**Tutorial**

**Using the Image class**

The most important class in the Python Imaging Library is the [:py:class:`~PIL.Image.Image`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) class, defined in the module with the same name. You can create instances of this class in several ways; either by loading images from files, processing other images, or creating images from scratch.

To load an image from a file, use the [:py:func:`~PIL.Image.open`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) function in the [:py:mod:`~PIL.Image`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) module:

>>> from PIL import Image

>>> im = Image.open("lena.ppm")

If successful, this function returns an [:py:class:`~PIL.Image.Image`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) object. You can now use instance attributes to examine the file contents:

>>> from \_\_future\_\_ import print\_function

>>> print(im.format, im.size, im.mode)

PPM (512, 512) RGB

The [:py:attr:`~PIL.Image.Image.format`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) attribute identifies the source of an image. If the image was not read from a file, it is set to None. The size attribute is a 2-tuple containing width and height (in pixels). The [:py:attr:`~PIL.Image.Image.mode`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst)attribute defines the number and names of the bands in the image, and also the pixel type and depth. Common modes are “L” (luminance) for greyscale images, “RGB” for true color images, and “CMYK” for pre-press images.

If the file cannot be opened, an [:py:exc:`IOError`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) exception is raised.

Once you have an instance of the [:py:class:`~PIL.Image.Image`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) class, you can use the methods defined by this class to process and manipulate the image. For example, let’s display the image we just loaded:

>>> im.show()

Note

The standard version of [:py:meth:`~PIL.Image.Image.show`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) is not very efficient, since it saves the image to a temporary file and calls the [:command:`xv`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) utility to display the image. If you don’t have [:command:`xv`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) installed, it won’t even work. When it does work though, it is very handy for debugging and tests.

The following sections provide an overview of the different functions provided in this library.

**Reading and writing images**

The Python Imaging Library supports a wide variety of image file formats. To read files from disk, use the[:py:func:`~PIL.Image.open`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) function in the [:py:mod:`~PIL.Image`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) module. You don’t have to know the file format to open a file. The library automatically determines the format based on the contents of the file.

To save a file, use the [:py:meth:`~PIL.Image.Image.save`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) method of the [:py:class:`~PIL.Image.Image`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) class. When saving files, the name becomes important. Unless you specify the format, the library uses the filename extension to discover which file storage format to use.

**Convert files to JPEG**

from \_\_future\_\_ import print\_function

import os, sys

from PIL import Image

for infile in sys.argv[1:]:

f, e = os.path.splitext(infile)

outfile = f + ".jpg"

if infile != outfile:

try:

Image.open(infile).save(outfile)

except IOError:

print("cannot convert", infile)

A second argument can be supplied to the [:py:meth:`~PIL.Image.Image.save`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) method which explicitly specifies a file format. If you use a non-standard extension, you must always specify the format this way:

**Create JPEG thumbnails**

from \_\_future\_\_ import print\_function

import os, sys

from PIL import Image

size = (128, 128)

for infile in sys.argv[1:]:

outfile = os.path.splitext(infile)[0] + ".thumbnail"

if infile != outfile:

try:

im = Image.open(infile)

im.thumbnail(size)

im.save(outfile, "JPEG")

except IOError:

print("cannot create thumbnail for", infile)

It is important to note that the library doesn’t decode or load the raster data unless it really has to. When you open a file, the file header is read to determine the file format and extract things like mode, size, and other properties required to decode the file, but the rest of the file is not processed until later.

This means that opening an image file is a fast operation, which is independent of the file size and compression type. Here’s a simple script to quickly identify a set of image files:

**Identify Image Files**

from \_\_future\_\_ import print\_function

import sys

from PIL import Image

for infile in sys.argv[1:]:

try:

with Image.open(infile) as im:

print(infile, im.format, "%dx%d" % im.size, im.mode)

except IOError:

pass

**Cutting, pasting, and merging images**

The [:py:class:`~PIL.Image.Image`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) class contains methods allowing you to manipulate regions within an image. To extract a sub-rectangle from an image, use the [:py:meth:`~PIL.Image.Image.crop`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) method.

**Copying a subrectangle from an image**

box = (100, 100, 400, 400)

region = im.crop(box)

The region is defined by a 4-tuple, where coordinates are (left, upper, right, lower). The Python Imaging Library uses a coordinate system with (0, 0) in the upper left corner. Also note that coordinates refer to positions between the pixels, so the region in the above example is exactly 300x300 pixels.

The region could now be processed in a certain manner and pasted back.

**Processing a subrectangle, and pasting it back**

region = region.transpose(Image.ROTATE\_180)

im.paste(region, box)

When pasting regions back, the size of the region must match the given region exactly. In addition, the region cannot extend outside the image. However, the modes of the original image and the region do not need to match. If they don’t, the region is automatically converted before being pasted (see the section on [:ref:`color-transforms`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) below for details).

Here’s an additional example:

**Rolling an image**

def roll(image, delta):

"Roll an image sideways"

xsize, ysize = image.size

delta = delta % xsize

if delta == 0: return image

part1 = image.crop((0, 0, delta, ysize))

part2 = image.crop((delta, 0, xsize, ysize))

part1.load()

part2.load()

image.paste(part2, (0, 0, xsize-delta, ysize))

image.paste(part1, (xsize-delta, 0, xsize, ysize))

return image

Note that when pasting it back from the [:py:meth:`~PIL.Image.Image.crop`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) operation, [:py:meth:`~PIL.Image.Image.load`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) is called first. This is because cropping is a lazy operation. If [:py:meth:`~PIL.Image.Image.load`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) was not called, then the crop operation would not be performed until the images were used in the paste commands. This would mean that part1 would be cropped from the version of image already modified by the first paste.

For more advanced tricks, the paste method can also take a transparency mask as an optional argument. In this mask, the value 255 indicates that the pasted image is opaque in that position (that is, the pasted image should be used as is). The value 0 means that the pasted image is completely transparent. Values in-between indicate different levels of transparency. For example, pasting an RGBA image and also using it as the mask would paste the opaque portion of the image but not its transparent background.

The Python Imaging Library also allows you to work with the individual bands of an multi-band image, such as an RGB image. The split method creates a set of new images, each containing one band from the original multi-band image. The merge function takes a mode and a tuple of images, and combines them into a new image. The following sample swaps the three bands of an RGB image:

**Splitting and merging bands**

r, g, b = im.split()

im = Image.merge("RGB", (b, g, r))

Note that for a single-band image, [:py:meth:`~PIL.Image.Image.split`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) returns the image itself. To work with individual color bands, you may want to convert the image to “RGB” first.

**Geometrical transforms**

The [:py:class:`PIL.Image.Image`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) class contains methods to [:py:meth:`~PIL.Image.Image.resize`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) and[:py:meth:`~PIL.Image.Image.rotate`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) an image. The former takes a tuple giving the new size, the latter the angle in degrees counter-clockwise.

**Simple geometry transforms**

out = im.resize((128, 128))

out = im.rotate(45) # degrees counter-clockwise

To rotate the image in 90 degree steps, you can either use the [:py:meth:`~PIL.Image.Image.rotate`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) method or the[:py:meth:`~PIL.Image.Image.transpose`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) method. The latter can also be used to flip an image around its horizontal or vertical axis.

**Transposing an image**

out = im.transpose(Image.FLIP\_LEFT\_RIGHT)

out = im.transpose(Image.FLIP\_TOP\_BOTTOM)

out = im.transpose(Image.ROTATE\_90)

out = im.transpose(Image.ROTATE\_180)

out = im.transpose(Image.ROTATE\_270)

transpose(ROTATE) operations can also be performed identically with [:py:meth:`~PIL.Image.Image.rotate`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) operations, provided the expand flag is true, to provide for the same changes to the image's size.

A more general form of image transformations can be carried out via the [:py:meth:`~PIL.Image.Image.transform`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) method.

**Color transforms**

The Python Imaging Library allows you to convert images between different pixel representations using the[:py:meth:`~PIL.Image.Image.convert`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) method.

**Converting between modes**

im = Image.open("lena.ppm").convert("L")

The library supports transformations between each supported mode and the “L” and “RGB” modes. To convert between other modes, you may have to use an intermediate image (typically an “RGB” image).

**Image enhancement**

The Python Imaging Library provides a number of methods and modules that can be used to enhance images.

**Filters**

The [:py:mod:`~PIL.ImageFilter`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) module contains a number of pre-defined enhancement filters that can be used with the[:py:meth:`~PIL.Image.Image.filter`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) method.

**Applying filters**

from PIL import ImageFilter

out = im.filter(ImageFilter.DETAIL)

**Point Operations**

The [:py:meth:`~PIL.Image.Image.point`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) method can be used to translate the pixel values of an image (e.g. image contrast manipulation). In most cases, a function object expecting one argument can be passed to this method. Each pixel is processed according to that function:

**Applying point transforms**

# multiply each pixel by 1.2

out = im.point(lambda i: i \* 1.2)

Using the above technique, you can quickly apply any simple expression to an image. You can also combine the[:py:meth:`~PIL.Image.Image.point`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) and [:py:meth:`~PIL.Image.Image.paste`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) methods to selectively modify an image:

**Processing individual bands**

# split the image into individual bands

source = im.split()

R, G, B = 0, 1, 2

# select regions where red is less than 100

mask = source[R].point(lambda i: i < 100 and 255)

# process the green band

out = source[G].point(lambda i: i \* 0.7)

# paste the processed band back, but only where red was < 100

source[G].paste(out, None, mask)

# build a new multiband image

im = Image.merge(im.mode, source)

Note the syntax used to create the mask:

imout = im.point(lambda i: expression and 255)

Python only evaluates the portion of a logical expression as is necessary to determine the outcome, and returns the last value examined as the result of the expression. So if the expression above is false (0), Python does not look at the second operand, and thus returns 0. Otherwise, it returns 255.

**Enhancement**

For more advanced image enhancement, you can use the classes in the [:py:mod:`~PIL.ImageEnhance`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) module. Once created from an image, an enhancement object can be used to quickly try out different settings.

You can adjust contrast, brightness, color balance and sharpness in this way.

**Enhancing images**

from PIL import ImageEnhance

enh = ImageEnhance.Contrast(im)

enh.enhance(1.3).show("30% more contrast")

**Image sequences**

The Python Imaging Library contains some basic support for image sequences (also called animation formats). Supported sequence formats include FLI/FLC, GIF, and a few experimental formats. TIFF files can also contain more than one frame.

When you open a sequence file, PIL automatically loads the first frame in the sequence. You can use the seek and tell methods to move between different frames:

**Reading sequences**

from PIL import Image

im = Image.open("animation.gif")

im.seek(1) # skip to the second frame

try:

while 1:

im.seek(im.tell()+1)

# do something to im

except EOFError:

pass # end of sequence

As seen in this example, you’ll get an [:py:exc:`EOFError`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) exception when the sequence ends.

Note that most drivers in the current version of the library only allow you to seek to the next frame (as in the above example). To rewind the file, you may have to reopen it.

The following class lets you use the for-statement to loop over the sequence:

**Using the ImageSequence Iterator class**

from PIL import ImageSequence

for frame in ImageSequence.Iterator(im):

# ...do something to frame...

**Postscript printing**

The Python Imaging Library includes functions to print images, text and graphics on Postscript printers. Here’s a simple example:

**Drawing Postscript**

from PIL import Image

from PIL import PSDraw

im = Image.open("lena.ppm")

title = "lena"

box = (1\*72, 2\*72, 7\*72, 10\*72) # in points

ps = PSDraw.PSDraw() # default is sys.stdout

ps.begin\_document(title)

# draw the image (75 dpi)

ps.image(box, im, 75)

ps.rectangle(box)

# draw title

ps.setfont("HelveticaNarrow-Bold", 36)

ps.text((3\*72, 4\*72), title)

ps.end\_document()

**More on reading images**

As described earlier, the [:py:func:`~PIL.Image.open`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) function of the [:py:mod:`~PIL.Image`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) module is used to open an image file. In most cases, you simply pass it the filename as an argument:

im = Image.open("lena.ppm")

If everything goes well, the result is an [:py:class:`PIL.Image.Image`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) object. Otherwise, an [:exc:`IOError`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) exception is raised.

You can use a file-like object instead of the filename. The object must implement [:py:meth:`~file.read`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst), [:py:meth:`~file.seek`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst)and [:py:meth:`~file.tell`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) methods, and be opened in binary mode.

**Reading from an open file**

fp = open("lena.ppm", "rb")

im = Image.open(fp)

To read an image from string data, use the [:py:class:`~StringIO.StringIO`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) class:

**Reading from a string**

import StringIO

im = Image.open(StringIO.StringIO(buffer))

Note that the library rewinds the file (using seek(0)) before reading the image header. In addition, seek will also be used when the image data is read (by the load method). If the image file is embedded in a larger file, such as a tar file, you can use the [:py:class:`~PIL.ContainerIO`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) or [:py:class:`~PIL.TarIO`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) modules to access it.

**Reading from a tar archive**

from PIL import TarIO

fp = TarIO.TarIO("Imaging.tar", "Imaging/test/lena.ppm")

im = Image.open(fp)

**Controlling the decoder**

Some decoders allow you to manipulate the image while reading it from a file. This can often be used to speed up decoding when creating thumbnails (when speed is usually more important than quality) and printing to a monochrome laser printer (when only a greyscale version of the image is needed).

The [:py:meth:`~PIL.Image.Image.draft`](https://github.com/python-pillow/Pillow/blob/3.3.x/docs/handbook/tutorial.rst) method manipulates an opened but not yet loaded image so it as closely as possible matches the given mode and size. This is done by reconfiguring the image decoder.

**Reading in draft mode**

from \_\_future\_\_ import print\_function

im = Image.open(file)

print("original =", im.mode, im.size)

im.draft("L", (100, 100))

print("draft =", im.mode, im.size)

This prints something like:

original = RGB (512, 512)

draft = L (128, 128)

Note that the resulting image may not exactly match the requested mode and size. To make sure that the image is not larger than the given size, use the thumbnail method instead.