class from OpenCV that will allow us to open or load a video or we can use feed from our webcam, and then we are going to display a single frame, the first frame of the video or from the webcam and then display that.

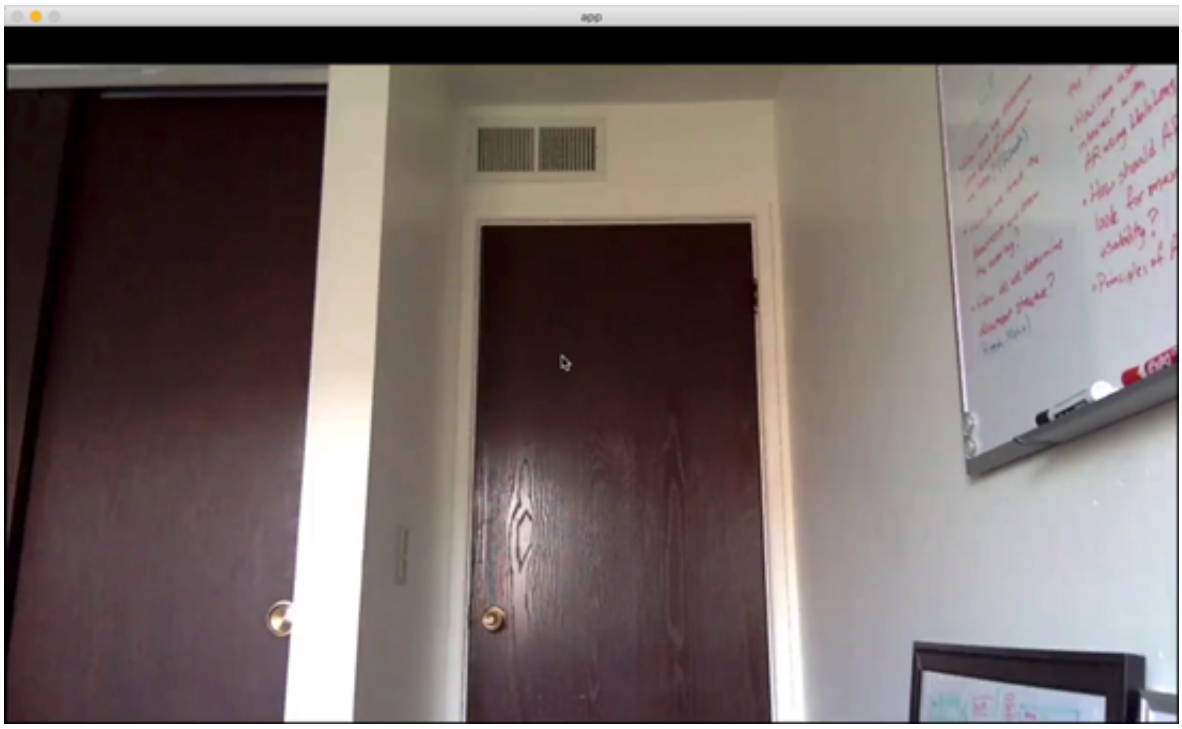
The first thing we need to do is import cv2, that's OpenCv.

See the code below and follow along:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import cv2

# open webcam (import video)
video_capture = cv2.VideoCapture('video.mp4')
# webcam: video_capture = cv2.VideoCapture(0)

# read first frame
_, current_frame = video_capture.read()

cv2.imshow('app', current_frame)
cv2.waitKey(0)

video_capture.release()
cv2.destroyAllWindows()
```

Save this file and **run this code** through the terminal.

So we have the single frame being displayed, the first frame from the video:



If you had the webcam active, it will take the frame from the webcam. You can press any key, and it will quit out of the app.

Now that we know that we can read frames form the video or the webcam, what we are going to do is setup our main UI loop and we will be outlining it using comments.

The first thing we need to do is convert our image to gray scale, because of the way that we have our image distance, or image similarity metrics,we're going to convert it to gray scale, because color isn't really going to offer us much value here.

See the code below and follow along:

```
# convert to grayscale
current_frame = cv2.cvtColor(current_frame, cv2.COLOR_BGR2GRAY)
```

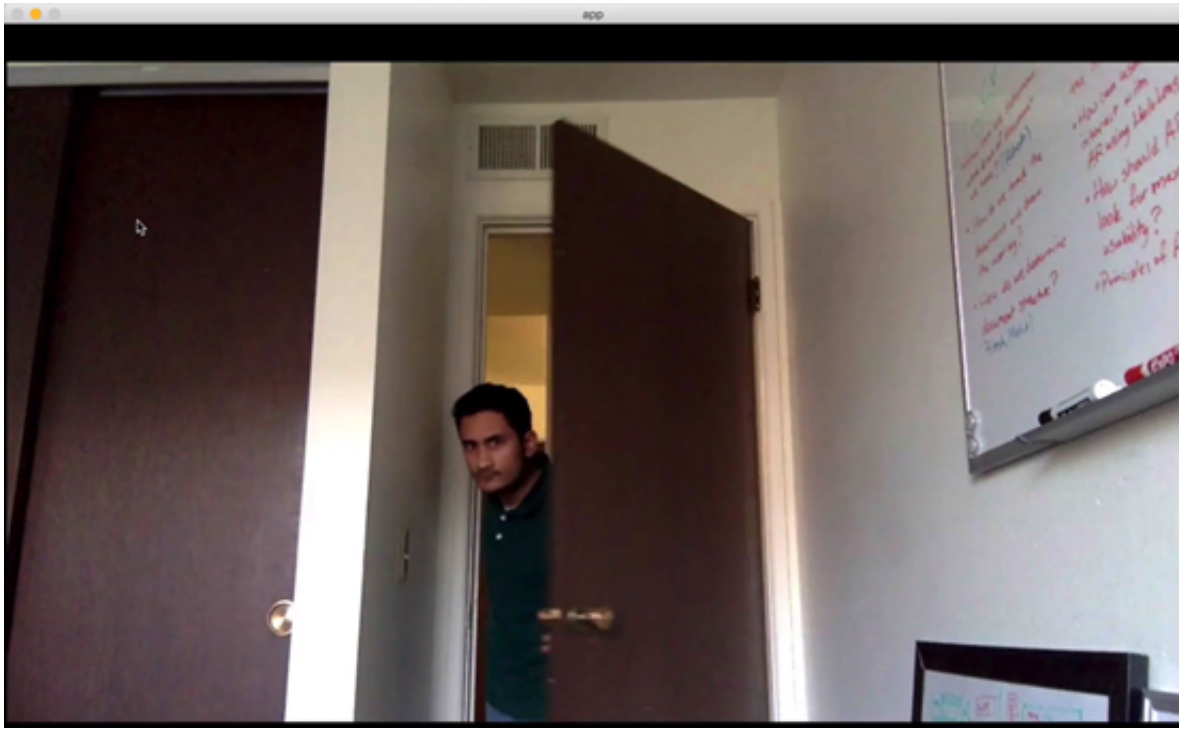
Save this and **run the code** through the terminal. You will be able to see the first frame, but its in gray scale:

See the code below and follow along:

```
# main loop
while True:
    # read in the next valid frame (and convert to grayscale)
    _, current_frame = video_capture.read()
    if current_frame is None:
        break

    #TODO: compare two frames
    #display the video/webcam feed
    cv2.imshow('app', current_frame)
    cv2.waitKey(1)
```


Save this and **run the code** through the terminal.

You should now see the entire video play or you feed from the webcam.



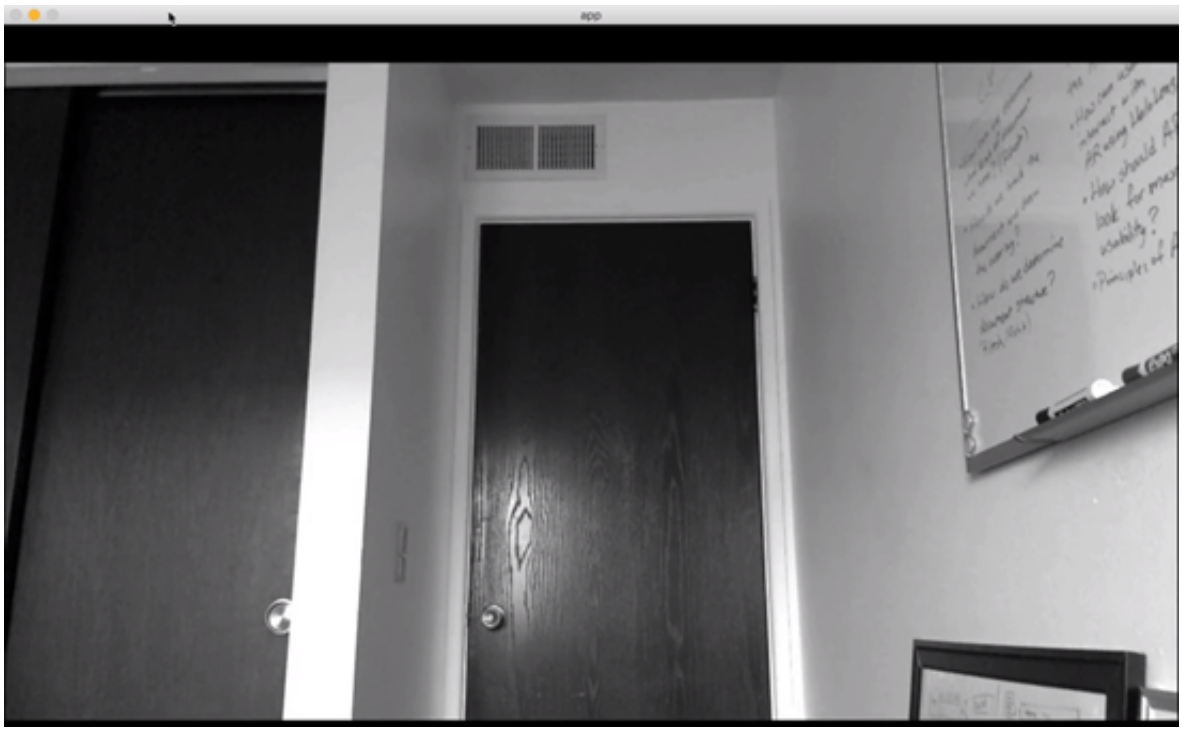
The video is playing kind of slowly, but you can change the wait key to something like “100 milliseconds” whihc will cause the video to play even slower.

One other thing to do in this lesson is to convert the frame into a gray scale image.

See the code below and follow along:

```
# main loop
while True:
    # read in the next valid frame (and convert to grayscale)
    _, current_frame = video_capture.read()
    if current_frame is None:
        break
    current_frame = cv2.cvtColor(current_frame, cv2.COLOR_BGR2GRAY)
```

Save this and **run the code** through the terminal and you will see everything in gray scale.



The code in the video for this particular lesson has been updated, please see the lesson notes below for the corrected code.

In this lesson, we will be setting up our distance metric and we are going to be using structural similarity metric that's in another package we'll have to import, and then we will also use it in our main loop so we will be implementing this compare_two_frames.

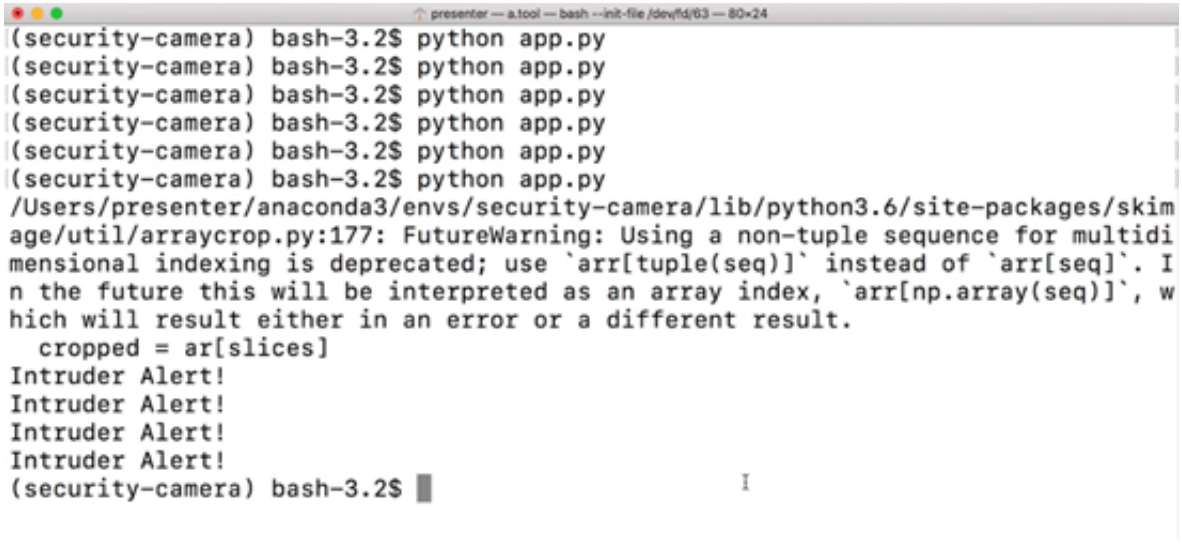
The following code has been updated, and differs from the video:

```
from skimage.metrics import structural_similarity as compare_ssim

def ssim(A, B):
    return compare_ssim(A, B, data_range=A.max()-A.min())

# initializes other frame
previous_frame = current_frame

# compare two frames
ssim_index = ssim(current_frame, previous_frame)
if ssim_index < 0.90:
     print('Intruder Alert!')

# updating previous frame
previous_frame = current_frame
```

Note that 'skimage.measure.compare_ssim' has been replaced by '**skimage.metrics.structural_similarity**'.

Save this and **run the code** through the terminal.

We can see that **"Intruder Alert!"** is being printed:



We now know that are comparison metric is working and we have our frame alternating, we have our frame setup is working as well. So depending on how sensitive you want the camera to be, you can adjust this value up or down.

Before we begin working with twilio there is a small improvement we can make to our security camera.

When looking at the video it runs very slowly and this because we're looking at every single pair of frames in our video and doing the comparison.

It would make more sense, and we will not lose any of the security if, instead of checking every pair of frames, we check every other frame, or every tenth frame, for example.

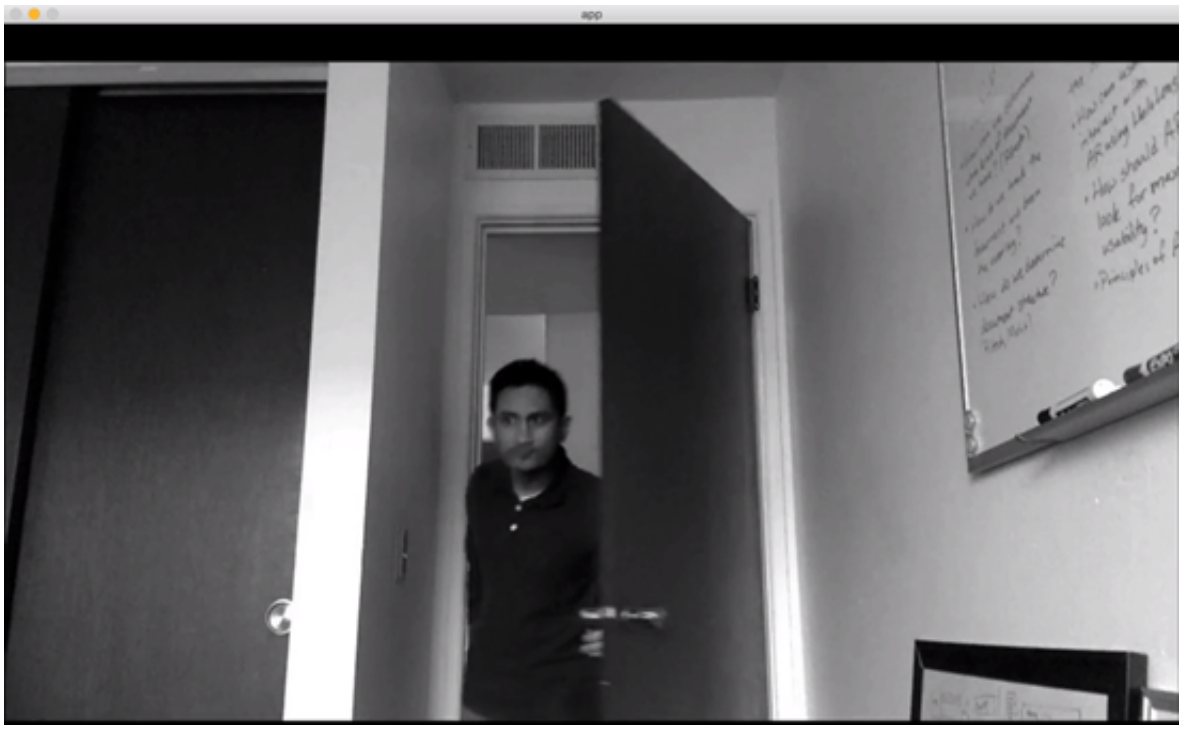
So, instead of updating the previous frame after every single iteration, maybe we only update it once every 10 times. Then that means that we are comparing one frame, and the previous frame comes ten frames before it.

See the code below and follow along:

```
frame_counter = 1

if frame_counter % 10 == 0:


# display the video/webcam feed
cv2.imshow('app', current_frame)
cv2.waitKey(1)
frame_counter += 1
```

Save this and **run the code** through the terminal.

You will notice that the video is going to seem to play a little bit faster because we are doing less comparisons.



We are also still getting the **“Intruder Alert!”** printed out.

```
presenter — a.tool — bash --init-file /dev/fd/63 — 80×24
(security-camera) bash-3.2$ python app.py
/Users/presenter/anaconda3/envs/security-camera/lib/python3.6/site-packages/
age/util/arraycrop.py:177: FutureWarning: Using a non-tuple sequence for mul
mensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]
n the future this will be interpreted as an array index, `arr[np.array(seq)]
hich will result either in an error or a different result.
  cropped = ar[slices]
Intruder Alert!
Intruder Alert!
Intruder Alert!
Intruder Alert!
(security-camera) bash-3.2$
```

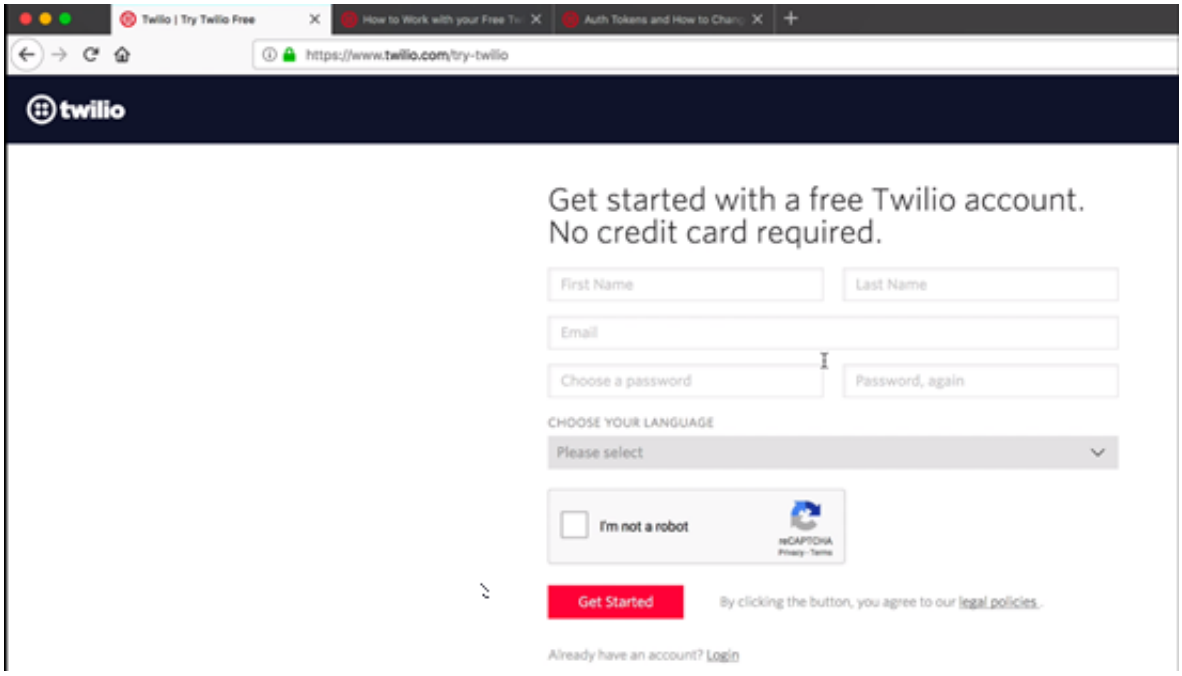

In this lesson we will be getting setup with twilio so that we can send text messages to any phone number from our Python app. So twilio is a service that allows us to as the website says, to send things like text messages or what not from a virtual phone number to a real phone number.

In order to set this up, you need to visit: https://www.twilio.com

The direct link is here: Twilio

We will be using the free trial version. **Click on the Get a free API key button.**

You will then have to sign up using the form with twilio:

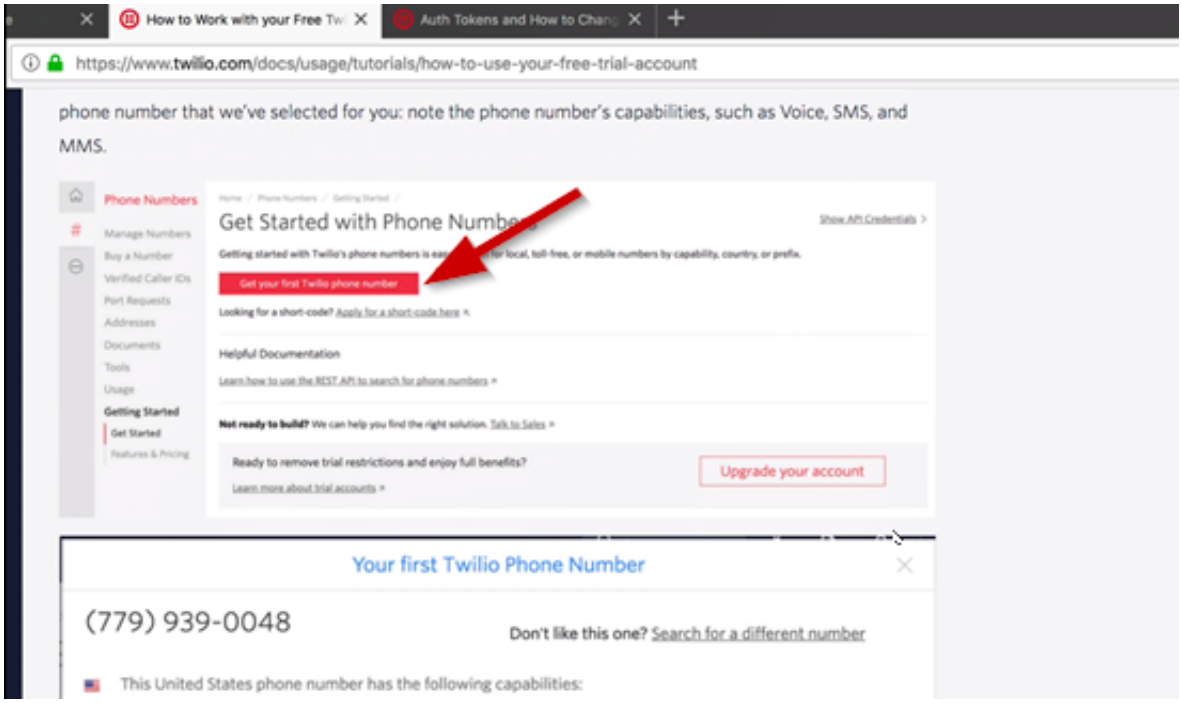


Once you complete this step you will be asked to verify your phone number.

There is great documentation on the twilio site if you need it in order to get setup with twilio.

We need to request a virtual phone number so that we can send a text message from the virtual phone number to our actual phone number. Because, we are in the trail version, we can request something like this.

Navigate to the “Get Started” page, adn click the button that says “Get your first twilio phone number”



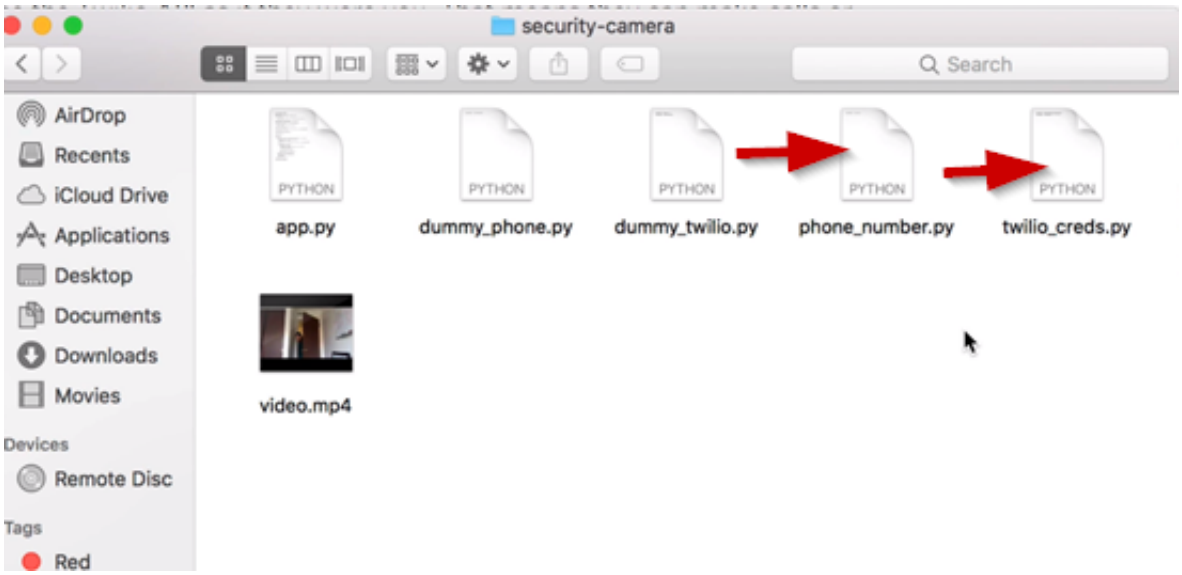
You will need to write this number down or remember it because you will need it to put in the code for our app.

There are two other things we need and they are the **account SID or the app location SID and then the authentication token.** These are two things that you will also want to keep private. You do not want to share these account details.

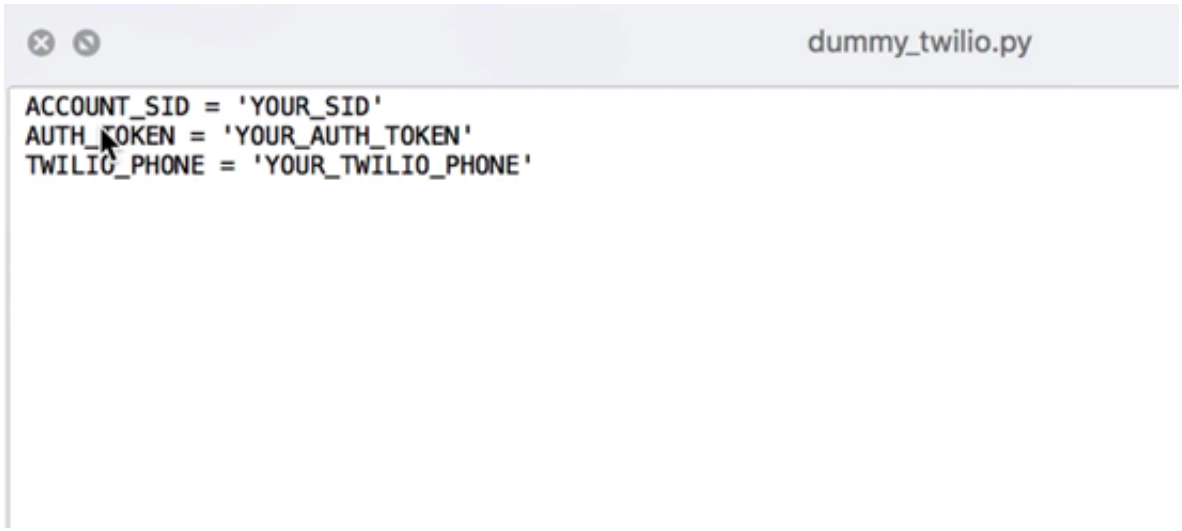
In order to see both these things, if you navigate to the console on the twilio website on the top right, you will see the Account SID and the Auth Token. **You will want to copy both these down.**

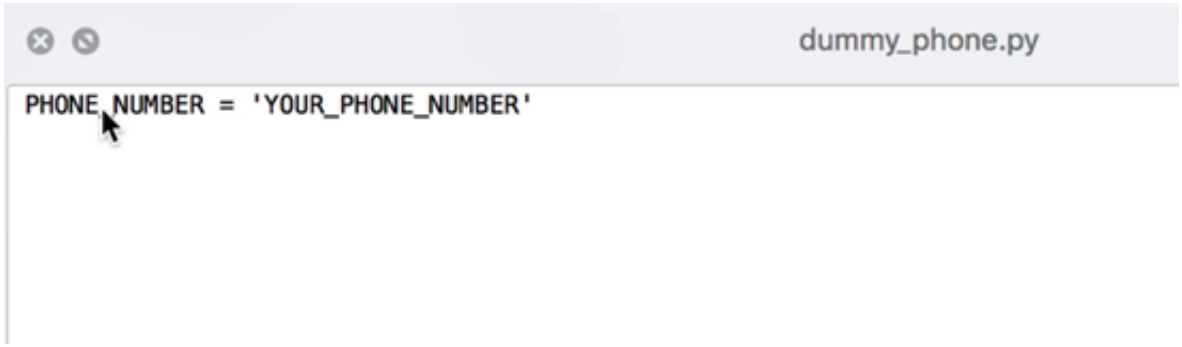
What we want to do is put these into a separate Python file so that our main app.py doesn’t have it in there in case someone were to read that.

So the **phone__number.py** and the **twilio_creds.py** hold the actual account SID and auth token:



You will want to **create these two files, and the file just has three lines:**





So we are now setup with twilio.

The code in the video for this particular lesson has been updated, please see the lesson notes below for the corrected code.

We are almost done with our security camera app. All we now need to do is just integrate our twilio information into the code.

First, we need to **import our account SID, our authentication token, twilio phone, and then the phone number for our different files and they're in the same directory as app.py.**

We also need to **import the twilio client.**

See the code below and follow along:

```
from twilio.rest import Client
from twilio_creds import ACCOUNT_SID, AUTH_TOKEN, TWILIO_PHONE
from phone_number import PHONE_NUMBER

# create the Twilio client
client = Client(ACCOUNT_SID, AUTH_TOKEN)

is_first_message = True

if ssim_index < 0.90 and is_first_message:
            # send text message using Twilio
            client.messages.create(body='Intruder Alert!', from_=TWILIO_PHONE, to=PHO
NE_NUMBER)
            print('Intruder Alert!')
            is_first_message = False
```

Save this and **run the code** through the terminal.

The video will play and as soon as you see one of the intruder alerts printed out, you will receive the text message.

So the security camera is now setup, and we are now finished with the app.

Congratulations on finishing the course!

The following code has been updated, and differs from the video:

Here is the entire **app.py file** for you to compare your code to if you are having any issues.

Note that 'skimage.measure.compare_ssim' has been replaced by '**skimage.metrics.structural_similarity**'.

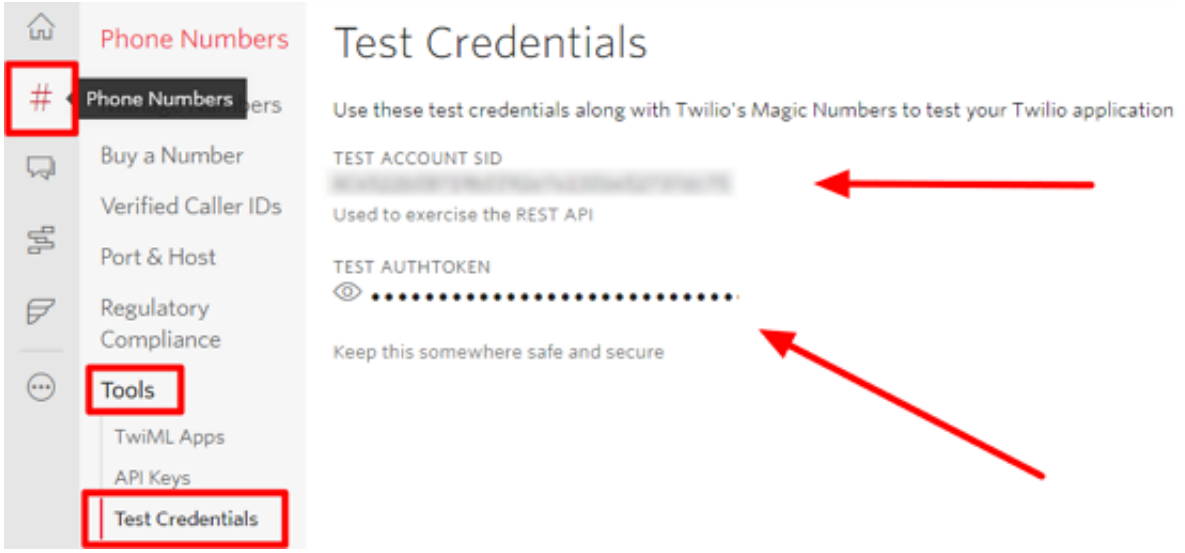
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import cv2
from skimage.metrics import structural_similarity as compare_ssim
from twilio.rest import Client
from twilio_creds import ACCOUNT_SID, AUTH_TOKEN, TWILIO_PHONE
from phone_number import PHONE_NUMBER
```



```
def ssim(A, B):
    return compare_ssim(A, B, data_range=A.max()-A.min())

# create the Twilio client
client = Client(ACCOUNT_SID, AUTH_TOKEN)

# open webcam (import video)
video_capture = cv2.VideoCapture('video.mp4')
# webcam: video_capture = cv2.VideoCapture(0)

# read first frame
_, current_frame = video_capture.read()

# convert to grayscale
current_frame = cv2.cvtColor(current_frame, cv2.COLOR_BGR2GRAY)

# initializes other frame
previous_frame = current_frame

frame_counter = 1
is_first_message = True

# main loop
while True:
    # read in the next valid frame (and convert to grayscale)
    _, current_frame = video_capture.read()
    if current_frame is None:
        break
    current_frame = cv2.cvtColor(current_frame, cv2.COLOR_BGR2GRAY)

    if frame_counter % 10 == 0:
        # compare two frames
        ssim_index = ssim(current_frame, previous_frame)
        if ssim_index < 0.90 and is_first_message:
            # send text message using Twilio
            client.messages.create(body='Intruder Alert!', from_=TWILIO_PHONE, to=PHO
NE_NUMBER)
            print('Intruder Alert!')
            is_first_message = False

        # updating previous frame
        previous_frame = current_frame

    # display the video/webcam feed
    cv2.imshow('app', current_frame)
    cv2.waitKey(1)
    frame_counter += 1

video_capture.release()
cv2.destroyAllWindows()
```

The following instructions have been updated, and differ from the video:

If you come across **Twilio errors** while running your code is due to the **trial** Twilio account not allowing your number to send SMSes. To fix that, go to '**Phone Numbers**' menu option ("**#**"), right below the 'Console Dashboard', go to **Tools > Test Credentials** and use the '**TEST ACCOUNT SID**' and '**TEST AUTHTOKEN**' info in your '**twilio_creds.py**' file instead of the 'ACCOUNT SID' and 'AUTH TOKEN' as shown in the course:



Also, in this case, use the Twilio test phone number '**+15005550006**' in your .py file too.