Basics of OpenCV

By Rahul Ray

Introduction

OpenCV is a massive open-source library for various fields like computer vision, machine learning, and image processing and plays a critical function in real-time operations, which are fundamental in today's systems. It is deployed for the detection of items, faces, Diseases, lesions, Number plates, and even handwriting in various images and videos. With the help of OpenCV in Deep Learning, we deploy vector space and execute mathematical operations on these features to identify visual patterns and their various features.

What is Computer Vision?

Computer vision is an approach to understanding how photos and movies are stored, as well as manipulating and extracting information from them. Artificial Intelligence depends on or is mostly based on computer vision. Self-driving cars, robotics, and picture editing apps all rely heavily on computer vision.

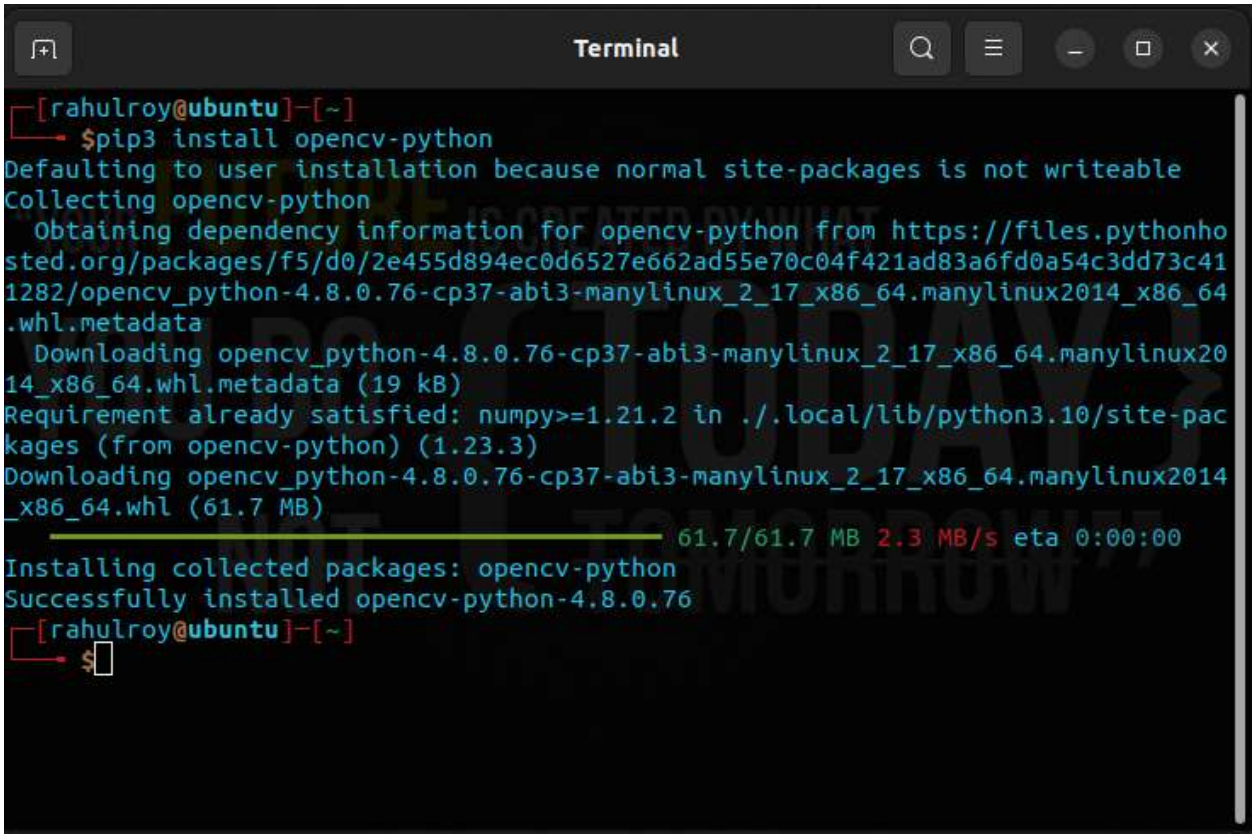
Human vision has a resemblance to that of computer vision. Human vision learns from various life experiences and deploys them to distinguish objects interpret the distance between various objects and estimate the relative position.

Installing OpenCV

For this tutorial I used OpenCV with Python, you can also use it with C and C++.

For installing OpenCV run the command below…..

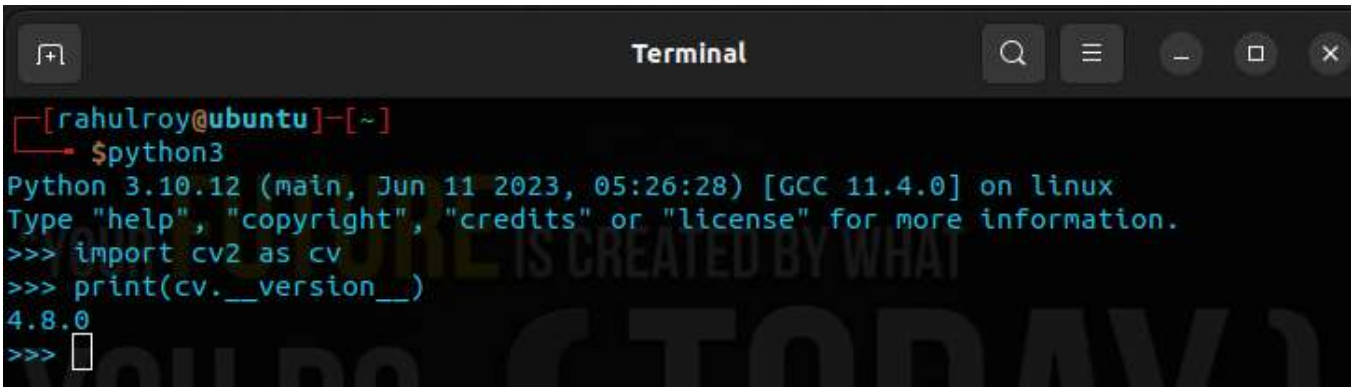
`pip3 install opencv-python` or `pip install oprncv-python`



Importing OpenCV into your code

A package in Python is a collection of modules that contain pre-written programs. These packages allow you to import modules separately or as a whole. Importing the package is as simple as calling the “cv2” module as seen below:

```
import cv2 as cv
```



Reading, Writing, and Displaying an Input Image

- Read an image from a file (using **cv.imread**())
- Display an image in an OpenCV window (using **cv.imshow**())
- Write an image to a file (using **cv.imwrite**())

Source codes

```
import cv2 as cv

img = cv.imread('images/manogya.jpeg')

cv.imshow("manogya", img)

cv.waitKey(0)
```

Explanation

As a first step, the OpenCV python library is imported. The proper way to do this is to additionally assign it the name *cv*, which is used in the following to reference the library.

```
import cv2 as cv
```

Now, let's analyze the main code. As a first step, we read the image "manogya.jpg" from the image folder. In order to do so, a call to the **cv.imread()** function loads the image using the file path specified by the first argument.

```
img = cv.imread('images/manogya.jpeg')
```

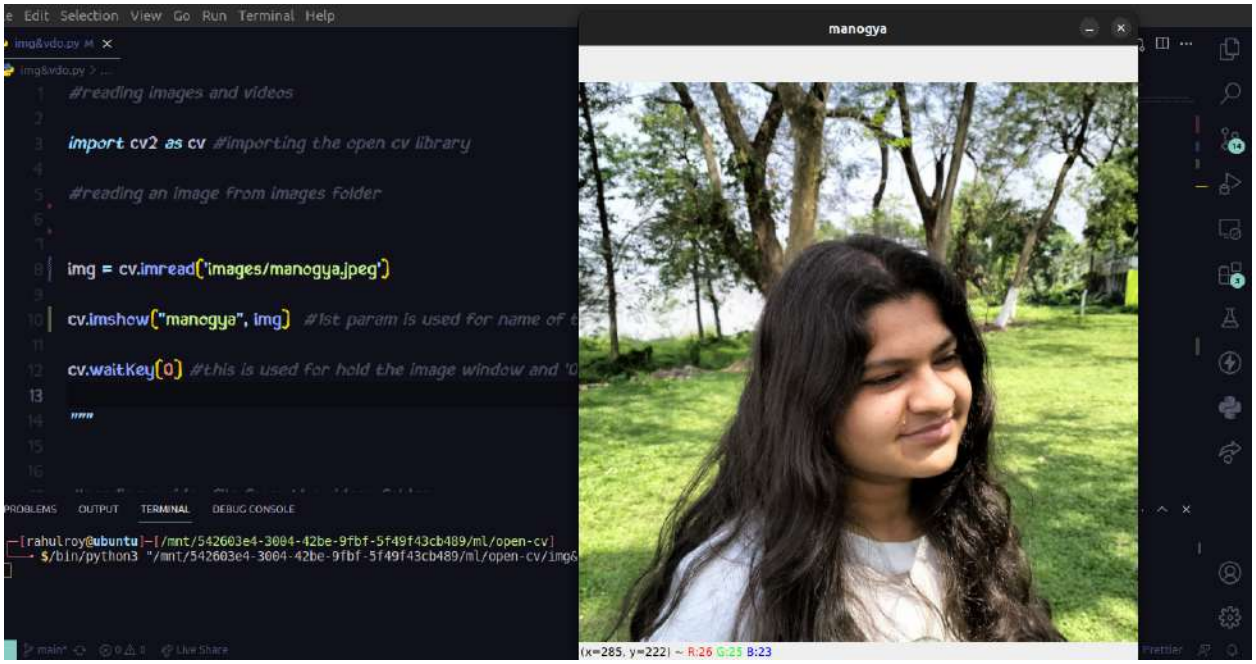
Then, the image is shown using a call to the **cv.imshow()** function. The first argument is the title of the window and the second argument is the **cv::Mat** object that will be shown.

```
cv.imshow("manogya", img)
```

Because we want our window to be displayed until the user presses a key (otherwise the program would end far too quickly), we use the **cv.waitKey()** function whose only parameter is just how long should it wait for a user input (measured in milliseconds). Zero means to wait forever. The return value is the key that was pressed.

```
cv.waitKey(0)
```

Output



The second argument in **cv.imread()** function 0, 1, and -1.

```
img = cv.imread('images/manogya.jpeg', 0)
```

Now, the image will be read in grayscale

```
img = cv.imread('images/manogya.jpeg', 1)
```

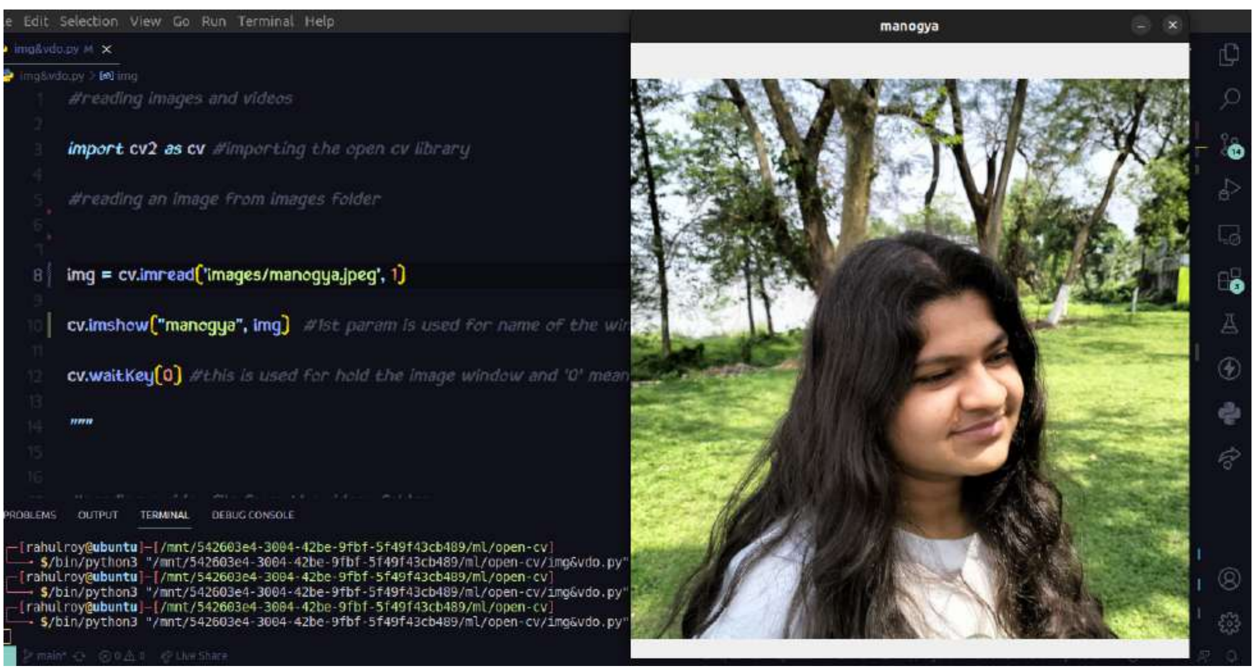
This is the default value if you do not pass 1 as the second argument, then it is automatically set to 1

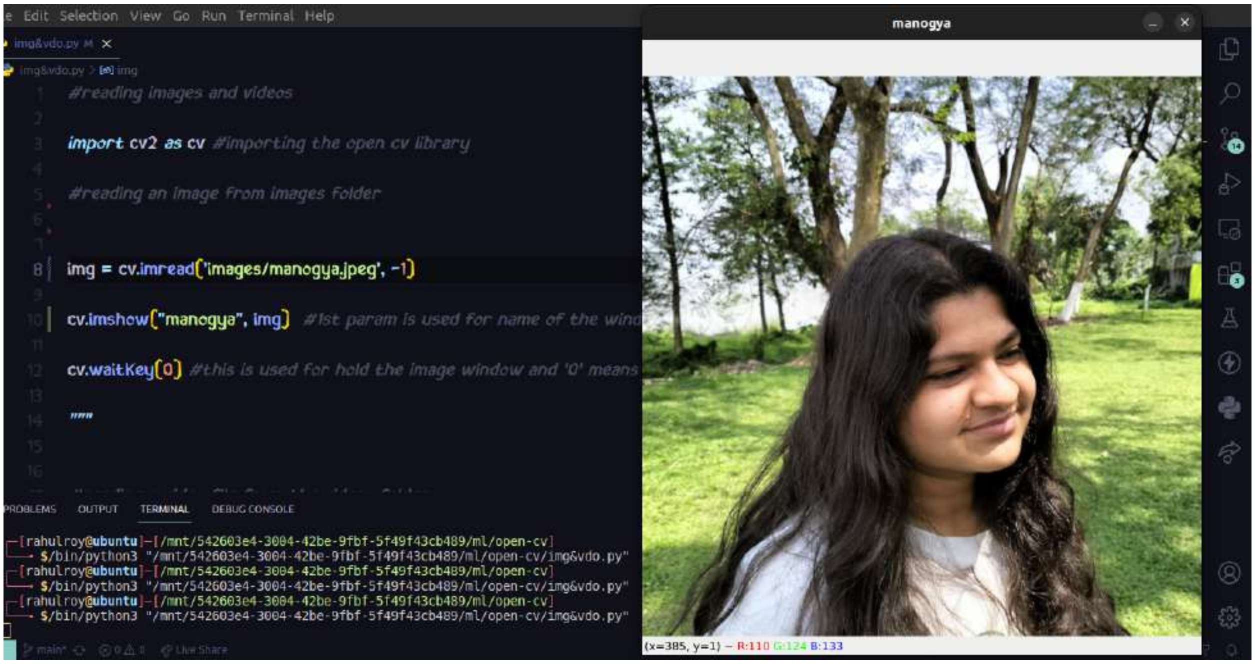
```
img = cv.imread('images/manogya.jpeg', -1)
```

-1 slightly increases the saturation of the image

Outputs


manogya
#reading images and videos
import cv2 as cv #importing the open cv library
#reading an image from images folder
img = cv.imread('images/manogya.jpeg', 0)
cv.imshow("manogya", img) #1st param is used for name of the w
cv.waitKey(0) #this is used for hold the image window and '0' mea
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE


manogya
#reading images and videos
import cv2 as cv #importing the open cv library
#reading an image from images folder
img = cv.imread('images/manogya.jpeg', 1)
cv.imshow("manogya", img) #1st param is used for name of the wi
cv.waitKey(0) #this is used for hold the image window and '0' mean
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE



In the end, the image is written to a file if the pressed key is the "s"-key. For this, the **cv.imwrite()** function is called which has the file path and the **cv::Mat** object as an argument.

In the **cv.imwrite()** function, the first argument will take the name of the output file and the second argument will take the image.

Source code

```
import cv2 as cv

img = cv.imread('images/manogya.jpeg', -1)

cv.imshow("manogya", img)

k = cv.waitKey(0)

if k == ord("s"):

    cv.imwrite("mano.jpeg", img)
```

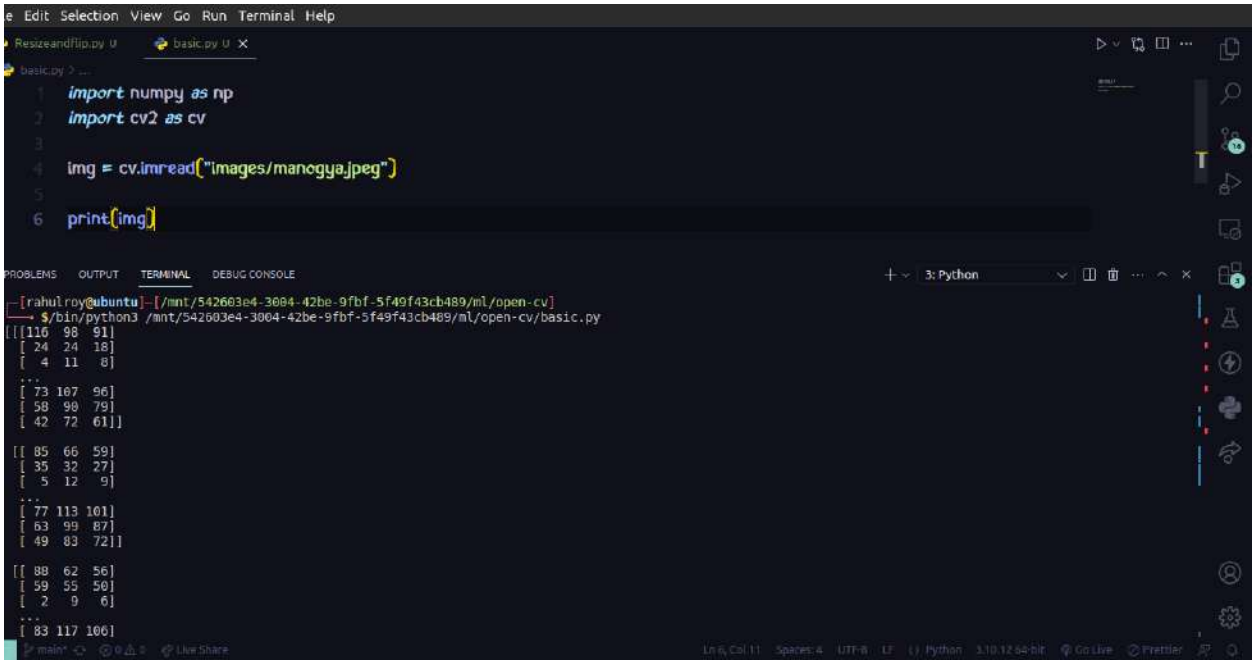
Now let's talk about some basics...

OpenCV reads the image in numpy array of pixel values of the image.

Source code

```
import numpy as np

import cv2 as cv

img = cv.imread("images/manogya.jpeg")

print(img)
```

Output



You can notice that a 3D array is printed in the console because the **cv.imread()** function reads the image in the BGR format, which means Blue, Green, and Red. In OpenCV, a color image is a combination of these color layers. Let's see all of them with a single colored layer.

Source Code

```
import numpy as np

import cv2 as cv

img = cv.imread("images/manogya.jpeg")

img = cv.resize(img, (300, 300))

L1 = img[:, :,0]

L2 = img[:, :,1]

L3 = img[:, :,2]

final = np.concatenate((L1, L2, L3), axis=1)

cv.imshow("layers", final)

cv.waitKey()
```

Explanation

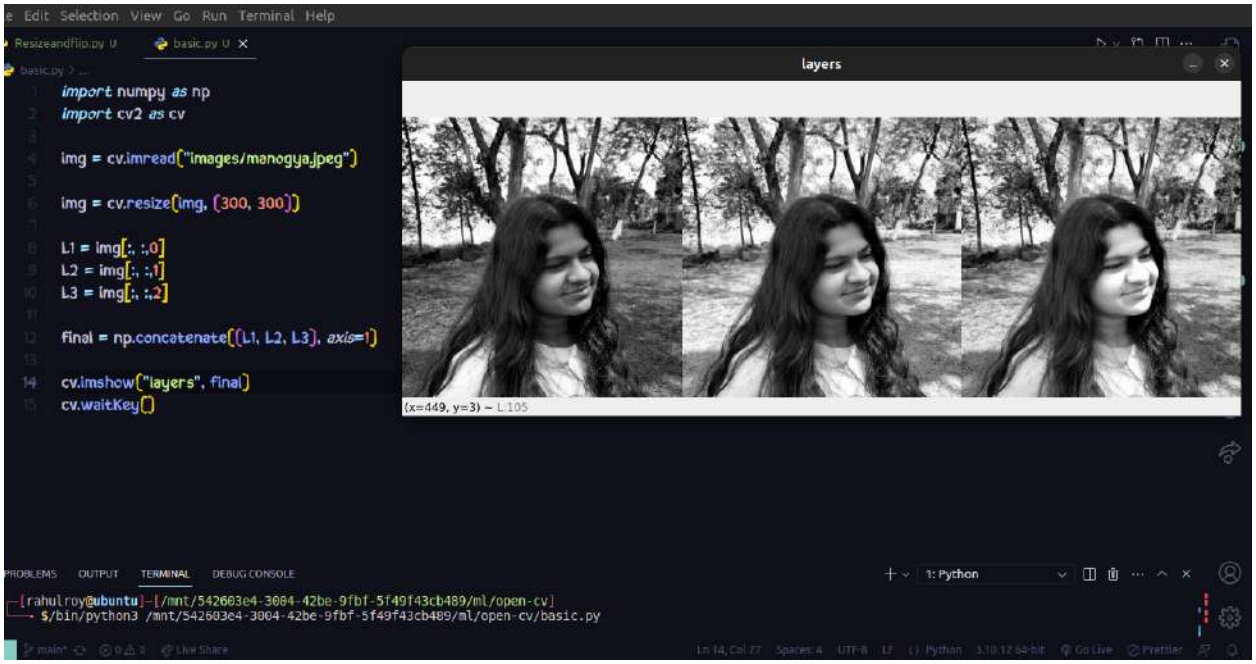
You already know the workings of **cv.imread()** and We study the **cv.resize()** function in the next section.

So, now let's talk about L1, and what it stores.

```
L1 = img[:, :,0]
```

Here, basic slicing is performed, we select the whole 2D array of the 1st layer from the 3D array. Ans same with L2 and L2

Output



What if, we set the values of any one layer to 0

Let's see it

Source code

```
import numpy as np

import cv2 as cv

img = cv.imread("images/manogya.jpeg")

img = cv.resize(img, (300, 300))

img[:, :,0] = 0

cv.imshow("layers", img)

cv.waitKey()
```

Output

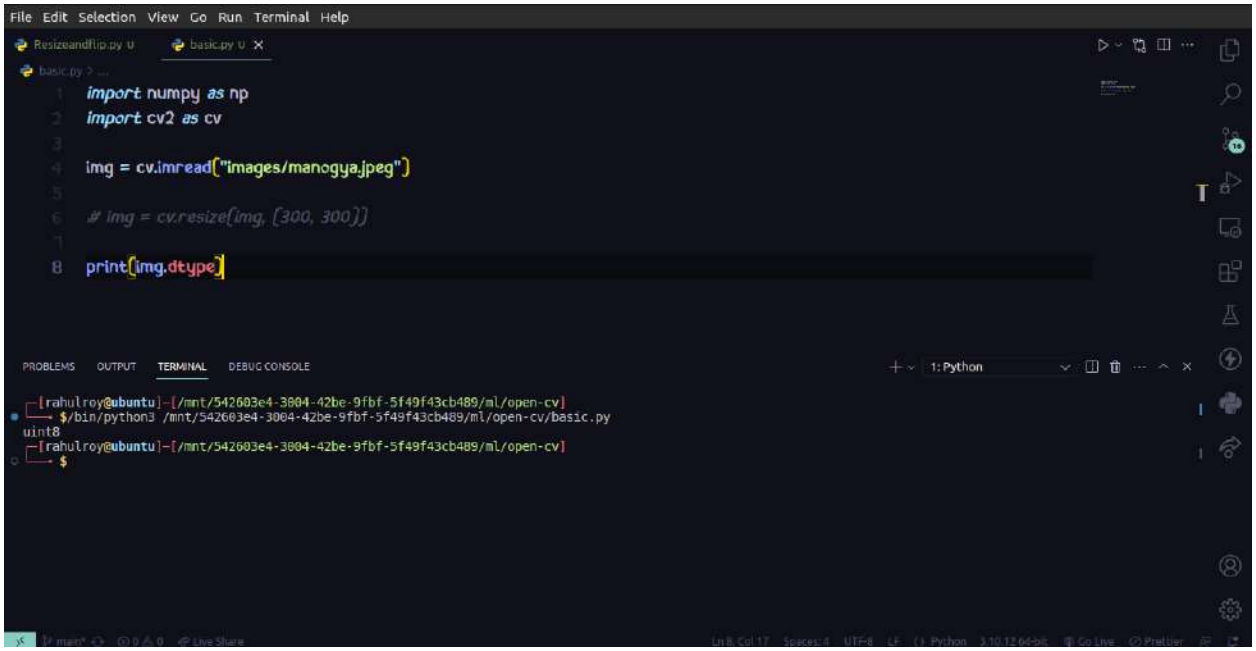
Try by yourself what you will get when the values of the second or third layer are 0.

Now Image Data Type, To discover the image's type, use the "dtype" technique. This strategy enables us to comprehend the representation of visual data and the pixel value.

Source code

```
import numpy as np

import cv2 as cv

img = cv.imread("images/manogya.jpeg")

print(img.dtype)
```

Output



notes

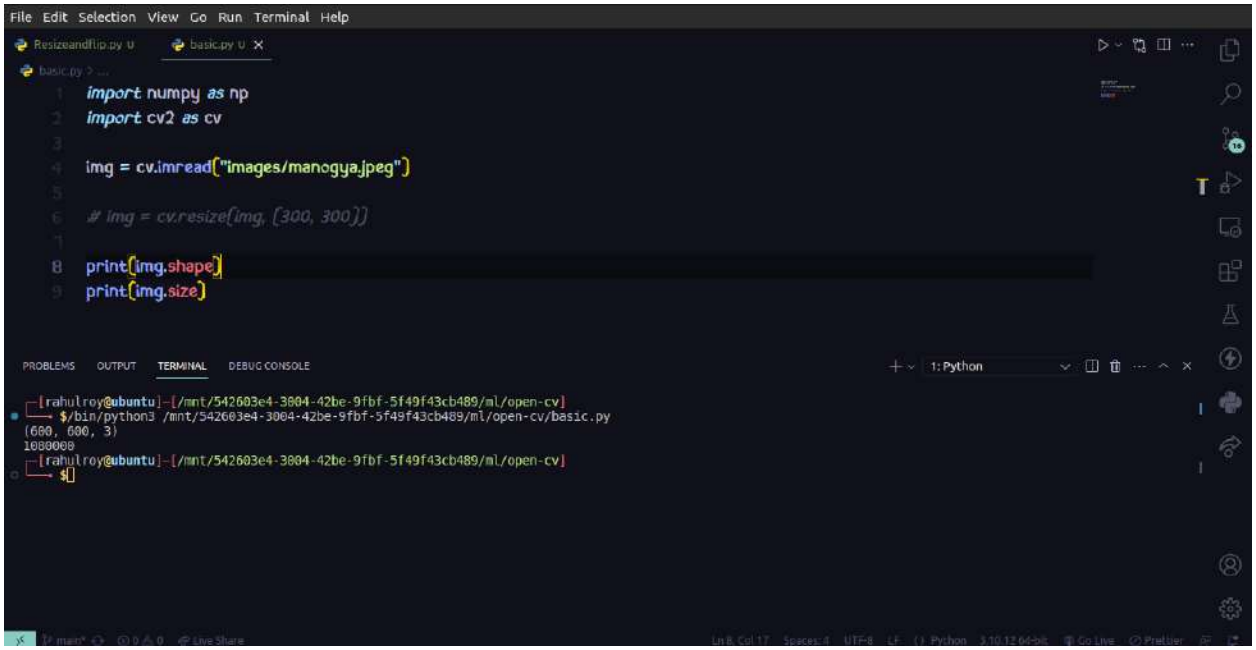
In addition to the image kind, It's a multidimensional container for things of comparable shape and size.

Source code

```
import numpy as np

import cv2 as cv

img = cv.imread("images/manogya.jpeg")

print(img.shape)

print(img.size)
```

We already know that the image was read in the form of an array. So we can get the size and shape of the array using .shape and .size functions.

Output



Resize and Flipping an input image

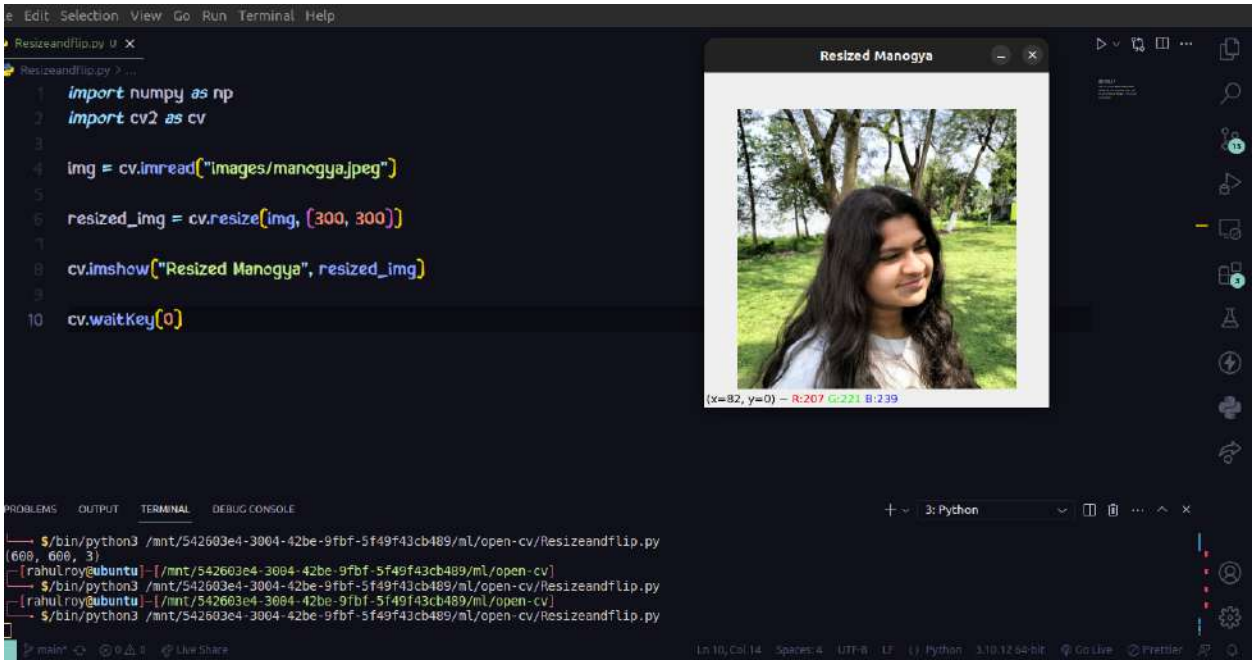
- Resizing an image(using **cv.resize()**)
- Flipping an image(using **cv.flip()**)

In the **cv.resize()** function, the first argument will tack the actual image, and the second argument will take the new size(resolution) of the image.

Source code

```
import numpy as np

import cv2 as cv

img = cv.imread("images/manogya.jpeg")

resized_img = cv.resize(img, (300, 300))

cv.imshow("Resized Manogya", resized_img)

cv.waitKey(0)
```

Output



In the **cv.flip()** function, the first argument will tack the actual image, and the second argument will take 0, 1, and -1.

Source Code

```
import numpy as np

import cv2 as cv

img = cv.imread("images/manogya.jpeg")

fliped_img = cv.flip(img, 0)

cv.imshow("Resized Manogya", fliped_img)

cv.waitKey(0)
```

0 is used for top-down flip

```
fliped_img = cv.flip(img, 0)
```

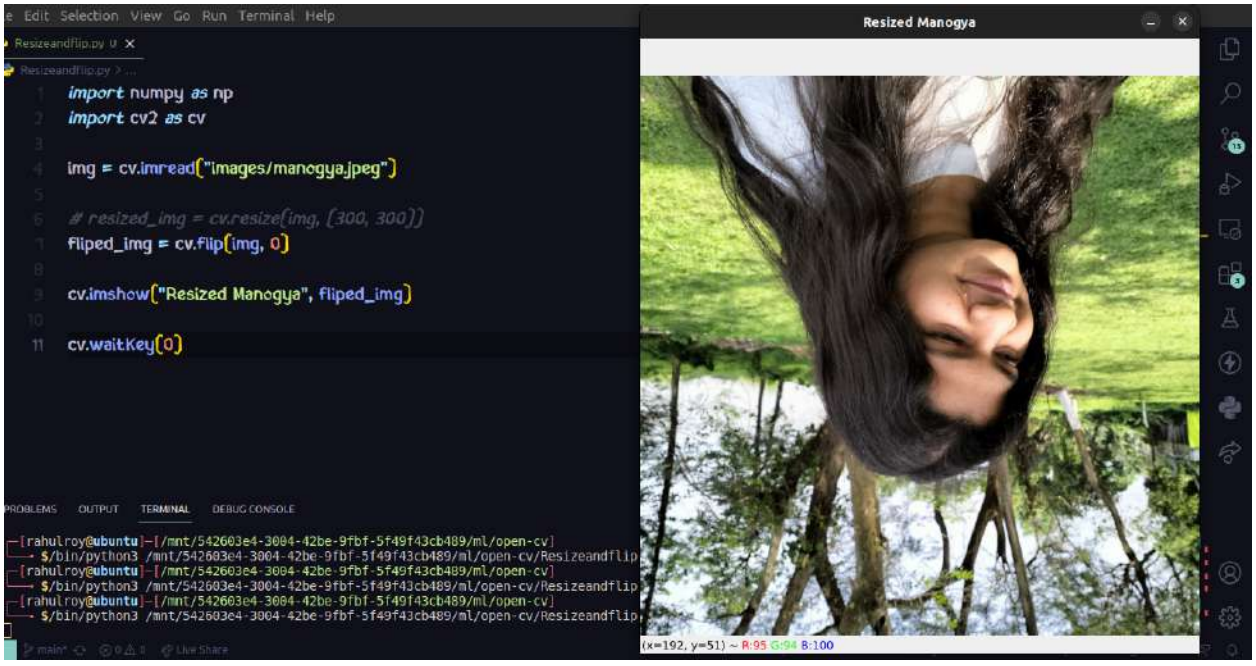
1 is used for right-left flip

```
fliped_img = cv.flip(img, 1)
```

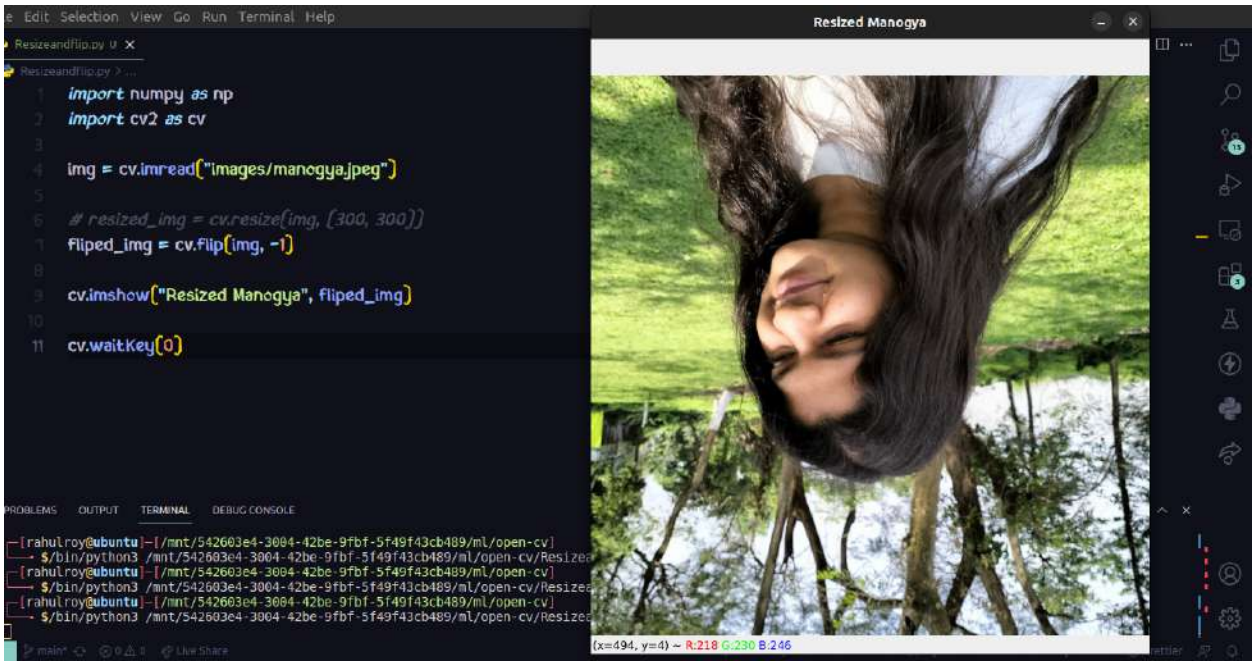
-1 is used for top-down and right-left both

```
fliped_img = cv.flip(img, -1)
```

Outputs




Edit Selection View Go Run Terminal Help
ResizeandFlip.py U
ResizeandFlip.py > fliped_img
import numpy as np
import cv2 as cv
img = cv.imread("images/manogya.jpeg")
# resized_img = cv.resize(img, (300, 300))
fliped_img = cv.flip(img, 1)
cv.imshow("Resized Manogya", fliped_img)
cv.waitKey(0)
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
[rahulroy@ubuntu]-[/mnt/542603e4-3004-42be-9fbf-5f49f43cb489/ml/open-cv]
$/bin/python3 /mnt/542603e4-3004-42be-9fbf-5f49f43cb489/ml/open-cv/ResizeandFlip.py
Resized Manogya


Edit Selection View Go Run Terminal Help
ResizeandFlip.py U
ResizeandFlip.py > ...
import numpy as np
import cv2 as cv
img = cv.imread("images/manogya.jpeg")
# resized_img = cv.resize(img, (300, 300))
fliped_img = cv.flip(img, -1)
cv.imshow("Resized Manogya", fliped_img)
cv.waitKey(0)
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
[rahulroy@ubuntu]-[/mnt/542603e4-3004-42be-9fbf-5f49f43cb489/ml/open-cv]
$/bin/python3 /mnt/542603e4-3004-42be-9fbf-5f49f43cb489/ml/open-cv/ResizeandFlip.py
Resized Manogya
(x=494, y=4) ~ R:218 G:230 B:246

Cropping an input image

Source code

```
import numpy as np

import cv2 as cv

img =cv.imread("images/manogya.jpeg")

crop = img[300:450, 200:500]

cv.imshow("croped manogya", crop)

cv.waitKey(0)
```

Output

Let's play with color channels

We know that OpenCV reads images in BGR format. What if, we convert it into RGB, let's see it 😉....

For this, we use the **cv.cvtColor()** function. "cvt" means convert and I hope you know the meaning of color 😂.

You can learn about these functions from the **OpenCV cvt function**

Source code

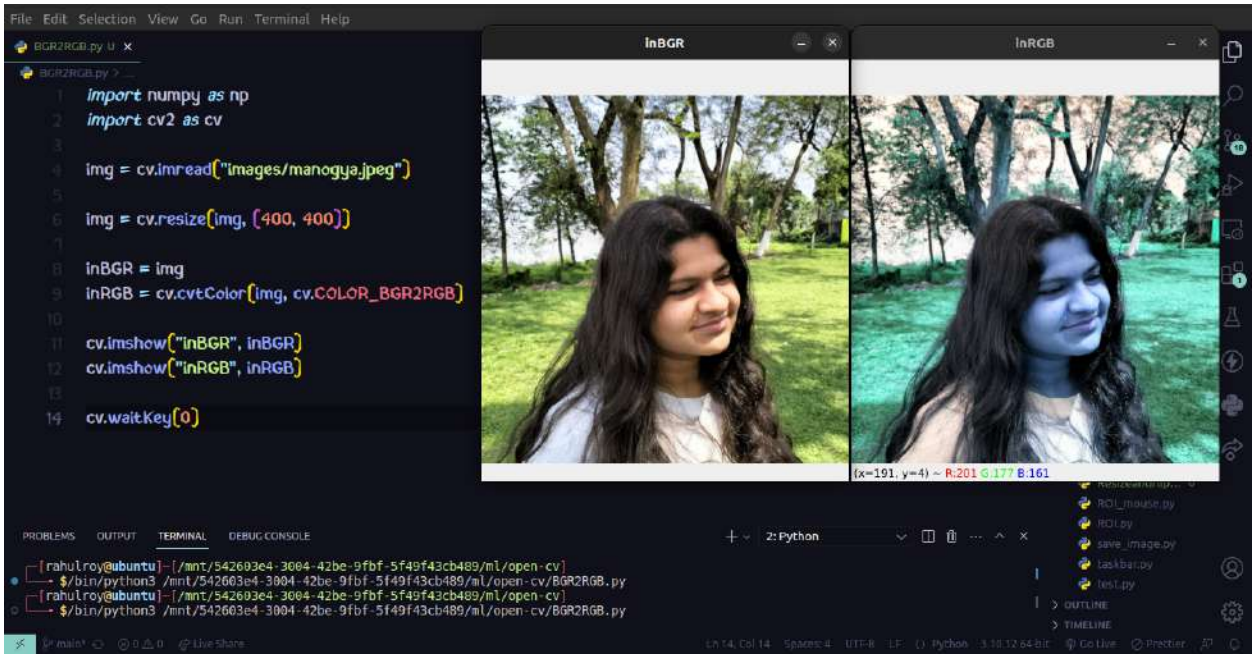
```
import numpy as np

import cv2 as cv

img = cv.imread("images/manogya.jpeg")

img = cv.resize(img, (400, 400))

inBGR = img

inRGB = cv.cvtColor(img, cv.COLOR_BGR2RGB)

cv.imshow("inBGR", inBGR)

cv.imshow("inRGB", inRGB)

cv.waitKey(0)
```

Explanation

```
cv.cvtColor(img, cv.COLOR_BGR2RGB)
```

Here, the **cv.cvtColor()** will take two parameters, the first one is the image and the second one is the format, to which color we need to convert.

Output

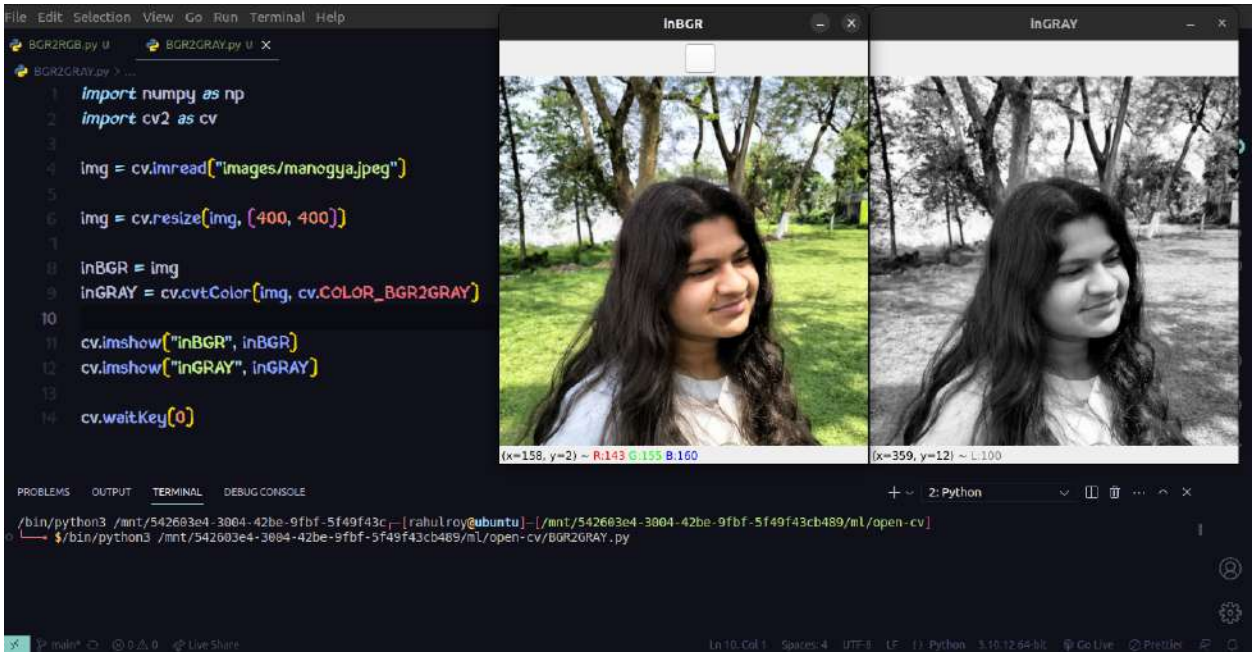


Let's convert BGR to Gray

Source code

```
import numpy as np

import cv2 as cv

img = cv.imread("images/manogya.jpeg")

img = cv.resize(img, (400, 400))

inBGR = img

inGRAY = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

cv.imshow("inBGR", inBGR)

cv.imshow("inGRAY", inGRAY)

cv.waitKey(0)
```

Output



BGR to HSV(Hue, Saturation, Value)

Source code

```
import numpy as np

import cv2 as cv

img = cv.imread("images/manogya.jpeg")

img = cv.resize(img, (400, 400))

inBGR = img

inHSV = cv.cvtColor(img, cv.COLOR_BGR2HSV)

cv.imshow("inBGR", inBGR)

cv.imshow("inHSV", inHSV)

cv.waitKey(0)
```

Output

BGR to LAB

Source code

```
import numpy as np

import cv2 as cv

img = cv.imread("images/manogya.jpeg")

img = cv.resize(img, (400, 400))

inBGR = img

inLAB = cv.cvtColor(img, cv.COLOR_BGR2LAB)

cv.imshow("inBGR", inBGR)

cv.imshow("inLAB", inLAB)

cv.waitKey(0)
```

Output

Drawing Functions

- Simple line(using **cv.line()**)
- Circle (using **cv.circle()**)
- Rectangle(using **cv.rectangle()**)
- Ellipse (using **cv.ellipse()**)
- Print text (using **cv.putText()**)

In all the above functions, you will see some common arguments as given below:

- img : The image where you want to draw the shapes
- color : Color of the shape. for BGR, pass it as a tuple, eg: (255,0,0) for blue. For grayscale, just pass the scalar value.
- thickness: Thickness of the line or circle etc. If **-1** is passed for closed figures like circles, it will fill the shape. *default thickness = 1*
- lineType: Type of line, whether 8-connected, anti-aliased line, etc. *By default, it is 8-connected.* **cv.LINE_AA** gives an anti-aliased line which looks great for curves.

Drawing Line

To draw a line, you need to pass the starting and ending coordinates of the line. We will create a black image and draw a blue line on it from top-left to bottom-right corners.

Source code

```
import numpy as np

import cv2 as cv

img = np.zeros((512, 512, 3), np.uint8)

cv.line(img,(1,256),(511, 256),(0,0,255),5)

cv.imshow("img", img)

cv.waitKey(0)
```

Explanation

```
img = np.zeros((512, 512, 3), np.uint8)
```

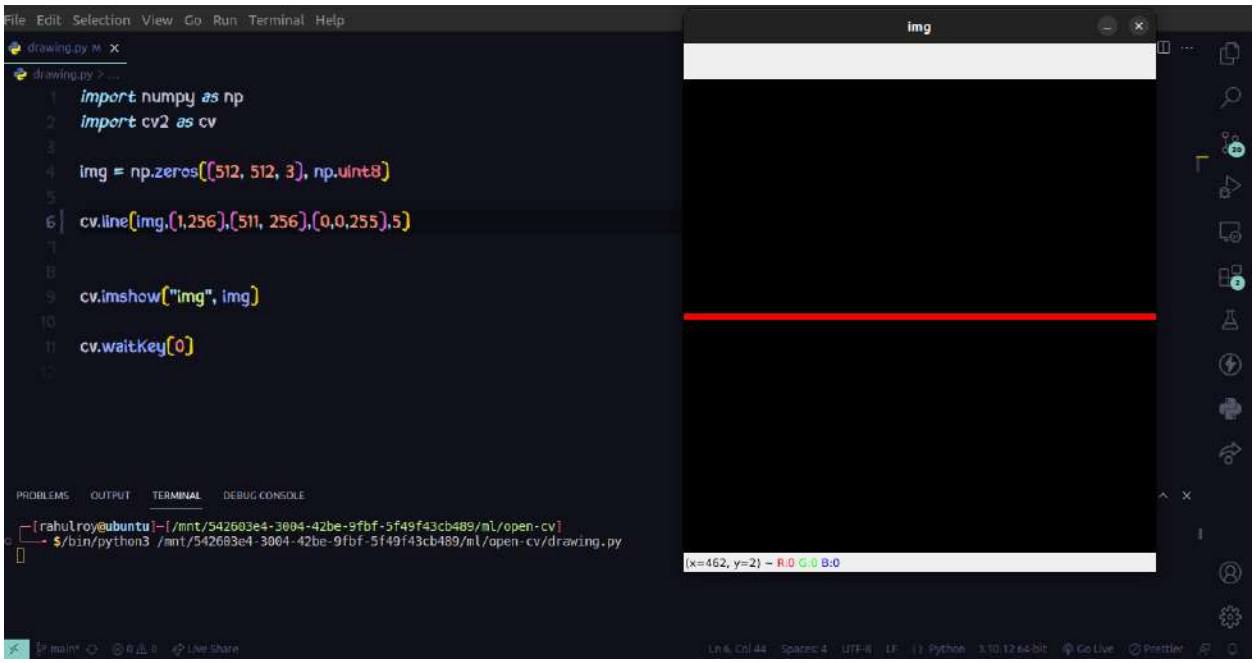
Here, numpy is used for creating the array. Now the **np.zeros()** function is used for creating an array that contains only zero. This gives a black image it gives a 512*512 px black image and 3 is used for the RGB value.

```
cv.line(img,(1,256),(511, 256),(0,0,255),5)
```

- 1st param ===> is used for giving the image
- 2nd param ===> is the starting index, from where the line will start
- 3rd param ===> it is the end index

- 4th param ===> is the color of the line
- 5th param ===> it takes the, how thick the line is

Output



Drawing Circle

To draw a circle, you need its center coordinates and radius. We will draw a circle inside the line drawn above.

```
cv.circle(img,(256,256), 63, (0,0,255), -1)
```

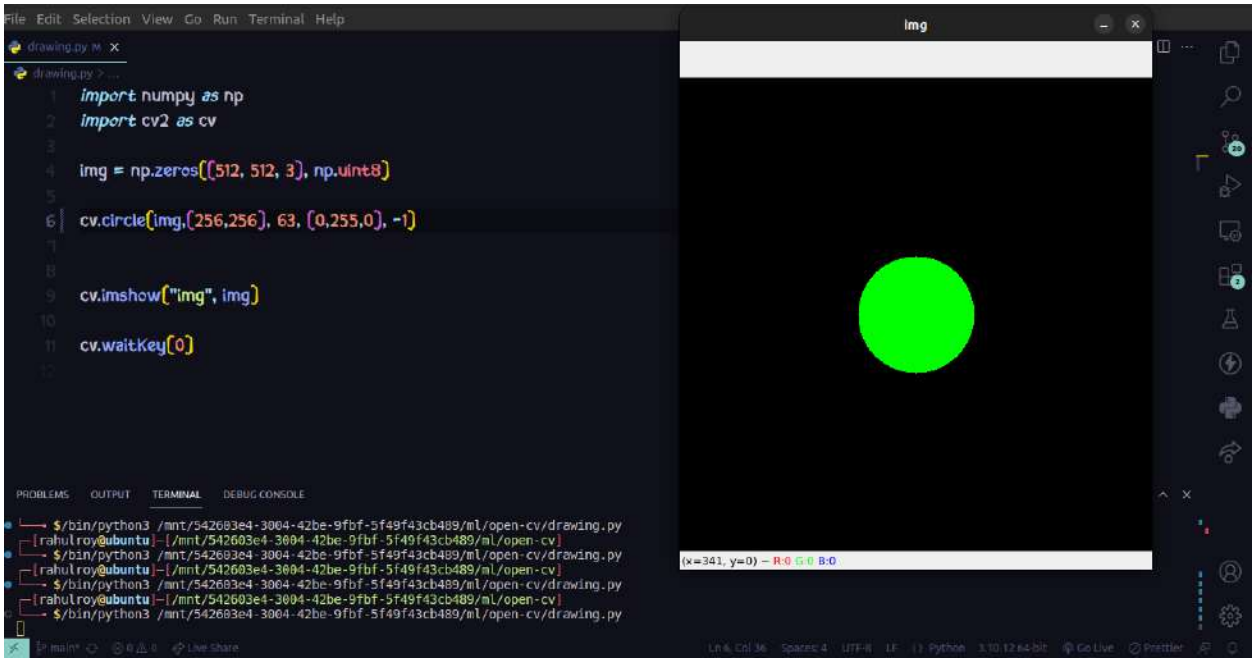
Here, (256, 256) is the center of the circle and 63 is the radius of the circle in pixels.

Source code

```
import numpy as np

import cv2 as cv

img = np.zeros((512, 512, 3), np.uint8)

cv.circle(img,(256,256), 63, (0,0,255), -1)

cv.imshow("img", img)
```

```
cv.waitKey(0)
```

Output



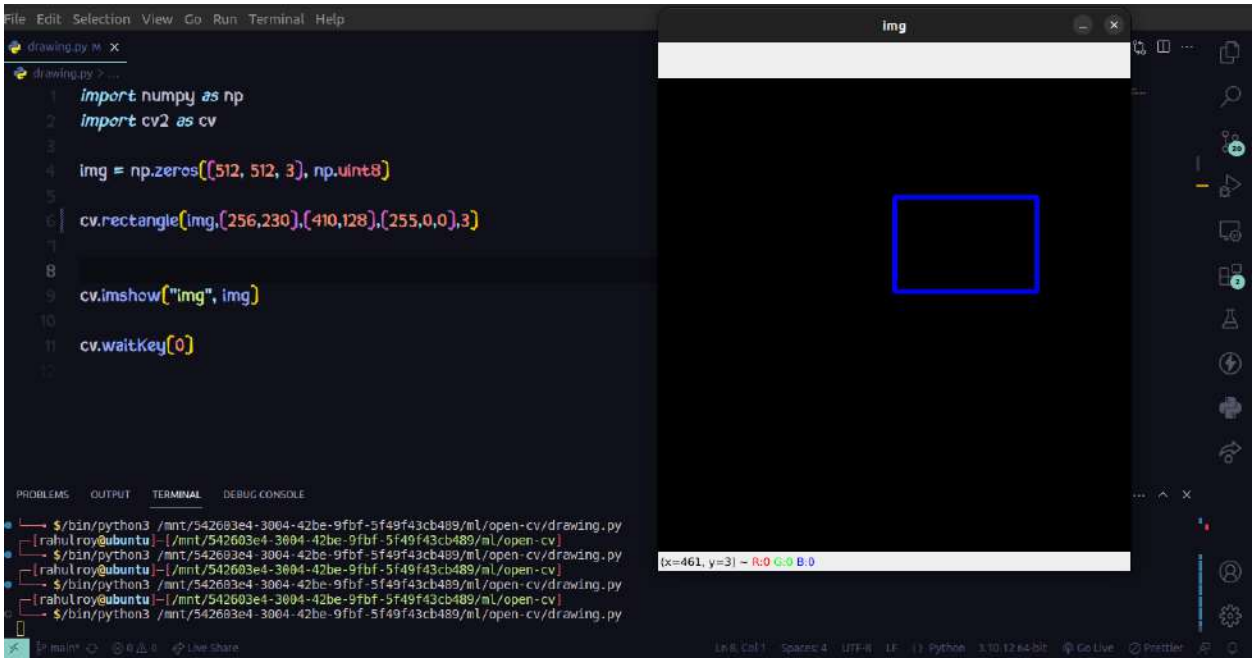
Drawing Rectangle

To draw a rectangle, you need the top-left corner and bottom-right corner of the rectangle. This time we will draw a green rectangle at the top-right corner of the image.

Source Code

```
import numpy as np

import cv2 as cv

img = np.zeros((512, 512, 3), np.uint8)

cv.rectangle(img,(256,230),(410,128),(255,0,0),3)

cv.imshow("img", img)

cv.waitKey(0)
```

Output



Drawing Ellipse

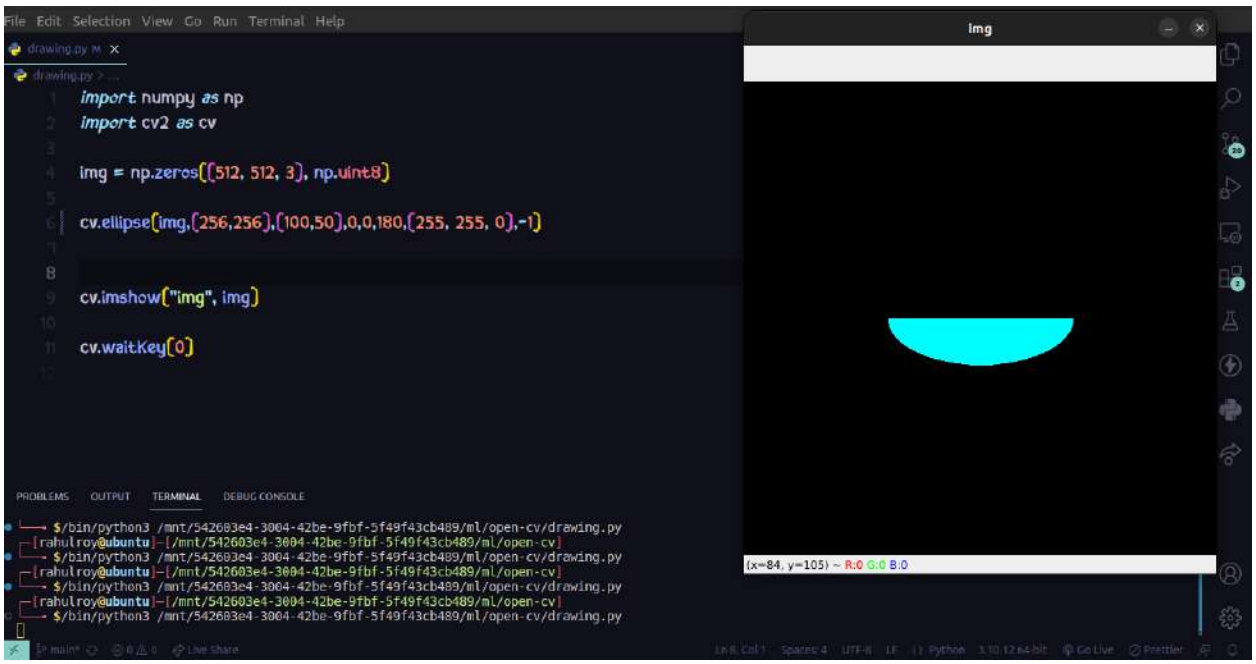
To draw the ellipse, we need to pass several arguments. One argument is the center location (x,y). The next argument is axe lengths (major axis length, minor axis length). angle is the angle of rotation of the ellipse in the anti-clockwise direction. startAngle and endAngle denote the starting and ending of the ellipse arc measured in a clockwise direction from the major axis. i.e. giving values 0 and 360 gives the full ellipse. For more details, check the documentation of **cv.ellipse()**. The below example draws a half ellipse at the center of the image.

Source code

```
import numpy as np

import cv2 as cv

img = np.zeros((512, 512, 3), np.uint8)

cv.ellipse(img,(256,256),(100,50),0,0,180,255,-1)
```

```
cv.imshow("img", img)

cv.waitKey(0)
```

Output



Drawing Polygon

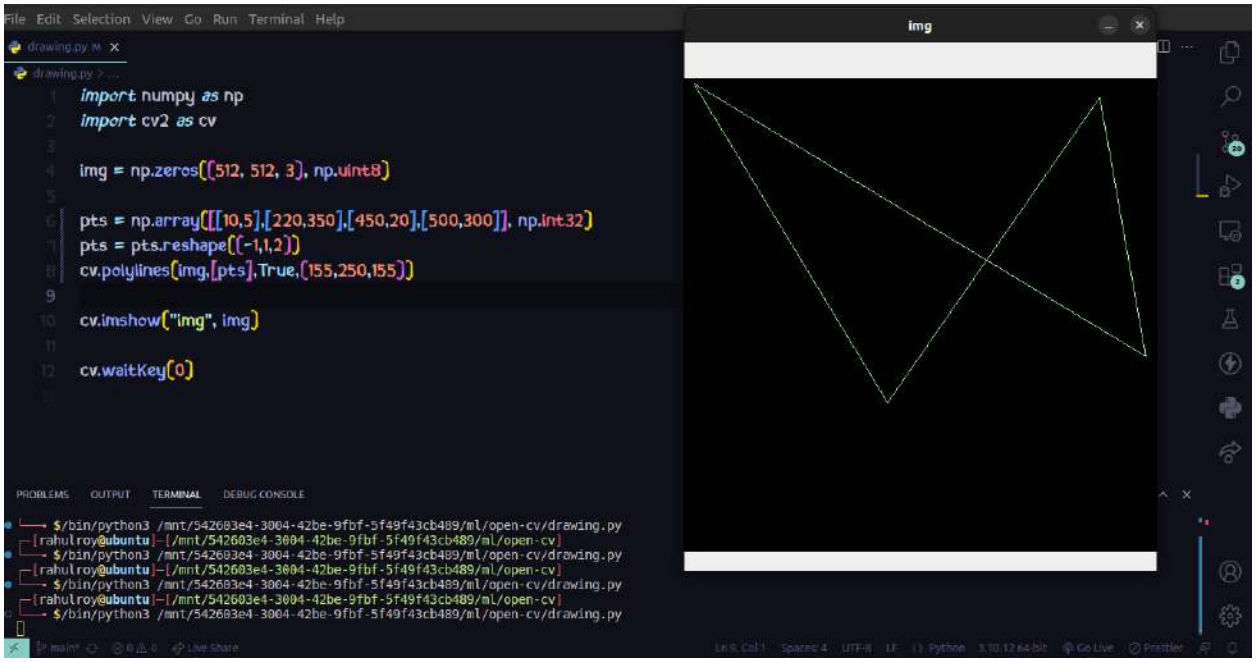
To draw a polygon, first, you need the coordinates of vertices. Make those points into an array of shapes ROWSx1x2 where ROWS are a number of vertices and it should be of type int32. Here we draw a small polygon with four vertices in yellow color.

Source code

```
import numpy as np

import cv2 as cv
```

```
img = np.zeros((512, 512, 3), np.uint8)

pts = np.array([[10,5],[220,350],[450,20],[500,300]], np.int32)

pts = pts.reshape((-1,1,2))

cv.polylines(img,[pts],True,(155,250,155))

cv.imshow("img", img)

cv.waitKey(0)
```

Output



Adding Text to Images

To put texts in images, you need to specify the following things.

- Text data that you want to write
- Position coordinates of where you want to put it (i.e. bottom-left corner where data starts).
- Font type (Check **cv.putText()** docs for supported fonts)

- Font Scale (specifies the size of the font)
- regular things like color, thickness, line type, etc. For a better look, lineType = **cv.LINE_AA** is recommended.

Source code

```
import numpy as np

import cv2 as cv

img = np.zeros((512, 512, 3), np.uint8)

font = cv.FONT_HERSHEY_SIMPLEX

cv.putText(img,'Rahul Ray',(100,256), font,
2,(219,252,3),2,cv.LINE_AA)

cv.imshow("img", img)

cv.waitKey(0)
```

Output