Python Imaging Library (PIL) quick reference

New Mexico Tech
Computer Center www.nmt.edu/tcc

Contents

1. Introduction

The Python Imaging Library (PIL) allows you to create, modify, and convert image files in a wide variety of formats using the Python language.

Refer to the author's companion publication, *Python language quick reference*, or to Web site `http://www.python.org/`, for general information about the Python language.

Use this form of `import` to use the Python Imaging Library:

```
import Image
```

First, a few definitions:

- A *band* is like a Photoshop channel. Grayscale images have one band; RGB images have three; and so on.
- A *mode* string has the same number of characters as the number of bands, and must be one of:

`"1"`	Black and white, 1 bit per pixel.
`"L"`	Gray scale, one 8-bit byte per pixel.
`"P"`	Palette encoding: one byte per pixel, with a palette of class `ImagePalette` translating the pixels to colors.
`"RGB"`	True red-green-blue color, three bytes per pixel.
`"RGBA"`	True color with a transparency band, four bytes per pixel, with the `A` channel varying from 0 for transparent to 255 for opaque.
`"CMYK"`	Cyan-magenta-yellow-black, four bytes per pixel.
`"YCbCr"`	3x8-bit pixels, color video format.
`"I"`	32-bit integer pixels.
`"F"`	32-bit floating point pixels.

- The *coordinates* of a pixel are its upper left corner. Coordinate $(0, 0)$ is the upper left corner of the image. The x coordinate increases to the right, and the y coordinate increases downward.
- *Rectangles* are given as a 4-tuple: `(`x_0`, `y_0`, `w`, `h`)` where the upper left corner is at point (x_0, y_0) and the rectangle is w pixels wide and h pixels high.

2. Creating objects of class `Image`

The following functions return an object of class `Image`:

`Image.open (` f `)`

Create an image from a file named f.

`Image.new (` $mode$`, ` $size$ `[, ` $color$`] )`

Creates a new, empty image with the given $mode$ and $size$, and all pixels with the value $color$ (defaulting to black).

`Image.blend (` i_1`, ` i_2`, ` α `)`

Creates an image by blending two images i_1 and i_2. Each pixel in the output is computed from the corresponding pixels p_1 from i_1 and p_2 from i_2, given by

$$(p_1 \times (1 - \alpha) + p_2 \times \alpha)$$

where p_1 is the pixel from i_1 and p_2 comes from i_2.

`Image.composite (` i_1`, ` i_2`, ` $mask$ `)`

Creates a composite image from images i_1 and i_2 and $mask$ is a mask image of the same size. Each pixel in the output has a value given by $(p_1 \times (1 - m) + p_2 \times m)$, where p_1 is the pixel from i_1, p_2 is the pixel from i_2, and m is the corresponding pixel from the mask.

`Image.eval (` f`, ` i `)`

Returns a new image obtained by applying to each pixel of image i a function f that takes one argument and returns one argument. If i has multiple bands, f is applied to each band.

`Image.merge (` $mode$`, ` $bandList$ `)`

Creates a multi-band image from multiple single-band images, where $mode$ is a mode string and $bandList$ is a sequence of single-band image objects all of the same size.

3. Methods on an object of class `Image`

Objects of class `Image` support these methods:

`.convert (` *mode* `)`

Returns a new image with the given *mode* string.

`.copy()`

Returns a copy of the image.

`.crop (` *box* `)`

Returns a region from the image whose location is specified by the rectangle *box*.

`.draft (` *mode*`,` *size* `)`

Set up the image loader so that future `.open()` calls convert the image to the given mode and size.

`.filter (` *name* `)`

Return a copy of the image filtered through a named filter. See `ImageFilter`, below, for valid filter names.

`.getbbox()`

Returns the bounding box of the nonzero parts of the image, as a rectangle.

`.getdata()`

Returns the entire image as a sequence of pixel values.

`.getpixel (` x`,` y `)`

Returns the pixel at position (x, y). If the image has multiple bands, returns a tuple.

`.histogram()`

For single-band images, returns a sequence of values `[`c_0`,` c_1`,` $\ldots$`]` where c_i is the number of pixels with value i. For multi-band images, returns the concatenation of those sequences for all bands.

`.offset (` Δx`[,` Δy`] )`

Returns an image where the pixels are rotated, offset by Δx in the x direction and Δy in the y directions. Pixels wrap around.

`.paste (` i`,` *box* `[,` *mask*`] )`

Modifies self in place by substituting new pixels from image i in a rectangle specified by *box*. To modify the entire image, substitute `None` for *box*. If no *mask* is given, pixels from i replace corresponding pixels in self.

If a *mask* is given, it must be an image of the same size as self. Each of its pixels is treated as a transparency mask, so that values of 0 leaves self's pixel alone, a value of 255 replaces self's pixel with the pixel from i, and intermediate values interpolate between self's and i's pixels. If i has mode `"RGBA"`, its `A` channel is used as a mask.

`.resize (` *size* `)`

Returns self, resized to a pixel size given by *size*.

`.rotate (` θ `)`

Returns self, rotated θ degrees counterclockwise around its center.

`.save (` f`[,` fmt`] )`

Save self to a file named f. If fmt is omitted, the type is given by the file extension f (e.g., `.save("foo.jpg")`. If fmt is given, it specifies the type (e.g., `"JPEG"`).

`.show()`

Under Unix, displays the image using *xv*.

`.split()`

Returns a tuple of the bands of self. For example, an RGB image will return three images, one for each band.

`.thumbnail (` $size$ `)`

Replaces self in place with a thumbnail image with the same aspect ratio and fitting inside the given $size$ 2-tuple.

`.transform (` x_s`,` y_s`, Image.EXTENT,` (x_0, y_0, x_1, y_1) `)`

Returns a transformed copy of the image. In the transformed image, the point at (x_0, y_0) in self will appear at $(0, 0)$, and point (x_1, y_1) in self will appear at (x_s, y_s).

`.transform (` x_s`,` y_s`, Image.AFFINE,` (a, b, c, d, e, f) `)`

Affine transformation. The values a through f are the first two rows of an affine transform matrix. Each pixel at (x, y) in the output image comes from position

$$(ax + by + c, dx + ey + f)$$

in the input image, rounded to the nearest pixel.

`.transpose ( Image.`*method* `)`

Return a flipped or rotated copy. The *method* may be any of `FLIP_RIGHT_LEFT`, `FLIP_TOP_BOTTOM`, `ROTATE_90`, `ROTATE_180`, or `ROTATE_270`.

4. Members of an object of class `Image`

`.format`	The format of the file from which the image was taken, or `None` if the image was created here.
`.mode`	The image mode, one of `"1"`, `"L"`, `"P"`, `"RGB"`, `"RGBA"`, or `"CMYK"`.
`.size`	The image size in pixels, as a 2-tuple `(`x`,` y`)`.
`.palette`	Color palette table if any. If the mode is `"P"`, this is an instance of class `ImagePalette`, otherwise it is `None`.
`.info`	A dictionary holding data associated with the image.

5. The `ImageDraw` module

The `ImageDraw` module gives you basic drawing functions on an image. To use:

```
import Image, ImageDraw
```

Functions:

`ImageDraw.ImageDraw ( i )`

For an image *i*, this constructor returns an object of class `ImageDraw` that can draw in image *i*. The initial ink color is set to 255, and fill is off.

`.line ( f, t )`

For two points *f* and *t*, draws a line between those points in the current ink color.

`.line ( L )`

For a list *L* containing either 2-tuples or pairs of points, draws a line connecting all the points.

`.point ( p )`

Draws a point at position *p*.

`.polygon ( L )`

Draws a polygon whose vertices are defined by list *L*.

`.rectangle ( box )`

Draws a rectangle. The *box* is a list of two positions as coordinate pairs. The second position is just outside the rectangle.

`.setink ( c )`

Set the current ink color to pixel value *c*.

`.setfill ( t )`

If *t* is true, subsequent polygons and rectangles are filled; if false, they are drawn as outlines.

6. The `ImagePalette` module

This module is necessary to work with color palettes. To use:

```
import ImagePalette
```

Functions:

`ImagePalette.ImagePalette ( mode="RGB" )`

Creates a new palette in the given mode. The initial values are a linear ramp.

7. The `ImageTk` module

This module is for use with GUI functions of Tkinter. To use:

```
import ImageTk
```

Functions:

`BitmapImage ( `*i*`[, `*options*`] )`

Given an image *i*, constructs a `BitmapImage` object that can be used wherever Tkinter expects an image object. The image must have mode `"1"`. Any keyword *options* are passed to Tkinter.

`PhotoImage ( `*i*` )`

Given an image *i*, returns a `PhotoImage` object that can be used wherever Tkinter expects an image object.

Warning! There is a bug in the current version of the Python Imaging Library that can cause your images not to display properly. When you create an object of class `PhotoImage`, the reference count for that object does not get properly incremented, so unless you keep a reference to that object somewhere else, the `PhotoImage` object may be garbage-collected, leaving your graphic blank on the application.

For example, if you have a canvas or label widget that refers to such an image object, keep a list named `.imageList` in that object, and append all `PhotoImage` objects to it as they are created. If your widget may cycle through a large number of images, you will also want to delete the objects from this list when they are no longer in use.

`PhotoImage ( `*mode*`, `*size*` )`

Creates an empty `PhotoImage` object with the given mode and size. Use the `.paste()` method, below, to add image data.

`.paste ( `*i*`, `*box*` )`

Image *i* is pasted into self. The *box* is a 4-tuple (x_0, y_0, w, h) specifying where *i* is placed in self, and that region must match the size of *i*. To paste all of *i* into self, use `None` as the second argument.

8. Supported file formats

This is a partial list of file extensions and the corresponding formats.

File Ext.	*Format*	Can Open	Can load modes:	Can save modes:
`.bmp`	`"BMP"`	Yes	All	`"1"`, `"L"`, `"P"`, `"RGB"`
`.eps`	`"EPS"`	Yes	—	`"L"`, `"RGB"`, `"CMYK"`
`.gif`	`"GIF"`	Yes	`"P"`	—
`.jpg` `.jpe` `.jpeg`	`"JPEG"`	Yes	All	`"L"`, `"RGB"`, `"CMYK"`
`.pcx`	`"PCX"`	Yes	All	`"1"`, `"L"`, `"P"`, `"RGB"`
`.pbm` `.pgm` `.ppm`	`"PPM"`	Yes	All	`"1"`, `"L"`, `"RGB"`
`.png`	`"PNG"`	Yes	`"1"`, `"L"`, `"P"`, `"RGB"`, `"RGBA"`	—
`.psd`	`"PSD"`	Yes	`"1"`, `"L"`, `"P"`, `"RGB"`	—
`.tif` `.tiff`	`"TIFF"`	Yes	All	`"1"`, `"L"`, `"P"`, `"RGB"`, `"CMYK"`
`.xbm`	`"XBM"`	Yes	All	`"1"`