What Is Computer Vision?

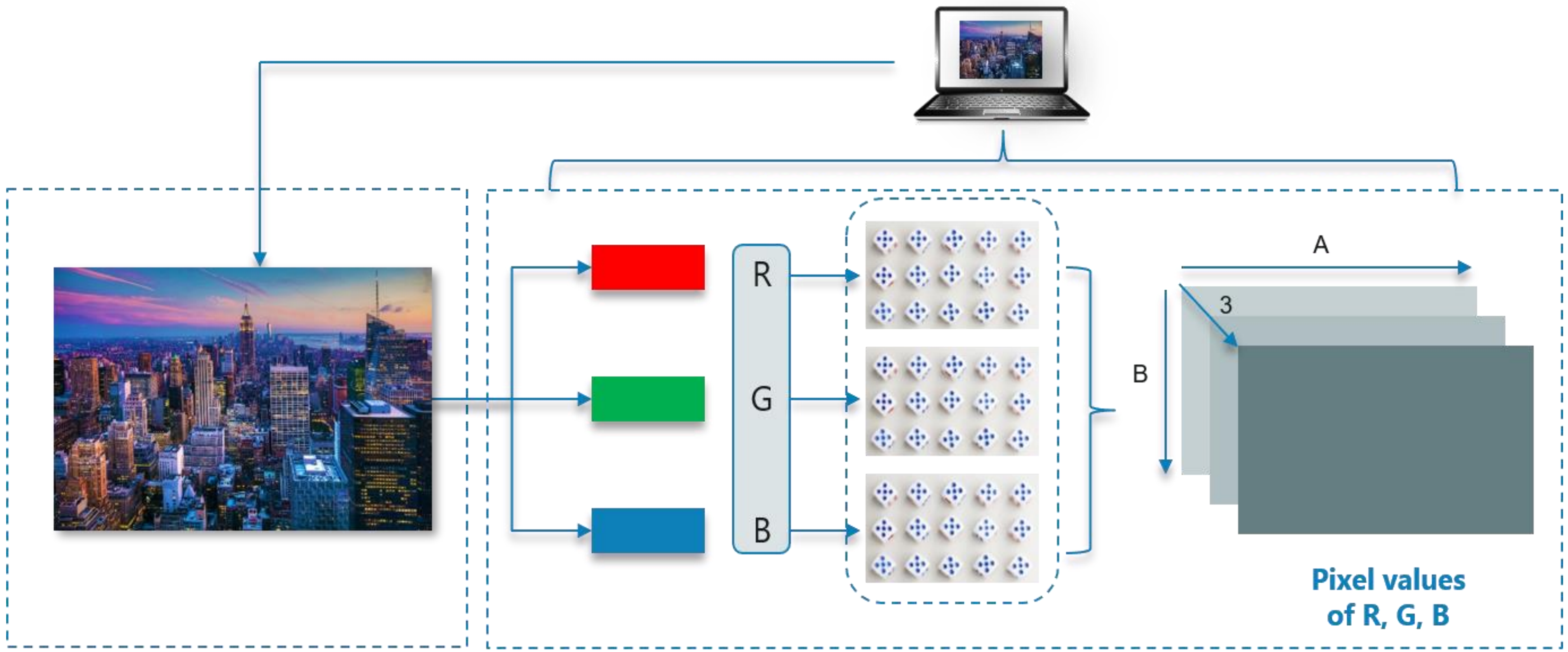
- To simplify the answer to this – Let us consider a scenario.
- We all use Facebook, correct? Let us say you and your friends went on a vacation and you clicked a lot of pictures and you want to upload them on Facebook and you did. But now, wouldn’t it take so much time just to find your friends faces and tag them in each and every picture? Well, Facebook is intelligent enough to actually tag people for you.
- So, how do you think the auto tag feature works? In simple terms, it works on computer vision.
- Computer Vision is an interdisciplinary field that deals with how computers can be made to gain a high-level understanding from digital images or videos.
- The idea here is to automate tasks that the human visual systems can do. So, a computer should be able to recognize objects such as that of a face of a human being or a lamppost or even a statue.

How Does A Computer Read An Image?

- Consider the below image:

- We can figure out that it is an image of the New York Skyline. But, can a computer find this out all on its own? The answer is no!
- The computer reads any image as a range of values between 0 and 255.
- For any color image, there are 3 primary channels – Red, green and blue. How it works is pretty simple.
- A matrix is formed for every primary color and later these matrices combine to provide a Pixel value for the individual R, G, B colors.
- Each element of the matrices provide data pertaining to the intensity of brightness of the pixel.

Consider the following image:



- As shown, the size of the image here can be calculated as B x A x 3.
- Note: For a black-white image, there is only one single channel.

What Is OpenCV?

- OpenCV is a Python library which is designed to solve computer vision problems. OpenCV was originally developed in 1999 by Intel but later it was supported by Willow Garage.
- OpenCV supports a wide variety of programming languages such as C++, Python, Java etc. Support for multiple platforms including Windows, Linux, and MacOS.
- OpenCV Python is nothing but a wrapper class for the original C++ library to be used with Python. Using this, all of the OpenCV array structures gets converted to/from NumPy arrays.
- This makes it easier to integrate it with other libraries which use NumPy. For example, libraries such as SciPy and Matplotlib.

Loading an image using OpenCV:

- Import cv2
- # colored Image
- Img = cv2.imread (‘1.jpg’,1)
- # Black and White (gray scale)
- Img_1 = cv2.imread (‘1.jpg’,0)

Image Shape/Resolution:

- Import cv2
- # Black and White (gray scale)
- Img = cv2.imread (“1.jpg”,0)
- Print(img.shape)
- By shape of the image, we mean the shape of the NumPy array. As you see from executing the code, the matrix consists of 768 rows and 1024 columns.

Displaying the image:

- import cv2
- import img as img

# Black and White (gray scale)

- Img = cv2.imread('bill.png', 0)
- print(Img)
- cv2.imshow('bill', Img)
- cv2.waitKey(0)
- cv2.waitKey(2000)
- cv2.destroyAllWindows()

Explanation:

- As you can see, we first import the image using **imread.** We require a window output to display the images, right?
- We use the **imshow** function to display the image by opening a window. There are 2 parameters to the **imshow** function which is the name of the window and the image object to be displayed.
- Later, we wait for a user event. **waitKey** makes the window static until the user presses a key. The parameter passed to it is the time in milliseconds.
- And lastly, we use **destroyAllWindows** to close the window based on the waitForKey parameter.
-

Resizing the image:

- import cv2
- # Black and White (gray scale)
- img = cv2.imread('bill.png', 0)
- resized_image = cv2.resize(img, (650, 500))
- cv2.imshow('bill', resized_image)
- cv2.waitKey(0)
- cv2.destroyAllWindows()

Explanation:

- Here, **resize** function is used to resize an image to the desired shape. The parameter here is the shape of the new resized image.
- Later, do note that the image object changes from **img** to **resized_image,** because of the image object changes now.
- Rest of the code is pretty simple to the previous one, correct?

Way to resize:

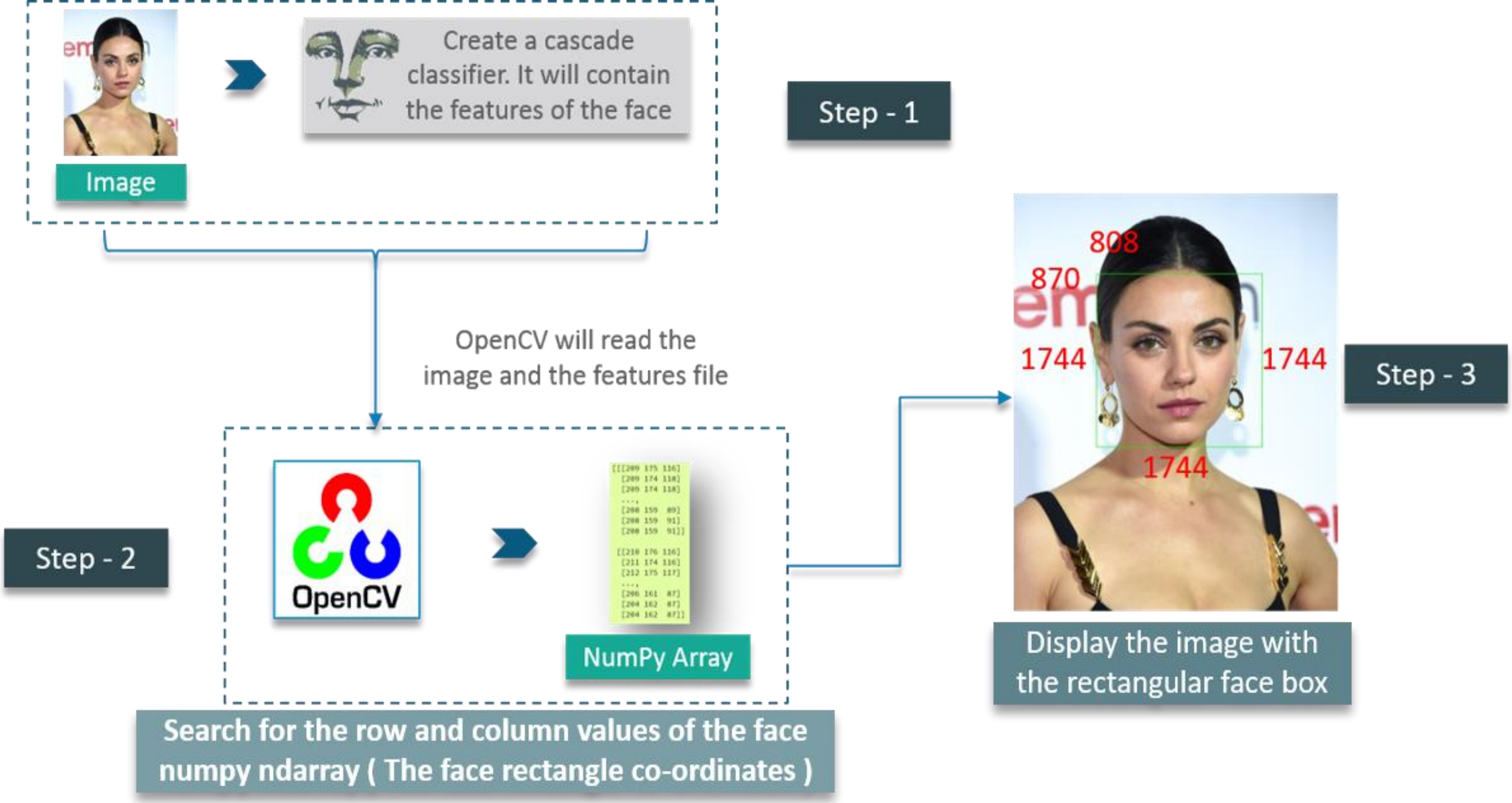
- Resized_image = cv2.resize(img, int(img.shape[1]*2), int(img.shape[0]*2)))
- Here, we get the new image shape to be half of that of the original image.

Face Detection Using OpenCV

- This seems complex at first but it is very easy. Let me walk you through the entire process and you will feel the same.
- **Step 1:** Considering our prerequisites, we will require an image, to begin with. Later we need to create a cascade classifier which will eventually give us the features of the face.
- **Step 2:** This step involves making use of OpenCV which will read the image and the features file. So at this point, there are NumPy arrays at the primary data points.
- All we need to do is to search for the row and column values of the face NumPy ndarray. This is the array with the face rectangle coordinates.

Continue…

- **Step 3:** This final step involves displaying the image with the rectangular face box.
- Check out the following image, here I have summarized the 3 steps in the form of an image for easier readability:
- Pretty straightforward, correct?



Create a cascade classifier. It will contain the features of the face

OpenCV will read the image and the features file

```
[[[209 175 116]
  [209 174 118]
  [209 174 118]
  ...,
  [208 159  89]
  [208 159  91]
  [208 159  91]]

 [[210 176 116]
  [211 174 116]
  [212 175 117]
  ...,
  [206 161  87]
  [204 162  87]
  [204 162  87]]
```

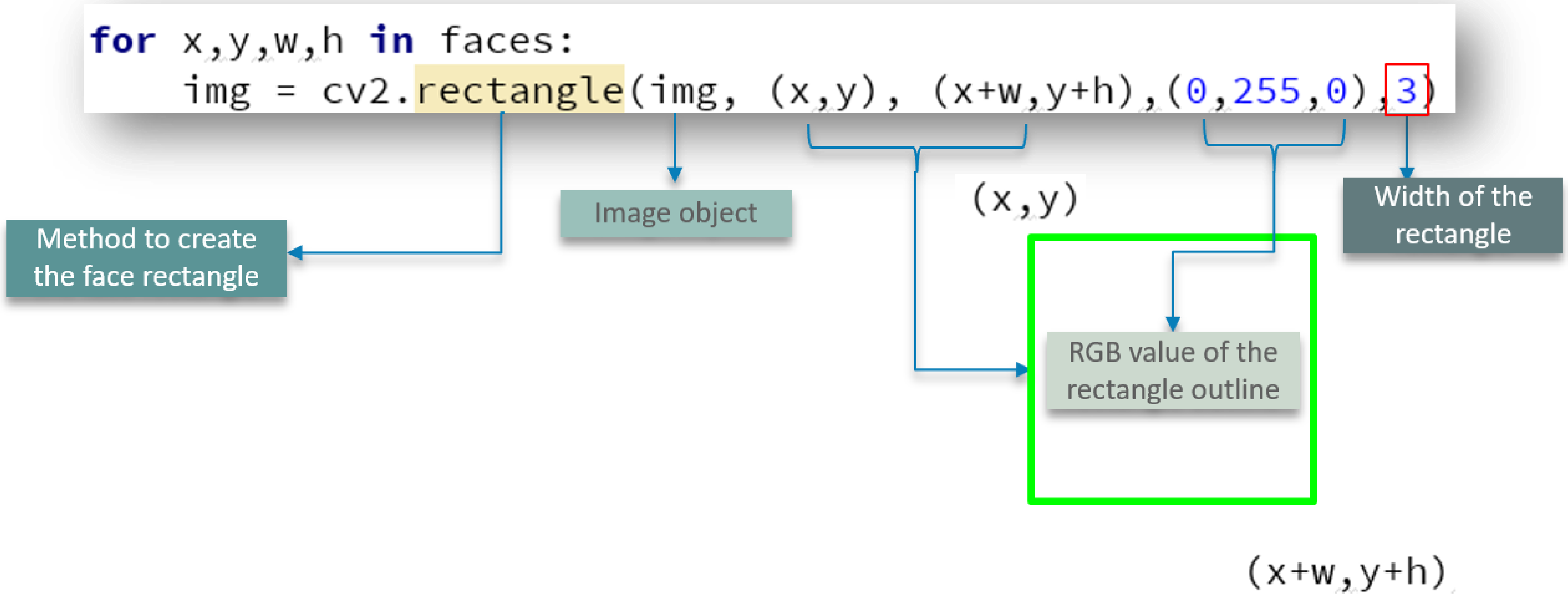
Search for the row and column values of the face numpy ndarray (The face rectangle co-ordinates)

Display the image with the rectangular face box

Explanation:

- First, we create a **CascadeClassifier** object to extract the features of the face as explained earlier. The path to the XML file which contains the face features is the parameter here.
- The next step would be to read an image with a face on it and convert it into a black and white image using **COLOR_BGR2GREY**. Followed by this, we search for the coordinates for the image. This is done using **detectMultiScale**.
- What coordinates, you ask? It's the coordinates for the face rectangle. The **scaleFactor** is used to decrease the shape value by 5% until the face is found. So, on the whole – Smaller the value, greater is the accuracy.
- Finally, the face is printed on the window.

Adding the rectangular face box:



```
for x,y,w,h in faces:
    img = cv2.rectangle(img, (x,y), (x+w,y+h),(0,255,0),3)
```

Method to create the face rectangle
Image object
(x,y)
RGB value of the rectangle outline
Width of the rectangle
(x+w,y+h)

Explanation:

- We define the method to create a rectangle using **cv2.rectangle** by passing parameters such as the image object, RGB values of the box outline and the width of the rectangle.
- Let us check out the entire code for face detection:

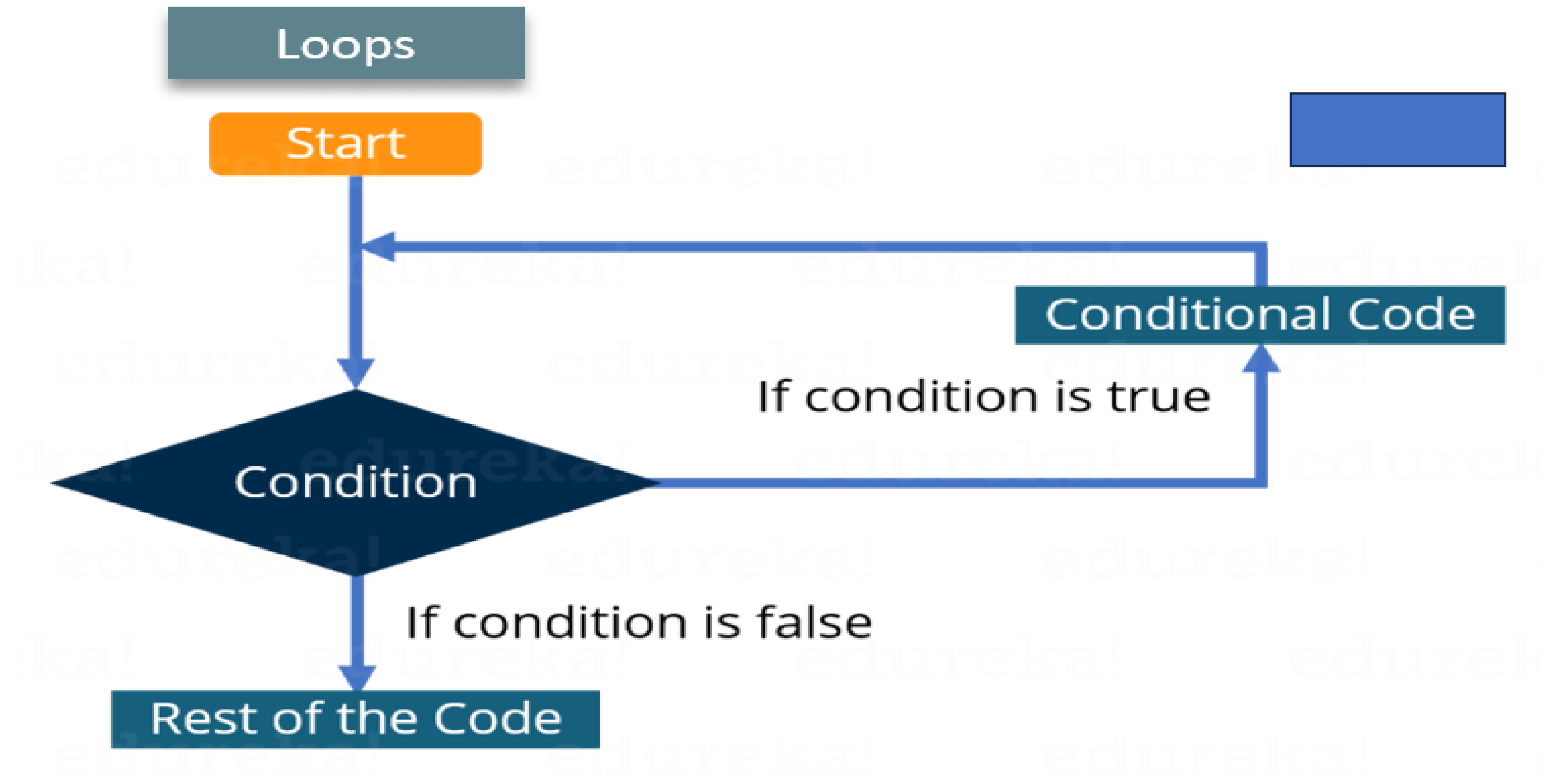
Face detection code

- import cv2

- # Create a CascadeClassifier Object
- face_cascade = cv2.CascadeClassifier('cascade_frontface_default.xml')
- # Reading the image as it is
- img = cv2.imread('bill.png')
- # Reading the image as gray scale image
- gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
- print(gray_img)

- # Search the co-ordintes of the image
- faces = face_cascade.detectMultiScale(gray_img, scaleFactor=1.05, minNeighbors=5)
- for x, y, w, h in faces:
-     img = cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 3)
- resized = cv2.resize(img, (int(img.shape[1] / 2), int(img.shape[0] / 2)))
- cv2.imshow("Gray", resized)
- cv2.waitKey(0)
- cv2.destroyAllWindows()

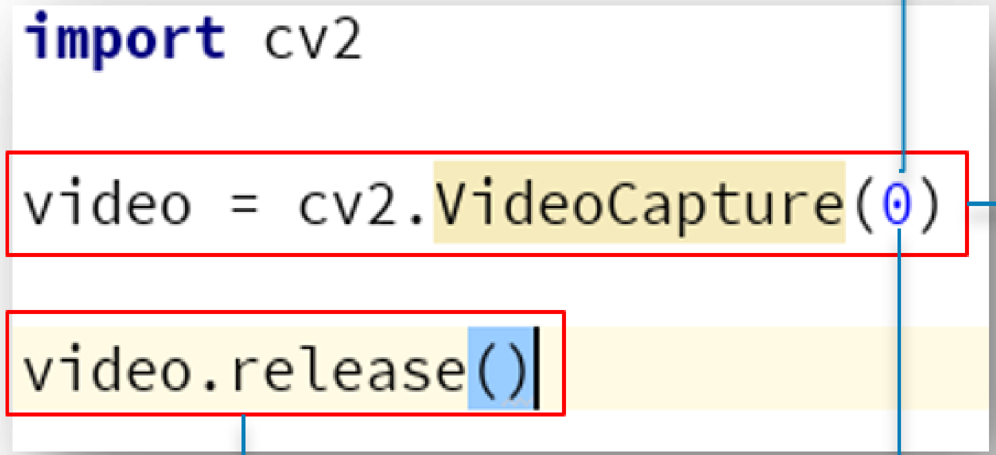
Capturing Video Using OpenCV



Explanation:

- The images are read one-by-one and hence videos are produced due to fast processing of frames which makes the individual images move.

Capturing Video:



```
import cv2

video = cv2.VideoCapture(0)

video.release()
```

Either give the path to the video file or use numbers. Numbers specify that you will be using the webcam to capture video

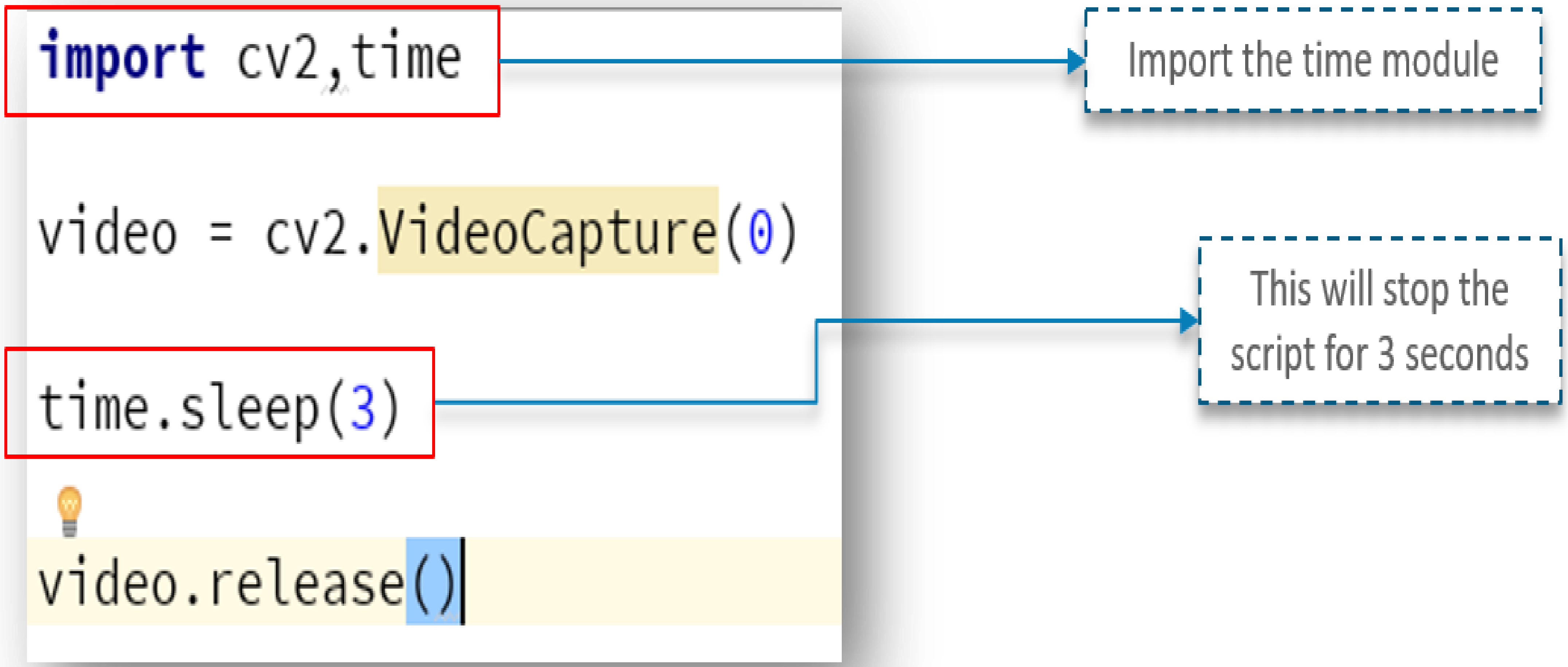
Method to create VideoCapture object. It will trigger the camera

This will release the camera in some milliseconds

'0' is to specify that use built-in camera

Explanation:

- First, we import the OpenCV library as usual. Next, we have a method called **VideoCapture** which is used to create the VideoCapture object. This method is used to trigger the camera on the user's machine. The parameter to this function denotes if the program should make use of the built-in camera or an add-on camera. '0' denotes the built-in camera in this case.
- And lastly, the **release** method is used to release the camera in a few milliseconds.
- When you go ahead and type in and try to execute the above code, you will notice that the camera light switches on for a split second and turns off later. Why does this happen?
- This happens because there is no time delay to keep the camera functional.

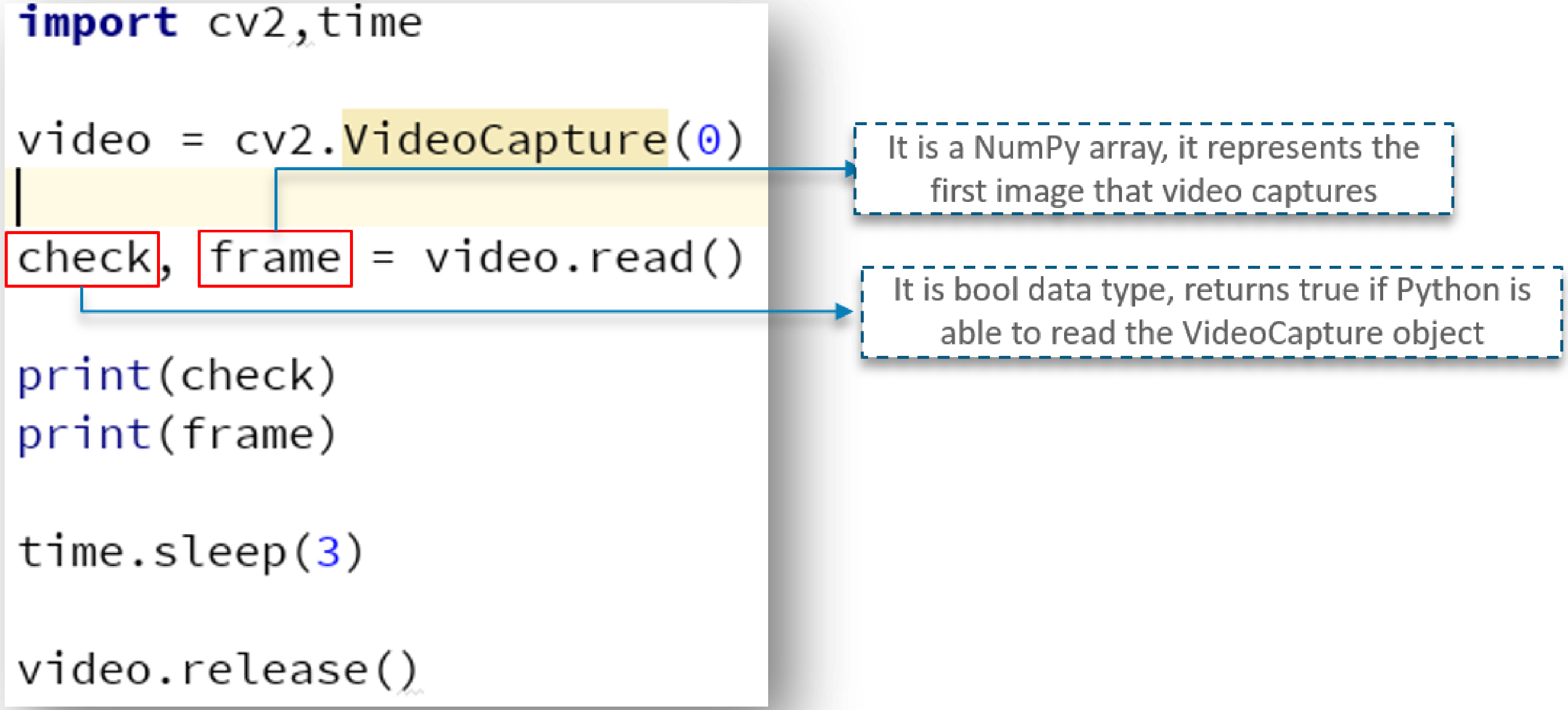


```
import cv2,time

video = cv2.VideoCapture(0)

time.sleep(3)

video.release()
```

- Looking at the above code, we have a new line called **time.sleep(3)** – This makes the script to stop for 3 seconds. Do note that the parameter passed is the time in seconds. So, when the code is executed, the webcam will be turned on for 3 seconds.

Adding the window:

- Adding a window to show the video output is pretty simple and can be compared to the same methods used for images. However, there is a slight change. Check out the following code:



```
import cv2,time

video = cv2.VideoCapture(0)

check, frame = video.read()

print(check)
print(frame)

time.sleep(3)

video.release()
```

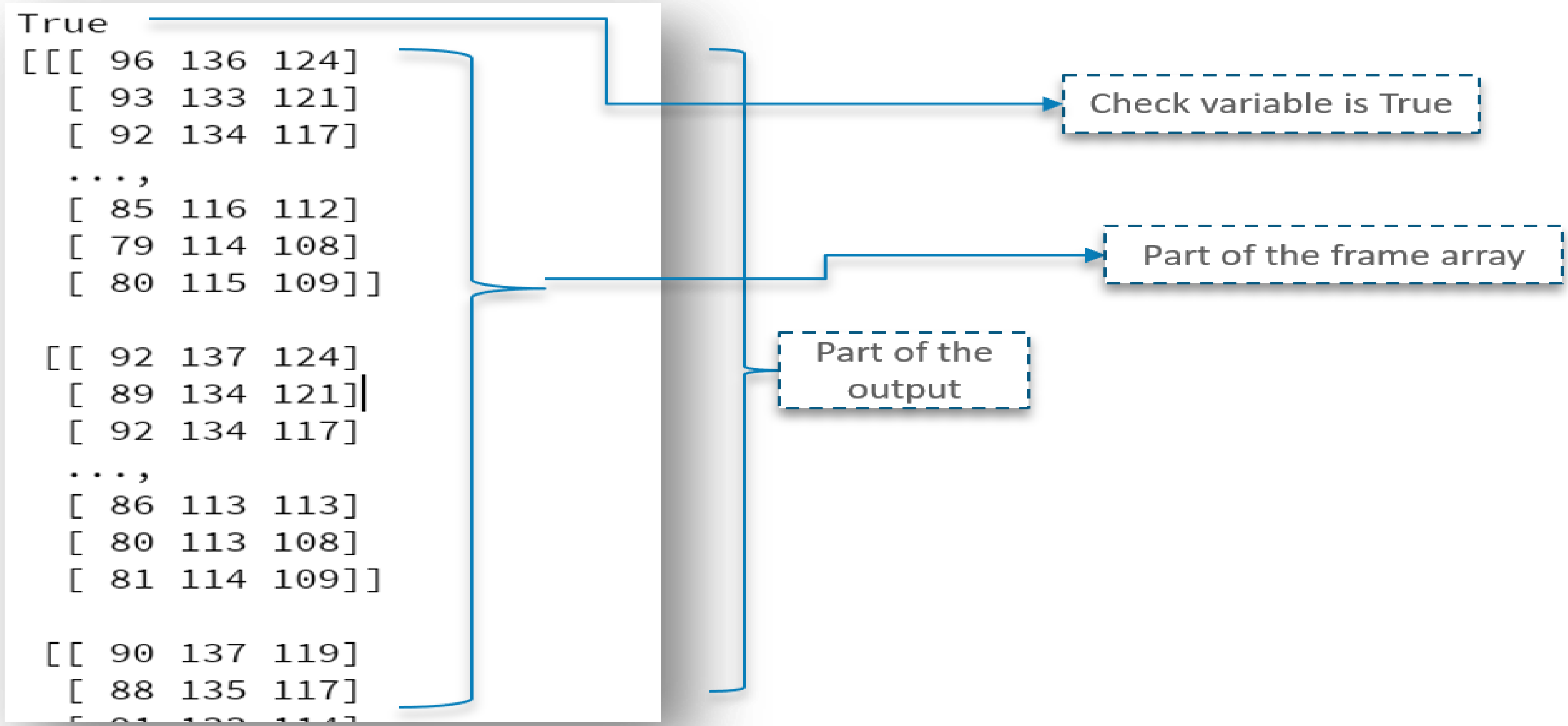
It is a NumPy array, it represents the first image that video captures

It is bool data type, returns true if Python is able to read the VideoCapture object

Explanation:

- Here, we have defined a NumPy array which we use to represent the first image that the video captures – This is stored in the **frame** array.
- We also have **check** – This is a boolean datatype which returns **True** if Python is able to access and read the **VideoCapture** object.

Output:

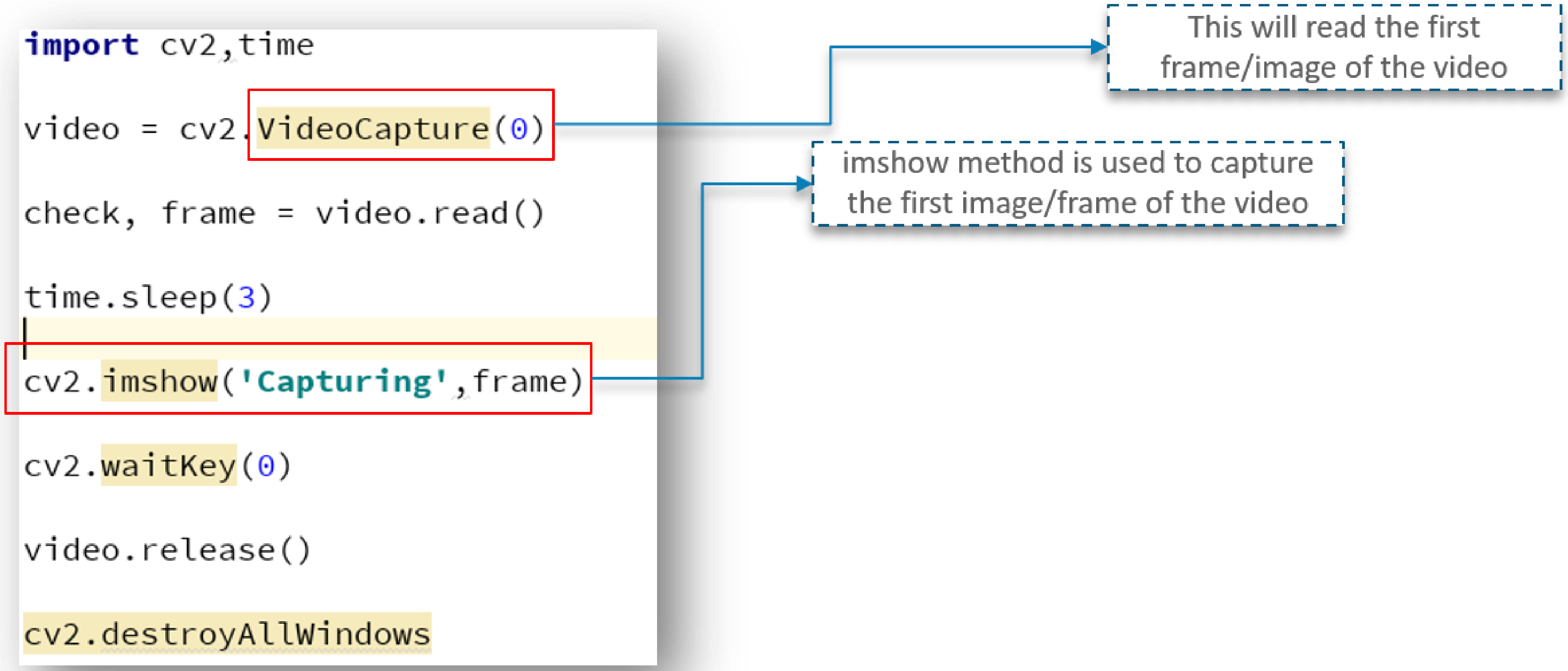

```
True
[[[ 96 136 124]
  [ 93 133 121]
  [ 92 134 117]
  ...,
  [ 85 116 112]
  [ 79 114 108]
  [ 80 115 109]]

 [[ 92 137 124]
  [ 89 134 121]
  [ 92 134 117]
  ...,
  [ 86 113 113]
  [ 80 113 108]
  [ 81 114 109]]

 [[ 90 137 119]
  [ 88 135 117]
```

Explanation:

- As you can check out, we got the output as **True** and the part of the frame array is printed.
- But we need to read the first frame/image of the video to begin, correct?
- To do exactly that, we need to first create a frame object which will read the images of the **VideoCapture** object.

As seen above, the imshow method is used to capture the first frame of the video.

All this while, we have tried to capture the first image/frame of the video but directly capturing the video.



```
import cv2,time

video = cv2.VideoCapture(0)

check, frame = video.read()

time.sleep(3)

cv2.imshow('Capturing',frame)

cv2.waitKey(0)

video.release()

cv2.destroyAllWindows
```

This will read the first frame/image of the video
imshow method is used to capture the first image/frame of the video

Installing Tesseract & Detecting Text from Image

- https://github.com/UB-Mannheim/tesseract/wiki
- https://pythongeeks.org/python-opencv-text-detection-and-extraction/