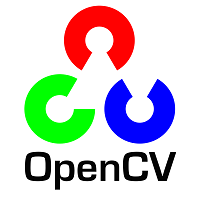
**OpenCV Tutorial | OpenCV using Python**

OpenCV tutorial provides basic and advanced concepts of OpenCV. Our OpenCV tutorial is designed for beginners and professionals.

OpenCV is an open-source library for the computer vision. It provides the facility to the machine to recognize the faces or objects. In this tutorial we will learn the concept of OpenCV using the Python programming language.

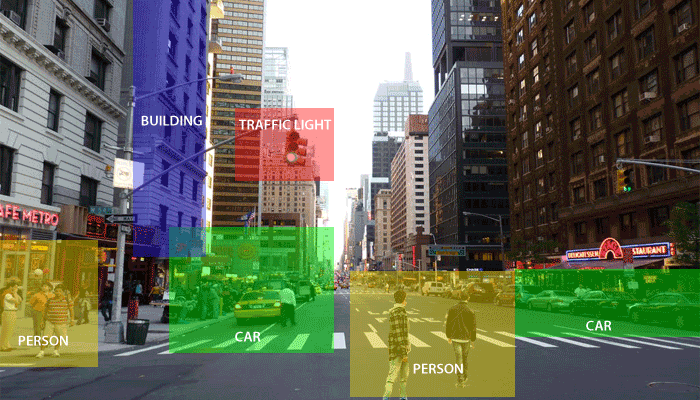
## What is OpenCV?



OpenCV is a Python open-source library, which is used for computer vision in Artificial intelligence, Machine Learning, face recognition, etc.

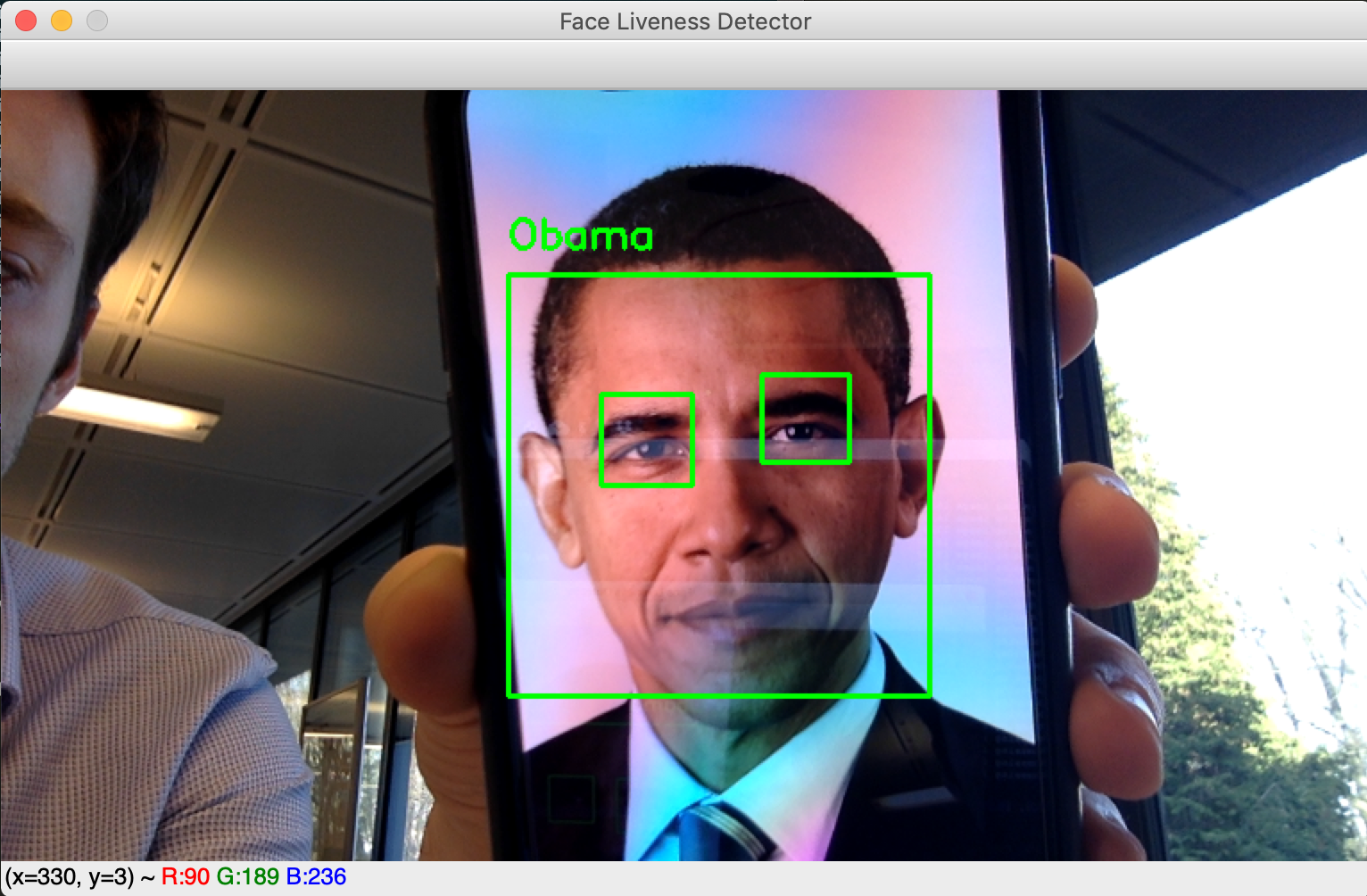
In OpenCV, the CV is an abbreviation form of a computer vision, which is defined as a field of study that helps computers to understand the content of the digital images such as photographs and videos.

The purpose of computer vision is to understand the content of the images. It extracts the description from the pictures, which may be an object, a text description, and three-dimension model, and so on. For example, cars can be facilitated with computer vision, which will be able to identify and different objects around the road, such as traffic lights, pedestrians, traffic signs, and so on, and acts accordingly.



Computer vision allows the computer to perform the same kind of tasks as humans with the same efficiency. There are a two main task which are defined below:

* **Object Classification -** In the object classification, we train a model on a dataset of particular objects, and the model classifies new objects as belonging to one or more of your training categories.
* **Object Identification -** In the object identification, our model will identify a particular instance of an object - for example, parsing a face in an image and tagging one as Obama.



source TOWARD DATA SCIENCE

## History

OpenCV stands for Open Source Computer Vision Library, which is widely used for image recognition or identification. It was officially launched in 1999 by Intel. It was written in C/C++ in the early stage, but now it is commonly used in Python for the computer vision as well.

The first alpha version of OpenCV was released for the common use at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and between 2001 and 2005, five betas were released. The first 1.0 version was released in 2006.

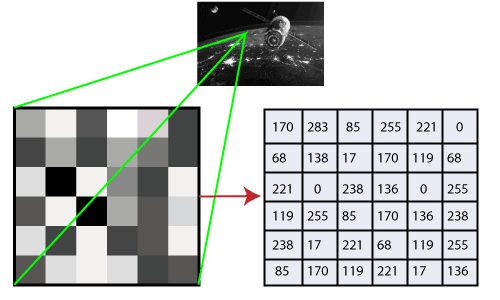
The second version of the OpenCV was released in October 2009 with the significant changes. The second version contains a major change to the C++ interface, aiming at easier, more type-safe, pattern, and better implementations. Currently, the development is done by an independent Russian team and releases its newer version in every six months.

## How OpenCV Works

In this tutorial, we will learn how computers perform image recognition.

### How does computer recognize the image?

Human eyes provide lots of information based on what they see. Machines are facilitated with seeing everything, convert the vision into numbers and store in the memory. Here the question arises how computer convert images into numbers. So the answer is that the pixel value is used to convert images into numbers. A pixel is the smallest unit of a digital image or graphics that can be displayed and represented on a digital display device.



The picture intensity at the particular location is represented by the numbers. In the above image, we have shown the pixel values for a grayscale image consist of only one value, the intensity of the black color at that location.

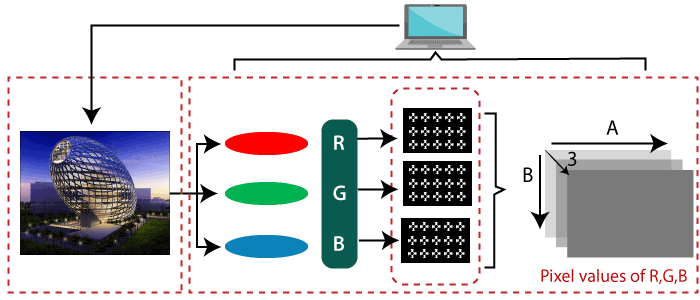
There are two common ways to identify the images:

**1. Grayscale**

Grayscale images are those images which contain only two colors black and white. The contrast measurement of intensity is black treated as the weakest intensity, and white as the strongest intensity. When we use the grayscale image, the computer assigns each pixel value based on its level of darkness.

**2. RGB**

An RGB is a combination of the red, green, blue color which together makes a new color. The computer retrieves that value from each pixel and puts the results in an array to be interpreted.



## Why OpenCV is used for Computer Vision?

* OpenCV is available for free of cost.
* Since the OpenCV library is written in C/C++, so it is quit fast. Now it can be used with Python.
* It require less RAM to usage, it maybe of 60-70 MB.
* Computer Vision is portable as OpenCV and can run on any device that can run on C.

## Prerequisite

Before learning OpenCV, you must have the basic knowledge of Python programming language.

# Installation of the OpenCV

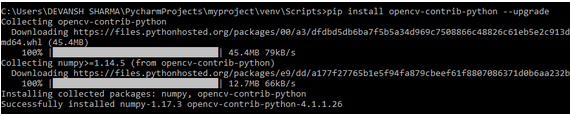
### Install OpenCV in the Windows via pip

OpenCV is a Python library so it is necessary to install Python in the system and install OpenCV using pip command:

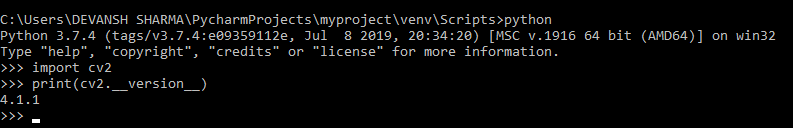
* pip install opencv-contrib-python --upgrade

We can install it without extra modules by the following command:

* pip install opencv-python



Open the command prompt and type the following code to check if the OpenCV is installed or not.



# OpenCV Read and Save Image

## OpenCV Reading Images

OpenCV allows us to perform multiple operations on the image, but to do that it is necessary to read an image file as input, and then we can perform the various operations on it. OpenCV provides following functions which are used to read and write the images.

### OpenCV imread function

The imread() function loads image from the specified file and returns it. The syntax is:

* cv2.imread(filename[,flag])

### Parameters:

**filename:** Name of the file to be loaded

**flag:** The flag specifies the color type of a loaded image:

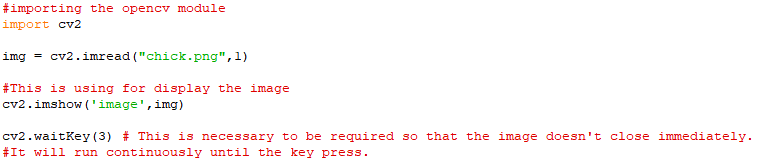
* **CV\_LOAD\_IMAGE\_ANYDEPTH -** If we set it as flag, it will return 16-bits/32-bits image when the input has the corresponding depth, otherwise convert it to 8-BIT.
* **CV\_LOAD\_IMAGE\_COLOR -** If we set it as flag, it always return the converted image to the color one.
* **C V\_LOAD\_IMAGE\_GRAYSCALE -** If we set it as flag, it always convert image into the grayscale.

The **imread()** function returns a matrix, if the image cannot be read because of unsupported file format, missing file, unsupported or invalid format. Currently, the following file formats are supported.

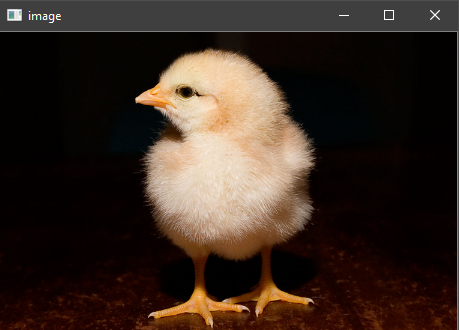
**Window bitmaps** - \*.bmp, \*.dib  
**JPEG files** - \*.jpeg, \*.jpg, \*.jpe  
**Portable Network Graphics** - \*.png  
**Portable image format**- \*.pbm, \*.pgm, \*.ppm  
**TIFF files** - \*.tiff, \*.tif

#### Note: The color images, the decoded images will have the channels stored in the BGR order.

Let' s consider the following example:



**Output:** it will display the following image.



## OpenCV Save Images

OpenCV **imwrite()** function is used to save an image to a specified file. The file extension defines the image format. The syntax is the following:

* cv2.imwrite(filename, img[,params])

### Parameters:

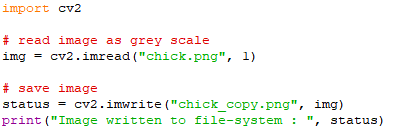
**filename-** Name of the file to be loaded

**image-** Image to be saved.

**params-** The following parameters are currently supported:

* For JPEG, quality can be from 0 to 100. The default value is 95.
* For PNG, quality can be the compress level from 0 to 9. The default value is 1.
* For PPM, PGM, or PBM, it can be a binary format flag 0 or 1. The default value is 1.

Let's consider the following example:



**Output:**



If the imwrite() function returns the True, which means the file is successfully written in the specified file.

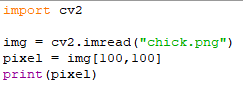
**OpenCV Basic Operation on Images**

In this tutorial, we will learn the essential operations that are related to the images. We are going to discuss the following topics.

* Access pixel values and modify them
* Access Image Properties
* Setting Region of Image
* Splitting and merging images
* Change the image color

## Accessing and Modifying pixel values

We can retrieve a pixel value by its row and column coordinates. It returns an array of blue, green, red values of the BGR image. It returns the corresponding intensity for the grayscale image. First, we need to load the BGR image.

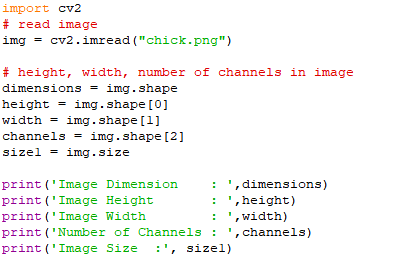


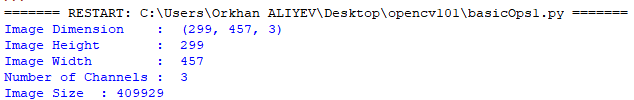


## Accessing Image Properties

It is better to know the size of the image to work with the image processing application. In OpenCV, images are generally stored in the Numpy ndarray. To get the image shape or size, use ndarray.shape to find the dimension of the image. Then, we can use the index position to get the height, width, and number of channels.

Consider the following example:





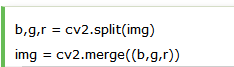
## Image ROI (Region of Interest)

Sometimes, we need to work with some areas of the image. As we discuss in the previous tutorial face detection is over the entire picture. When a face is obtained, we select only the face region and search for eyes inside it instead of searching the whole image. It enhances accuracy and performance because eyes are always on the face and don't need to search the entire image.



## Splitting and Merging Image channels

An image's BGR channels can be split into their planes when needed. Then, the individual channels can be merged back together from the BGR image again. This can be done by following way:



or



#### Note: The cv2.split() function is a slow function. Numpy indexing is quit efficient and it should be used if possible.

## Change in Image color

### OpenCV cvtColor

The **cvtColor** is used to convert an image from one color space to another. The syntax is following:

* cv2.cvtColor(src, dst, code)

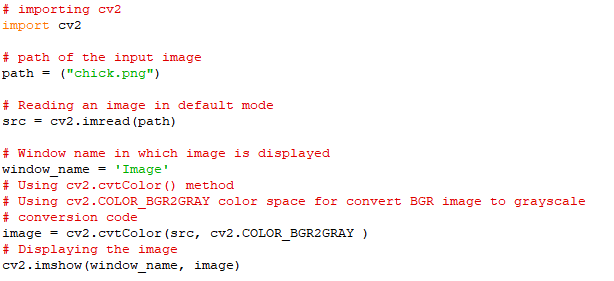
### Parameters:

**src -** It is used to input an image: 8-bit unsigned.

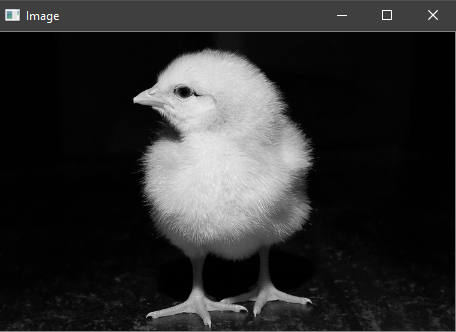
**dst -** It is used to display an image as output. The output image will be same size and depth as input image.

**code -** color space conversion code.

Consider the following example:



**Output:**



# OpenCV Resize the image

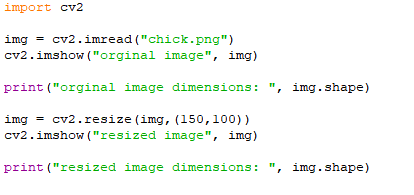
Sometimes, it is necessary to transform the loaded image. In the image processing, we need to resize the image to perform the particular operation. Images are generally stored in Numpy ndarray(array). The **ndarray.shape** is used to obtain the dimension of the image. We can get the width, height, and numbers of the channels for each pixel by using the index of the dimension variable.

The resizing of image means changing the dimension of the image, its width or height as well as both. Also the aspect ratio of the original image could be retained by resizing an image. OpenCV provides **cv2.resize()** function to resize the image. The syntax is given as:

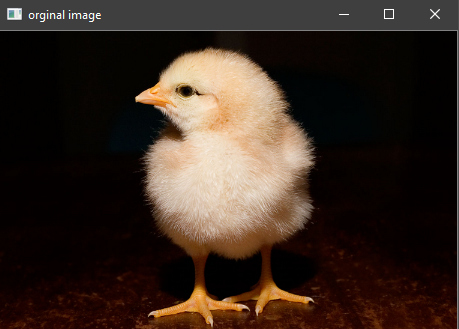
* cv2.resize(src, dsize[, dst[, fx[,fy[,interpolation]]])

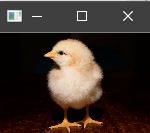
### Parameters:

* **src** - source/input image (required).
* **dsize** - desired size for the output image(required)
* **fx** - Scale factor along the horizontal axis.(optional)
* **fy** - Scale factor along the vertical axis.
* **Interpolation(optional)** - This flag uses following methods:
  + INTER\_NEAREST - A nearest-interpolation INTER\_AREA - resampling using pixel area relation. When we attempt to do image zoom, it is similar to the INTER\_NEAREST method.
  + INTER\_CUBIC - A bicubic interpolation over 4×4 pixel neighborhood.
  + INTER\_LANCOZS4 - Lanczos interpolation over 8×8 pixel neighborhood.



**Output:**







# OpenCV Drawing Functions

We can draw the various shapes on an image such as **circle, rectangle, ellipse, polylines, convex,** etc. It is used when we want to highlight any object in the input image. The OpenCV provides functions for each shape. Here we will learn about the drawing functions.

## Drawing Circle

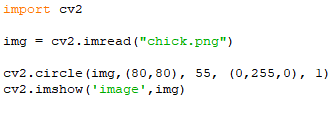
We can draw the circle on the image by using the **cv2.circle()** function. The syntax is the following:

* cv2.circle(img, center, radius, color[,thickness [, lineType[,shift]]])

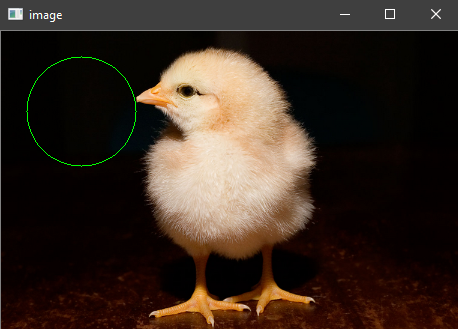
### Parameters:

* **img-** It represents the given image.
* **center-** Center of the circle
* **radius-** Radius of the circle
* **color-** Circle color
* **thickness-** It denotes the thickness of the circle outline, if it is positive. And negative thickness means that a filled circle is to be drawn.
* **lineType-** Defines the type of the circle boundary.
* **shift-** It represents the number of fractional bits in the coordinate of the center and the radius value.

Consider the following example:



**Output:**



## Drawing Rectangle

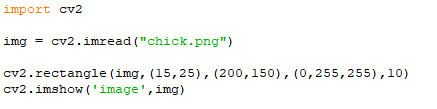
The OpenCV provides a function to draw a simple, thick or filled up-right rectangle. The syntax is following:

* cv2.rectangle(img, pt1, pt2, color[, thickness[,lineType[,shift]]])

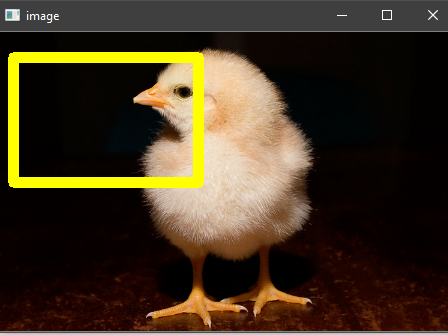
### Parameters:

* **img-** It represents an image.
* **pt1-** It denotes **vertex** of the rectangle.
* **pt2-** It denotes the vertex of the rectangle opposite to pt1.
* **color-** It denotes the rectangle color of brightness (grayscale image).
* **thickness-** It represents the thickness of the lines that makes up the rectangle. Negative values (CV\_FILLED) mean that the function has to draw a filled rectangle.
* **linetype-** It represents the types of the line.
* **shift-** It represents the number of fractional bits in the point coordinates.

Consider the following example:



**Output:**



## Drawing Ellipse

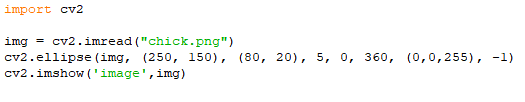
We can draw an ellipse on an image by using the **cv2.ellipse()** function. It can draw a simple or thick elliptic arc or can fill an ellipse sector.

1. cv2.ellipse(img, center, axes, angle, startAngle, endAngle, color[, thickness[, lineType[, shift]]])
2. cv2.ellipse(img, box, color[, thickness[, lineType]])

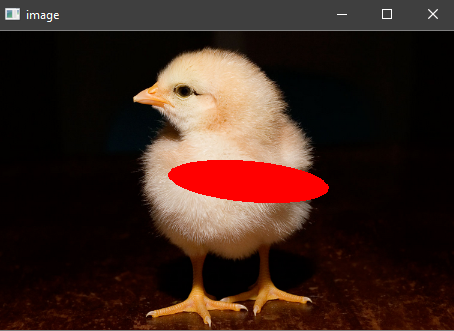
### Parameters:

* **img -** It represents an image.
* **box -** It represents alternative ellipse representation via RotatedRect or CvBox2D. It means that the function is used to draws an ellipse in a curved rectangle.
* **color -** It denotes the ellipse color.
* **angle-** It denotes the angle of rotation.
* **startAngle -** It denotes the initial angle of the elliptic arc in degrees.
* **endAngle -** It denotes the ending angle of the elliptic arc in degrees.
* **thickness -** It is used to draw thickness of the ellipse arc outline if the value is positive. Otherwise, this specifies that a filled ellipse is to be drawn.
* **lineType -** It denotes the type of the ellipse boundary.
* **shift -** It represents the number of fractional bits in the coordinates of the center and values of axes.

Consider the following example:



**Output:**



There are two functions to draw the ellipse. The first function is used to draw the whole ellipse, not an arc bypassing **startAngle=0** and **endAngle = 360**. The second function of an ellipse is used to draw an ellipse outline, a filled ellipse, an elliptic arc, or a filled ellipse sector.

## Drawing lines

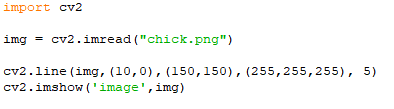
OpenCV provides the **line()** function to draw the line on the image. It draws a line segment between ptr1 and ptr2 points in the image. The image boundary clips the line.

* cv2.line(img, pt1, pt2, color[, thickness[, lineType[, shift]]])

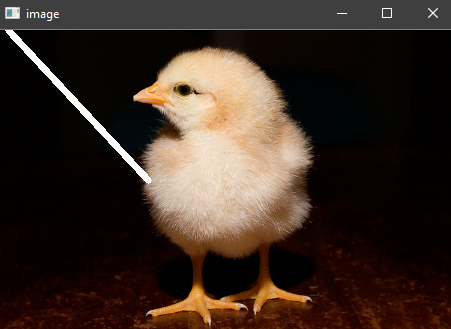
### Parameters:

* **img-** It represents an image.
* **pt1-** It denotes the first point of the line segments.
* **pt2-** It denotes the second point of the line segment.
* **color -** Represents the Line-color
* **thickness-** Represents the Line thickness
* **lineType-** There are various types of line:
  + 8 (or omitted) - 8 connected lines.
  + 4 - 4-connected line.
  + CV\_\_AA- antialiased line
* **shift-** It represents the number of fractional bits in the point coordinates.

Consider the following example:



**Output:**



### Write Text on Image

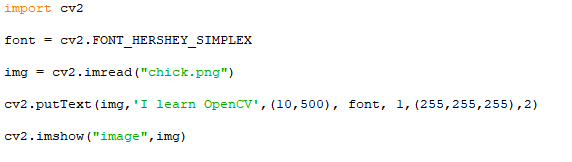
We can write text on the image by using the **putText()** function. The syntax is given below.

* cv2.putText(img, text, org, font, color)

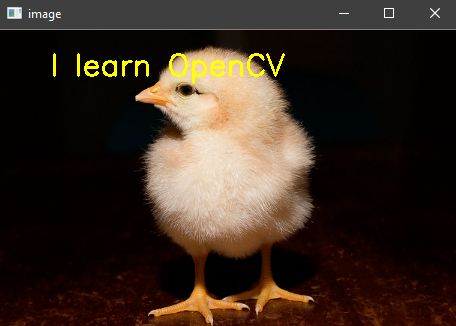
### Parameters:

* **img:** It represents an image
* **text:** It represents a text which we want to write on the image.
* **org:** It denotes the Bottom-left corner of the text string in the image.
* **font:** CvFont structure is initialized using InitFont().
* **color:** Represents the Text color.

Consider the following example.



**Output:**



## Drawing Polylines

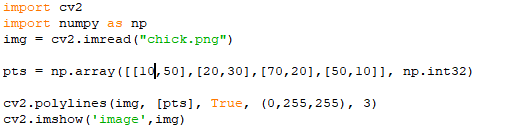
We can draw the polylines on the image. OpenCV provides the **polylines()** function, that is used to draw polygonal curves on the image. The syntax is given below:

* cv2.polyLine(img, polys, is\_closed, color, thickness=1, lineType=8, shift=0)

### Parameters:

* **img -** It represents an image.
* **pts -** It denotes the array of polygon curves.
* **npts -** It denotes an array of polygon vertex counters.
* **ncounters -** It represents the number of curves.
* **is\_Closed -** It is a flag that indicates whether the drawn polylines are closed or not.
* **color -** Color of polylines.
* **thickness -** It represents the Thickness of the polylines edges.
* **lineType -** Type of the line segment.
* **shift-** It represents the number of fractional bits in the point coordinates.

Consider the following program to draw polylines in image:



**Output:**

