**1. What does RGBA stand for?**

**ANS:-**

**RGBA** stands for **red green blue alpha**. While it is sometimes described as a [color space](https://en.wikipedia.org/wiki/Color_space" \o "Color space), it is actually a three-channel [RGB color model](https://en.wikipedia.org/wiki/RGB_color_model) supplemented with a fourth *alpha channel*. Alpha indicates how opaque each pixel is and allows an image to be combined over others using [alpha compositing](https://en.wikipedia.org/wiki/Alpha_compositing), with [transparent](https://en.wikipedia.org/wiki/Transparency_(graphic)) areas and [anti-aliasing](https://en.wikipedia.org/wiki/Spatial_anti-aliasing) of the edges of opaque regions.

**2. From the Pillow module, how do you get the RGBA value of any images?**

**ANS:-**

To create a transparent png using Python3, the Pillow library is used. The Pillow library comes with python itself. If python is unable to find Pillow library then open the command prompt and run this command:-

pip install Pillow

**Note:** If you got any problem installing Pillow using pip, then install and setup pip first. For this check [this](https://www.geeksforgeeks.org/how-to-install-pip-on-windows/) article.

**Approach:**

**1. import the Image module from the Pillow library**

from PIL import Image

**2. Open any image and get the RAGBAG values.**

*img = Image.open(‘image.png’)*

*rgba = img.convert(“RGBA”)*

*datas = rgba.getdata()*

**3. Change the color**

Data will be an Imaging Core object containing thousands of tuples of RGBA values. To make transparent the background firstly we have to find the RGBA values of the background or any color we want to make transparent. Here in this image, the background color is black.



The RGB value of black is (0, 0, 0). Now we will loop through the data (RGBA values) and whenever we find a black pixel we will replace it with a transparent RGBA value which is ((255, 255, 255, 0), and the other colors will be unchanged.  And we will store the values in a new list called newData.

*newData = []*

*for item in datas:*

*if item[0] == 0 and item[1] == 0 and item[2] == 0:*

*newData.append((255, 255, 255, 0))*

*else:*

*newData.append(item)*

**4. Store the changed image**

Store the newData into RGBA value and save the image as a png format(transparent image can’t be stored into jpg or jpeg format).

*rgba.putdata(newData)*

*rgba.save(“transparent\_image.png”, “PNG”)*

**3. What is a box tuple, and how does it work?**

**ANS:-**

The box.tuple submodule provides read-only access for the tuple userdata type. It allows, for a single [tuple](https://www.tarantool.io/en/doc/latest/concepts/data_model/value_store/" \l "index-box-tuple): selective retrieval of the field contents, retrieval of information about size, iteration over all the fields, and conversion to a [Lua table](https://www.lua.org/pil/2.5.html).

Below is a list of all box.tuple functions.

| Name | Use |
| --- | --- |
| [box.tuple.new()](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/new/) | Create a tuple |
| [box.tuple.is()](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/is/) | Check whether a given object is a tuple |
| [#tuple\_object](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/count_fields/) | Count tuple fields |
| [box.tuple.bsize()](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/bsize/) | Get count of bytes in a tuple |
| [tuple\_object[field-number]](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/field_number/) | Get a tuple’s field by specifying a number |
| [tuple\_object[field-name]](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/field_name/) | Get a tuple’s field by specifying a name |
| [tuple\_object[field-path]](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/field_path/) | Get a tuple’s fields or parts by specifying a path |
| [tuple\_object:find(), tuple\_object:findall()](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/find/) | Get the number of the first field/all fields matching the search value |
| [tuple\_object:next()](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/next/) | Get the next field value from tuple |
| [tuple\_object:pairs(), tuple\_object:ipairs()](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/pairs/) | Prepare for iterating |
| [tuple\_object:totable()](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/totable/) | Get a tuple’s fields as a table |
| [tuple\_object:tomap()](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/tomap/) | Get a tuple’s fields as a table along with key:value pairs |
| [tuple\_object:transform()](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/transform/) | Remove (and replace) a tuple’s fields |
| [tuple\_object:unpack()](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/unpack/) | Get a tuple’s fields |
| [tuple\_object:update()](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/update/) | Update a tuple |
| [tuple\_object:upsert()](https://www.tarantool.io/en/doc/latest/reference/reference_lua/box_tuple/upsert/) | Update a tuple ignoring errors |

**4. Use your image and load in notebook then, How can you find out the width and height of an Image object?**

**ANS:-**

In order to find the height and width of an image, there are two approaches. The first approach is by using the**PIL(Pillow)**library and the second approach is by using the **Open-CV** library.

**Approach 1:**

*PIL* is the Python Imaging Library is an important module which is used for image processing. It supports many formats of images such as “jpeg”, “png”, “ppm”, “tiff”, “bmp”, “gif”.  It provides many image editing capabilities. The Image module provides a class with the same name which is used to represent a *PIL* image.

***PIL.Image.open()*** is used to open the image and then**.width** and **.height** property of Image are used to get the height and width of the image. The same results can be obtained by using **.size**property.

**# import required module**

**from PIL import Image**

**# get image**

**filepath = "geeksforgeeks.png"**

**img = Image.open(filepath)**

**# get width and height**

**width = img.width**

**height = img.height**

**# display width and height**

**print("The height of the image is: ", height)**

**print("The width of the image is: ", width)**

**5. What method would you call to get Image object for a 100×100 image, excluding the lower-left quarter of it?**

**ANS:-**

To get an Image object for a 100x100 image, excluding the lower-left quarter of it, you would need to use the HTML5 Canvas API to manipulate the image data. Here's an example of how you can achieve this:

1. // Create a new canvas element
2. const canvas = document.createElement('canvas');
3. canvas.width = 100;
4. canvas.height = 100;
6. // Get the canvas context and load the image
7. const ctx = canvas.getContext('2d');
8. const img = new Image();
9. img.src = 'path/to/image.png';
11. // Once the image has loaded, draw it on the canvas
12. img.onload = function() {
13. // Draw the image on the canvas
14. ctx.drawImage(img, 0, 0, 100, 100);
16. // Exclude the lower-left quarter of the image
17. ctx.clearRect(0, 75, 25, 25);
19. // Get the resulting image data
20. const imageData = ctx.getImageData(0, 0, 100, 100);
22. // Create a new Image object from the resulting image data
23. const newImg = new Image();
24. newImg.src = URL.createObjectURL(new Blob([imageData], {type: 'image/png'}));
26. // Append the new Image object to the document
27. document.body.appendChild(newImg);
28. };

In this example, we create a new canvas element with a width and height of 100 pixels. We then load the image and draw it on the canvas using the drawImage method. We then use the clearRect method to exclude the lower-left quarter of the image. Finally, we use the getImageData method to get the resulting image data and create a new Image object from it using the Blob constructor. The new Image object can then be appended to the document as needed.

**6. After making changes to an Image object, how could you save it as an image file?**

**ANS:-**

## Saving Images with Popular Python Libraries

In Python, manipulating and saving images is a common task encountered across various domains, from web development to data science. Thankfully, Python offers a plethora of powerful libraries tailored for this. Whether you’re a seasoned developer or just diving into the world of image processing, exploring these libraries opens up a world of possibilities.

### Using the Pillow Library

PIL (Python Imaging Library), now known as Pillow, is a widely-used Python library for image processing tasks. It provides comprehensive functions for opening, manipulating, and saving images in various formats. When it comes to saving images, Pillow offers simple yet powerful methods for writing images to disk in formats such as JPEG, PNG, and BMP. Its ease of use and compatibility with a wide range of image formats make it a go-to choice for many Python developers seeking to save images efficiently and effectively.

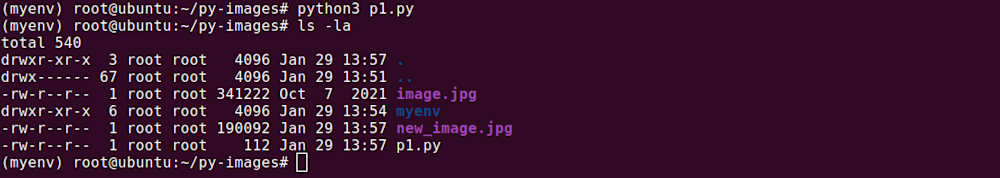
To install Pillow, run pip install pillow

To save an image with PIL, you use the save() function. The save() function requires one parameter: the name of the image file (including the format). Here is an example:

from PIL import Image

img = Image.open('image.jpg') # Load the image

img.save('new\_image.jpg') # Save the image



### Using the URLLIB Library

The urllib library in Python is primarily used for making HTTP requests and handling URLs. While it’s not specifically designed for image manipulation, it can download images from the web by fetching their URLs. By combining urllib with other libraries like os or io, you can save the downloaded image data to your local filesystem. While it provides basic functionality for downloading images, more specialized image processing tasks may require additional libraries like Pillow or OpenCV.

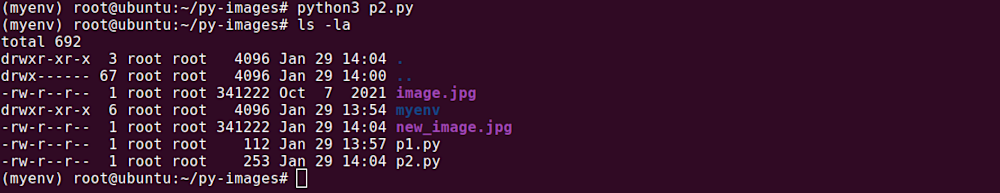
**Note:** In Python 3.5, you’ll need to install the requests library by running pip3 install requests

Here is how to save an image from a URL using urllib:

import urllib.request

url = 'http://example.com/image.jpg' # The image URL

urllib.request.urlretrieve(url, 'new\_image.jpg') # Save the image



In this example, we specified the image URL and then saved the image to our local system using the urlretrieve() function.

### Using the OpenCV Library

OpenCV, an open-source computer vision and image processing library, is a powerhouse for handling and manipulating images in Python. While its primary focus lies in tasks like object detection, image segmentation, and feature extraction, OpenCV also provides robust functionality for saving images. With its imwrite() function, you can effortlessly save pictures in various formats, including JPEG, PNG, and BMP. OpenCV’s efficiency and versatility make it a popular choice for basic image processing tasks and complex computer vision projects, offering a seamless experience for saving images alongside its myriad other capabilities.

To install OpenCV, run the command pip install opencv-python. Once you’ve installed OpenCV, you can start manipulating and saving images.

To save an image with OpenCV, use the imwrite() function. The imwrite() function requires two parameters: the name of the image file (including the format) and the image you want to save. Here’s a basic example:

import cv2

img = cv2.imread('image.jpg', 1) # Load the image

cv2.imshow('image', img) # Display the image

k = cv2.waitKey(0) # Wait for a key press

if k == 27: # wait for ESC key to exit

cv2.destroyAllWindows() # destroy all windows

elif k == ord('s'): # wait for 's' key to save and exit

cv2.imwrite('new\_image.jpg', img) # Save the image

cv2.destroyAllWindows() # Destroy all windows

In this example, we loaded an image, displayed it, and then waited for a key press. If the key pressed was s, we saved the image and destroyed all windows.

### Using the Pickle Module

The Pickle library in Python is primarily used for serializing and deserializing Python objects. However, it’s not suitable for saving images directly. Pickle is designed to handle Python objects and is not optimized for binary data like images. Attempting to save images using Pickle may result in larger file sizes and potential data corruption. For saving images, it’s recommended to use libraries like Pillow or OpenCV, which are specifically designed for image manipulation and file I/O operations. These libraries offer more efficient and reliable methods for saving images in various formats.

Here is how to save an image with Pickle:

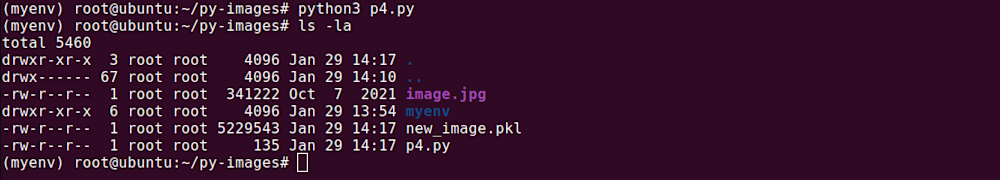
import pickle

import cv2

img = cv2.imread('image.jpg') # Load the image

pickle.dump(img, open('new\_image.pkl', 'wb')) # Save the image

In this example, we used OpenCV to load the image and then saved it with Pickle.



### Using the Matplotlib Library

Matplotlib is renowned for its capability to create an array of visualizations in Python, but it can be used for more than just generating plots. With Matplotlib, you can effortlessly load an image, manipulate it, and then save the modified version. Whether it’s adjusting colors, adding annotations, or applying filters, Matplotlib provides a convenient framework for these tasks.

You can install Matplotlib by running the command pip install matplotlib.

To save an image with Matplotlib, you use the imsave() function. Here is an example:

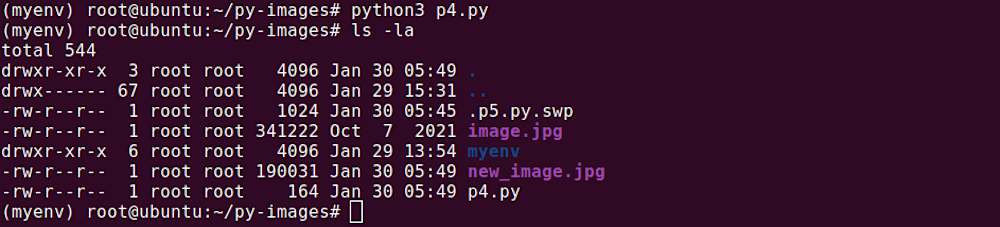
import matplotlib.pyplot as plt

import matplotlib.image as mpimg

img = mpimg.imread('image.jpg') # Load the image

plt.imsave('new\_image.jpg', img) # Save the image

In this example, we loaded an image and then saved it using the imsave() function.



### Using the Skimage Library

The Scikit-image (skimage) library in Python is a comprehensive toolbox for image processing tasks, offering a wide range of functions for tasks such as filtering, segmentation, and feature extraction. While its primary focus is on image manipulation and analysis, skimage also provides utilities for saving images in various formats. Whether you’re working with medical images, satellite imagery, or photographs, skimage’s intuitive interface makes it easy to apply transformations and enhancements before saving the results to disk.

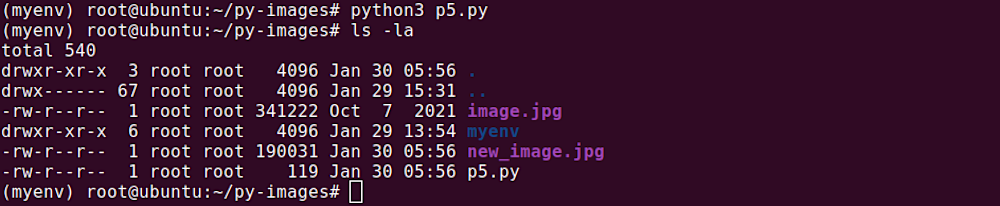
You can install skimage by running the command: pip install scikit-image.

To save an image with Skimage, you use the imsave() function. Here is an example:

from skimage import io

img = io.imread('image.jpg') # Load the image

io.imsave('new\_image.jpg', img) # Save the image



In this example, we loaded an image and then saved it using the imsave() function.

**7. What module contains Pillow’s shape-drawing code?**

**ANS:-**

## The Image Object

A crucial class in the Python Imaging Library is the Image class. It's defined in the Image module and provides a PIL image on which manipulation operations can be carried out. An instance of this class can be created in several ways: by loading images from a file, creating images from scratch, or as a result of processing other images. We'll see all these in use.

To load an image from a file, we use the open() function in the Image module, passing it the path to the image.

from PIL import Image

image = Image.open('demo\_image.jpg')

If successful, the above returns an Image object. If there was a problem opening the file, an OSError exception will be raised.

After obtaining an Image object, you can now use the methods and attributes defined by the class to process and manipulate it. Let's start by displaying the image. You can do this by calling the show() method on it. This displays the image on an external viewer (usually Preview on macOS, xv on Unix, and the Paint program on Windows).

image.show()

You can get some details about the image using the object's attributes.

# The file format of the source file.

print(image.format) # Output: JPEG

# The pixel format used by the image. Typical values are "1", "L", "RGB", or "CMYK."

print(image.mode) # Output: RGB

# Image size, in pixels. The size is given as a 2-tuple (width, height).

print(image.size) # Output: (1920, 1280)

# Colour palette table, if any.

print(image.palette) # Output: None

For more on what you can do with the Image class, check out the [documentation](https://pillow.readthedocs.io/en/stable/reference/Image.html).

## Changing Image Type

When you are done processing an image, you can save it to a file with the save() method, passing in the name that will be used to label the image file. When saving an image, you can specify a different extension from its original, and the saved image will be converted to the specified format.

image = Image.open('demo\_image.jpg')

image.save('new\_image.png')

The above creates an Image object loaded with the demo\_image.jpg image and saves it to a new file, new\_image.png. Pillow sees the file extension has been specified as PNG and so it converts it to PNG before saving it to file. You can provide a second argument to save() to explicitly specify a file format. This image.save('new\_image.png', 'PNG') will do the same thing as the previous save(). Usually, it's unnecessary to supply this second argument as Pillow will determine the file storage format to use from the filename extension, but if you're using non-standard extensions, then you should always specify the format this way.

## Resizing Images

To resize an image, you call the resize() method on it, passing in a two-integer tuple argument representing the width and height of the resized image. The function doesn't modify the used image; it instead returns another Image with the new dimensions.

image = Image.open('demo\_image.jpg')

new\_image = image.resize((400, 400))

new\_image.save('image\_400.jpg')

print(image.size) # Output: (1920, 1280)

print(new\_image.size) # Output: (400, 400)

The resize() method returns an image whose width and height exactly match the passed in value. This could be what you want, but at times you might find that the images returned by this function aren't ideal. This is mostly because the function doesn't account for the image's Aspect Ratio, so you might end up with an image that either looks stretched or squished.

You can see this in the newly created image from the above code: image\_400.jpg. It looks a bit squished horizontally.



If you want to resize images and keep their aspect ratios, then you should instead use the thumbnail() function to resize them. This also takes a two-integer tuple argument representing the maximum width and maximum height of the thumbnail.

image = Image.open('demo\_image.jpg')

image.thumbnail((400, 400))

image.save('image\_thumbnail.jpg')

print(image.size) # Output: (400, 267)

The above will result in an image sized 400x267, having kept the aspect ratio of the original image. As you can see below, this results in a better-looking image.



Another significant difference between the resize() and thumbnail() functions is that the resize() function 'blows up' an image if given parameters that are larger than the original image, while the thumbnail() function doesn't. For example, given an image of size 400x200, a call to resize((1200, 600)) will create a larger-sized image 1200x600; thus, the image will have lost some definition and is likely to be blurry compared to the original. On the other hand, a call to thumbnail((1200, 600)) using the original image will result in an image that keeps its size 400x200 since both the width and height are less than the specified maximum width and height.

## Cropping

When an image is cropped, a rectangular region inside the image is selected and retained while everything else outside the region is removed. With the Pillow library, you can crop an image with the crop() method of the Image class. The method takes a box tuple that defines the position and size of the cropped region and returns an Image object representing the cropped image. The coordinates for the box are (left, upper, right, lower). The cropped section includes the left column and the upper row of pixels and goes up to (but doesn't include) the right column and bottom row of pixels. This is better explained with an example.

image = Image.open('demo\_image.jpg')

box = (200, 300, 700, 600)

cropped\_image = image.crop(box)

cropped\_image.save('cropped\_image.jpg')

# Print size of cropped image

print(cropped\_image.size) # Output: (500, 300)

This is the resulting image:



The Python Imaging Library uses a coordinate system that starts with (0, 0) in the upper left corner. The first two values of the box tuple specify the upper left starting position of the crop box. The third and fourth values specify the distance in pixels from this starting position towards the right and bottom direction, respectively. The coordinates refer to positions between the pixels, so the region in the above example is exactly 500x300 pixels.

## Pasting an Image onto Another Image

Pillow enables you to paste an image onto another one. Some example use cases where this could be useful is in the protection of publicly available images by adding watermarks on them, the branding of images by adding a company logo, and in any other case where there is a need to merge two images.

Pasting is done with the paste() function. This modifies the Image object in place, unlike the other processing functions we've looked at so far that return a new Image object. Because of this, we'll first make a copy of our demo image before performing the paste so that we can continue with the other examples with an unmodified image.

image = Image.open('demo\_image.jpg')

logo = Image.open('logo.png')

image\_copy = image.copy()

position = ((image\_copy.width - logo.width), (image\_copy.height - logo.height))

image\_copy.paste(logo, position)

image\_copy.save('pasted\_image.jpg')

In the above, we load in two images, unsplash\_01.jpg and logo.png, then make a copy of the former with copy(). We want to paste the logo image onto the copied image, and we want it to be placed on the bottom right corner. This is calculated and saved in a tuple. The tuple can either be a 2-tuple giving the upper left corner, a 4-tuple defining the left, upper, right, and lower pixel coordinate, or None (same as (0, 0)). We then pass this tuple to paste() together with the image that will be pasted.

You can see the result below.



That's not the result we were expecting.

By default, when you perform a paste, transparent pixels are pasted as solid pixels, thus the black (white on some OSs) box surrounding the logo. Most of the time, this isn't what you want. You can't have your watermark covering the underlying image's content. We would rather have transparent pixels appear as such.

To achieve this, you need to pass in a third argument to the paste() function. This argument is the transparency mask Image object. A mask is an Image object where the alpha value is significant, but its green, red, and blue values are ignored. If a mask is given, paste() updates only the regions indicated by the mask. You can use either 1, L, or RGBA images for masks. Pasting an RGBA image and also using it as the mask would paste the opaque portion of the image but not its transparent background. If you modify the paste as shown below, you should have a pasted logo with transparent pixels.

image\_copy.paste(logo, position, logo)



## Rotating Images

You can rotate images with Pillow using the rotate() method. This takes an integer or float argument representing the degrees to rotate an image and returns a new Image object of the rotated image. The rotation is done counterclockwise.

image = Image.open('demo\_image.jpg')

image\_rot\_90 = image.rotate(90)

image\_rot\_90.save('image\_rot\_90.jpg')

image\_rot\_180 = image.rotate(180)

image\_rot\_180.save('image\_rot\_180.jpg')

In the above, we save two images to disk: one rotated at 90 degrees, the other at 180. The resulting images are shown below.





By default, the rotated image keeps the dimensions of the original image. This means that for angles other than multiples of 180, the image will be cut and/or padded to fit the original dimensions. If you look closely at the first image above, you'll notice that some of it has been cut to fit the original height, and its sides have been padded with black background (transparent pixels on some OSs) to fit the original width. The example below shows this more clearly.

image.rotate(18).save('image\_rot\_18.jpg')

The resulting image is shown below:



To expand the dimensions of the rotated image to fit the entire view, you pass a second argument to rotate() as shown below.

image.rotate(18, expand=True).save('image\_rot\_18.jpg')

Now the contents of the image will be fully visible, and the dimensions of the image will have increased to account for this.



## Flipping Images

You can also flip images to get their mirror version. This is done with the transpose() function. It takes one of the following options: PIL.Image.FLIP\_LEFT\_RIGHT, PIL.Image.FLIP\_TOP\_BOTTOM, PIL.Image.ROTATE\_90, PIL.Image.ROTATE\_180, PIL.Image.ROTATE\_270 PIL.Image.TRANSPOSE or PIL.Image.TRANSVERSE.

image = Image.open('demo\_image.jpg')

image\_flip = image.transpose(Image.FLIP\_LEFT\_RIGHT)

image\_flip.save('image\_flip.jpg')

The resulting image can be seen below.



## Drawing on Images

With Pillow, you can also draw on an image using the [ImageDraw](https://pillow.readthedocs.io/en/stable/reference/ImageDraw.html" \t "_blank) module. You can draw lines, points, ellipses, rectangles, arcs, bitmaps, chords, pie slices, polygons, shapes, and text.

from PIL import Image, ImageDraw

canvas = Image.new('RGB', (400, 300), 'white')

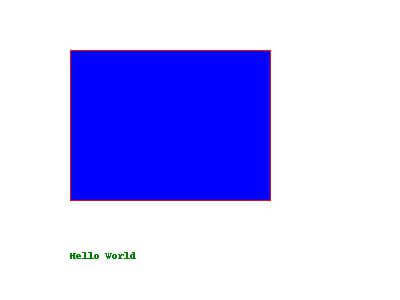
img\_draw = ImageDraw.Draw(canvas)

img\_draw.rectangle((70, 50, 270, 200), outline='red', fill='blue')

img\_draw.text((70, 250), 'Hello World', fill='green')

canvas.save('drawn\_image.jpg')

In the example, we create an Image object with the new() method. This returns an Image object with no loaded image. We then add a rectangle and some text to the image before saving it.



## Color Transforms

### Converting between modes

The Pillow library enables you to convert images between different pixel representations using the convert() method. It supports conversions between L (greyscale), RGB, and CMYK modes.

In the example below, we convert the image from RGB to L (luminance) mode, which will result in a greyscale image.

image = Image.open('demo\_image.jpg')

greyscale\_image = image.convert('L')

greyscale\_image.save('greyscale\_image.jpg')

print(image.mode) # Output: RGB

print(greyscale\_image.mode) # Output: L



### Splitting and Merging Bands

You can also split a multi-band image (such as an RGB) into individual bands using the split() method. split() creates new images, each containing one band from the original image.

You can merge a set of single band images into a new multi-band image using the merge() function. merge() takes a mode and a tuple of images and combines them into a new image.

image = Image.open('demo\_image.jpg')

red, green, blue = image.split()

print(image.mode) # Output: RGB

print(red.mode) # Output: L

print(green.mode) # Output: L

print(blue.mode) # Output: L

new\_image = Image.merge("RGB", (green, red, blue))

new\_image.save('new\_image.jpg')

print(new\_image.mode) # Output: RGB

In the above code, we split an RGB image into individual bands, swap them, and then merge them. Below is the resulting image.



## Image enhancements

Pillow allows you to enhance an image by adjusting its contrast, color, brightness, and sharpness using classes in the ImageEnhance module.

from PIL import Image, ImageEnhance

image = Image.open('demo\_image.jpg')

contrast = ImageEnhance.Contrast(image)

contrast.enhance(1.5).save('contrast.jpg')

In the above, we adjust the image contrast by a factor of 1.5. The factor used in the enhancement classes is a floating-point value that determines the level of enhancement. A factor of 1.0 returns a copy of the original image; lower factors mean less of the particular enhancement and higher values more. There is no restriction to this value.

You can see the enhanced image below.



Below, we increase the color of the image. If we used a factor of 0.0, we would get a black and white image.

color = ImageEnhance.Color(image)

color.enhance(1.5).save('color.jpg')



Below we make the image brighter. A factor of 0.0 would produce a black image.

brightness = ImageEnhance.Brightness(image)

brightness.enhance(1.5).save('brightness.jpg')



Below we make the image sharper. An enhancement factor of 0.0 would produce a blurred image, and a factor of 2.0 would give a sharpened image.

sharpness = ImageEnhance.Sharpness(image)

sharpness.enhance(1.5).save('sharpness.jpg')



**8. Image objects do not have drawing methods. What kind of object does? How do you get this kind of object?**

**ANS:-**

# Python Pillow – ImageDraw Module

**Last Updated :**08 Feb, 2022

Python’s Pillow which is a fork of the discontinued Python Imaging Library (PIL) is a powerful library that is capable of adding image processing capabilities to your python code. Pillow offers many modules that ease the process of working and modifying images.

In this article, we will have a look at the ImageDraw module of this library. ImageDraw provides a variety of methods to, as its name suggests, draw on images. With the help of this module, we can draw lines, circles, rectangles and, even write and format text on an image.

## Drawing common shapes on image

The image we will be using can be displayed using PIL as follows:

* Python

|  |
| --- |
| # Importing Image and ImageDraw from PIL  from PIL import Image, ImageDraw    # Opening the image to  # be used and displaying it  img = Image.open('img\_path.png')  img.show() |

**Output**



*The output image is as follows*

We can draw shapes and figures on an Image using the Draw method by firstly creating a Draw object.

### Drawing a rectangle on the image:

For drawing a rectangle, we use the rectangle drawing method of the ImageDraw module:

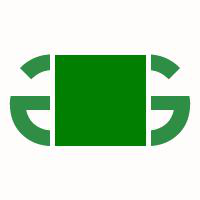
***Syntax:****ImageDraw.rectangle(xy, fill, outline, width)*

***The parameters for this method are:***

* ***xy :****Corresponds to the tuple of set of points in the upper left corner and lower right coordinates enclosing your shape. The points are passed in a tuple as follows :  (upper left x-coordinate, upper left y-coordinate, lower right x-coordinate, lower right y-coordinate)*
* ***fill :****Corresponds to the tuple of RGB colour values to fill the shape with.*
* ***outline :****Corresponds to the tuple of RGB colour values assigned for the shape’s boundary.*
* ***width :****Integer value corresponding to the thickness of the boundary of the shape.NOTE: The parameters are similar across various shape drawing methods.*
* Python

|  |
| --- |
| # Importing Image and ImageDraw from PIL  from PIL import Image, ImageDraw    # Opening the image to be used  img = Image.open('img\_path.png')    # Creating a Draw object  draw = ImageDraw.Draw(img)    # Drawing a green rectangle  # in the middle of the image  draw.rectangle(xy = (50, 50, 150, 150),                 fill = (0, 127, 0),                 outline = (255, 255, 255),                 width = 5)    # Method to display the modified image  img.show() |

**Output:**



*Output image of the rectangle method*

### Drawing an ellipse(circle) on the image:

For drawing an ellipse shape, we use the ellipse method of the ImageDraw methods:

***Syntax:****ImageDraw.ellipse(xy, fill, outline, width)*

The co-ordinates you will provide in xy will act as a box in which the circle will be enclosed.

* Python

|  |
| --- |
| # Importing Image and ImageDraw from PIL  from PIL import Image, ImageDraw    # Opening the image to be used  img = Image.open('img\_path.png')    # Creating a Draw object  draw = ImageDraw.Draw(img)    # Drawing a green circle on the image  draw.circle(xy = (50, 50, 150, 150),              fill = (0, 127, 0),              outline = (255, 255, 255),              width = 5)    # Method to display the modified image  img.show() |

**Output:**



*Output image of the ellipse method*

### Drawing a line on the image:

For drawing a line, we use the line method of the ImageDraw methods:

***Syntax:****lImageDraw.ine(xy, fill, width)*

Here, the outline parameter is not considered and the width will determine how long the line should be.

* Python

|  |
| --- |
| # Importing Image and ImageDraw from PIL  from PIL import Image, ImageDraw    # Opening the image to be used  img = Image.open('img\_path.png')    # Creating a Draw object  draw = ImageDraw.Draw(img)    # Drawing a green vertical  # line in the middle image  draw.line(xy=(50, 150, 150, 50),            fill=(0, 128, 0), width = 5)    # Method to display the modified image  img.show() |

**Output:**



*Output image of the line method*

### Drawing a polygon on the image:

We can draw a polygon of the desired shape by using the polygon method of the ImageDraw methods:

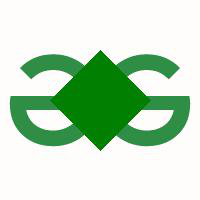
***Syntax:****ImageDraw.polygon(xy, fill, outline)*

The xy tuple parameter will contain co-ordinates based on the number of sides you want for your shape. Here, the width parameter is not valid.

* Python

|  |
| --- |
| # Importing Image and ImageDraw from PIL  from PIL import Image, ImageDraw    # Opening the image to be used  img = Image.open('img\_path.png')    # Creating a Draw object  draw = ImageDraw.Draw(img)    # Drawing a green diamond-shaped  # polygon in the middle of the image  draw.polygon(xy=(50, 150, 150, 50),               fill=(0, 128, 0),               outline=(255, 255, 255))    # Method to display the modified image  img.show() |

**Output:**



*Output image of the polygon method*

**Similarly, we can draw some other shapes using these methods:**

1. **Arc:** ImageDraw.arc(xy, start, end, fill, width)
2. **Chord (Bow-shape):** ImageDraw.chord(xy, start, end, fill, outline, width)
3. **Point:**ImageDraw.point(xy, fill)
4. **Pieslice:** ImageDraw.pieslice(xy, start, end, fill, outline, width)

The start and end parameters correspond to the degree of angles in clockwise direction, which will be connected with a line.

## Writing text on image:

Using our Draw object we can also write text on an image. It can be done using the Text method:

***Syntax:****ImageDraw.text(xy, text, fill, font, anchor, spacing, align, direction, features, language, stroke\_width, stroke\_fill, embedded\_color)*

We will also be using ImageFont from PIL to use the desired font for our text.

* Python

|  |
| --- |
| # Importing Image, ImageDraw and ImageFont  # from PIL  from PIL import Image, ImageDraw, ImageFont    # Opening the image to be used  img = Image.open('img\_path.png')    # Creating an instance for  # the font to be used using ImageFont  # Here we pass the font name and  # the font size as arguments  fnt = ImageFont.truetype("Pillow/Tests/fonts/FreeMono.ttf", 20)    # Creating a Draw object  draw = ImageDraw.Draw(img)    # Drawing the text on the image  draw.text(xy=(25, 160),            text="Hello, Geeks!",            font=fnt,            fill=(0, 127, 0))    img.show() |

**Output:**

