PIL-Tools

Release 1.1.0

Steven Shrewsbury (stshrewsburyDev)

Apr 09, 2021

CONTENTS

Release v1.1.0.

An extension module for Pillow to add functions that help simplify some processes.

Simple and easy to learn usage

```
import PIL
import PILTools

image = PIL.Image.open("test.png")
image_draw = PILTools.ImageTools.Draw(image)

# Add a black and white filter to the image with a custom threshold
image_draw.black_and_white(threshold=100)
# Curve the corners of the image and replace the removed areas with purple
image_draw.rounded_edges(radius=50, fill=(175, 0, 200))

image.show()
```

PIL-Tools uses a PIL styled set of tools to help simplify some effects that you can create with PIL but without the long boring process of having to create them from scratch. It also supports multiple modes of images and makes setting up compatibility between modes easy.

CHAPTER

ONE

OVERVIEW

1.1 Reference

1.1.1 The PILTools module

The base PILTools module includes a few utility functions that can help with opening and handling images from PIL.

Example: Open an image from the internet into PIL

```
import PILTools

image = PILTools.open_image(url="http://example.com/image.png")
# Note: http://example.com/image.png isn't actually an image it is just there for␣
↪example purposes
```

Functions

open_online(*url*, *mode='RGBA'*, *size=None*, *resize_type=3*)

Fetches an image from the internet and then loads it into PIL. (requires requests lib)

Note: For a async version of this function refer to *async_open_online()*

Parameters

- **url** (*str*) – The source URL for the image. HTTP(S) supported, other forms may not work.
- **mode** (*str*) – The PIL mode to load/convert the image to.
- **size** (*tuple*) – The size to scale the image to. Leave as None to just keep the original image size.
- **resize_type** (*int*) – Optional resampling filter for resize if `size` defined.

Returns The loaded image as a PIL Image class.

Return type PIL.Image.Image

Raises *InvalidImageURL()* if the module cannot read any image data from the given URL.

async_open_online(*url*, *mode='RGBA'*, *size=None*, *resize_type=3*)

Asynchronously fetches an image from the internet and then loads it into PIL. (requires aiohttp lib)

Note: For a non-async version of this function refer to *open_online()*

Async

Parameters

- **url** (*str*) – The source URL for the image. HTTP(S) supported, other forms may not work.
- **mode** (*str*) – The PIL mode to load/convert the image to.
- **size** (*tuple*) – The size to scale the image to. Leave as None to just keep the original image size.
- **resize_type** (*int*) – Optional resampling filter for resize if `size` defined.

Returns The loaded image as a PIL Image class.

Return type PIL.Image.Image

Raises *InvalidImageURL()* if the module cannot read any image data from the given URL.

1.1.2 The ImageTools module

The ImageDraw module provides simple 2D effects for Image objects, this includes some image filters, custom cropping/filling and more.

Example: Round off corners of an image

```
from PIL import Image
from PILTools import ImageTools

with Image.open("test.png") as im:
    draw = ImageTools.Draw(im)
    draw.rounded_edges(radius=50)

    # write to stdout
    im.save(sys.stdout, "PNG")
```

Concepts

Colours / Colour names

Just like with PIL you can define colours using both integers and tuples. For more info on how PIL's colour system works refer to the PIL documentation.

Draw class

class Draw(*im*)

Creates a draw object that can be used to edit the defined image.

Note: the image will be modified in place.

Parameters **im** (*PIL.Image.Image*) – The image to draw in.

Methods

Draw.**black_and_white**(*threshold=150*)
Converts replaces the colour in the image with just black and white without changing the base image mode.

New in version 1.0: Supports alpha transparency

Parameters **threshold** (*int*) – The threshold for the selection between black and white between 0-255, the higher the threshold the more black the image will be. Defaults to 150.

Raises *InvalidThreshold()* if the threshold provided was invalid (not in 0-255).

Draw.**blue_screen**(*r_threshold: int = 80*, *g_threshold: int = 80*, *b_threshold: int = 100*)
Attempts to remove blue from an image based on the thresholds set.

Parameters

- **r_threshold** (*int*) – The red threshold to use between 0-255, anything **above** this threshold will be kept. Defaults to 80.
- **g_threshold** (*int*) – The green threshold to use between 0-255, anything **above** this threshold will be kept. Defaults to 80.
- **b_threshold** (*int*) – The blue threshold to use between 0-255, anything **below** this threshold will be kept. Defaults to 100.

Raises *InvalidThreshold()* if one of the thresholds provided were invalid (not in 0-255).

Draw.**create_mask**(*image=None*)
Creates a mask of the current image or a defined image.

New in 1.1.0

Parameters **image** (*PIL.Image.Image*) – The image you wish to create a mask of, will use the current image if not defined.

Returns A generated mask in *L* mode.

Return type PIL.Image.Image

Draw.**grayscale**()
Converts replaces the colour in the image with shades of gray without changing the base image mode.

New in version 1.0: Supports alpha transparency

Draw.**greyscale**()
Alias for *Draw.grayscale()*

New in version 1.0: Supports alpha transparency

Draw.**green_screen**(*r_threshold: int = 80*, *g_threshold: int = 100*, *b_threshold: int = 80*)
Attempts to remove green from an image based on the thresholds set.

Parameters

- **r_threshold** (*int*) – The red threshold to use between 0-255, anything **above** this threshold will be kept. Defaults to 80.
- **g_threshold** (*int*) – The green threshold to use between 0-255, anything **below** this threshold will be kept. Defaults to 100.
- **b_threshold** (*int*) – The blue threshold to use between 0-255, anything **above** this threshold will be kept. Defaults to 80.

Raises *InvalidThreshold()* if one of the thresholds provided were invalid (not in 0-255).

Draw.**invert**()
Inverts the colours on the base image (supports transparency).

Draw.**rainbow_text**(*xy*, *text*, *fill=None*, *randomise=False*, *font=None*, *align='left'*, *alignY='top'*, *stroke_width=0*, *stroke_fill=None*, *embedded_color=None*)
Draws a string of text at the defined position but uses different colours for each letter.

Note: This method is slightly different to vanilla PIL's PIL.ImageDraw.ImageDraw.text() function as anchors are not used.

Parameters

- **xy** (*tuple*) – The anchor coordinates of the text.
- **text** (*str*) – String to be drawn. If it contains any newline characters, the text is passed on to *Draw.rainbow_multiline_text()*.
- **fill** (*list*) – List of colours to use for the rainbow. Will uses the default rainbow colours (*RAINBOW_DEFAULT*) if none parsed.
- **randomise** (*bool*) – Make the chosen colours randomised instead of rendering in order. Makes use of random.choice() for this.
- **font** (*PIL.ImageFont.ImageFont*) – An PIL.ImageFont.ImageFont instance.
- **align** (*str*) – Determines the relative alignment of the text based off of the x co-ord.
- **alignY** (*str*) – Determines the relative alignment of the text based off of the y co-ord.
- **stroke_width** (*int*) – The width of the text stroke.
- **stroke_fill** (*tuple*) – Color to use for the text stroke. If not given, will default to the `fill` parameter colours.
- **embedded_color** (*bool*) – Whether to use font embedded color glyphs (COLR or CBDT).

Draw.**rainbow_multiline_text**(*xy*, *text*, *fill=None*, *randomise=False*, *font=None*, *spacing=4*, *align='left'*, *alignY='top'*, *text_align='left'*, *stroke_width=0*, *stroke_fill=None*, *embedded_color=None*)
Draws a string of text at the defined position but uses different colours for each letter.

Note: This method is slightly different to vanilla PIL's PIL.ImageDraw.ImageDraw.multiline_text() function as anchors are not used.

Parameters

- **xy** (*tuple*) – The anchor coordinates of the text.
- **text** (*str*) – String to be drawn.
- **fill** (*list*) – List of colours to use for the rainbow. Will uses the default rainbow colours (*RAINBOW_DEFAULT*) if none parsed.
- **randomise** (*bool*) – Make the chosen colours randomised instead of rendering in order. Makes use of random.choice() for this.
- **font** (*PIL.ImageFont.ImageFont*) – An PIL.ImageFont.ImageFont instance.
- **spacing** (*int*) – The number of pixels between lines.
- **align** (*str*) – Determines the relative alignment of the text based off of the x co-ord.
- **alignY** (*str*) –
  - Determines the relative alignment of the text based off of the y co-ord.

- **text_align** (*str*) – Sets the text alignment similar to the PIL.ImageDraw.ImageDraw.multiline_text()'s `align` argument.
- **stroke_width** (*int*) – The width of the text stroke.
- **stroke_fill** (*tuple*) – Color to use for the text stroke. If not given, will default to the `fill` parameter colours.
- **embedded_color** (*bool*) – Whether to use font embedded color glyphs (COLR or CBDT).

`Draw.`**red_screen**(*r_threshold: int = 100*, *g_threshold: int = 80*, *b_threshold: int = 80*)

Attempts to remove red from an image based on the thresholds set.

Parameters

- **r_threshold** (*int*) – The red threshold to use between 0-255, anything **below** this threshold will be kept. Defaults to 100.
- **g_threshold** (*int*) – The green threshold to use between 0-255, anything **above** this threshold will be kept. Defaults to 80.
- **b_threshold** (*int*) – The blue threshold to use between 0-255, anything **above** this threshold will be kept. Defaults to 80.

Raises *InvalidThreshold()* if one of the thresholds provided were invalid (not in 0-255).

rounded_edges(*radius*, *fill=None*, *inverted=False*, *tl=True*, *tr=True*, *bl=True*, *br=True*)

Adds a rounded edge (corner) effect to the image.

Parameters

- **radius** (*int*) – Radius of the edges in pixels.
- **fill** (*tuple*) – Colour to fill corners with, makes transparent if `None`. Uses black if image not in transparent friendly mode like `RGBA`.
- **inverted** (*bool*) – Determines if the `fill` colour covers the cropped corners or the rest of the image.
- **tl** (*bool*) – Determines whether the `top left` corner gets rounded.
- **tr** (*bool*) – Determines whether the `top right` corner gets rounded.
- **bl** (*bool*) – Determines whether the `bottom left` corner gets rounded.
- **br** (*bool*) – Determines whether the `bottom right` corner gets rounded.

Constants

RAINBOW_DEFAULT

A simple list of tuples that define the default colours used in the rainbow text.

List of colours:

- `(255, 0, 0)` - Red
- `(255, 106, 0)` - Orange
- `(255, 216, 0)` - Yellow
- `(0, 170, 0)` - Green
- `(0, 148, 255)` - Blue
- `(0, 65, 106)` - Indigo

- `(120, 0, 175)` - Violet

  Type list

1.1.3 PILTools errors

exception InvalidImageURL
Raised when PIL cannot read data from a given image URL

exception InvalidThreshold
Raised when an invalid threshold is provided

1.2 Release Notes

As the Pillow library updates I plan to try keeping this module updated to work alongside it with no issues.

As well as this I also plan to update this module whenever I find problems with the code or when I get new ideas/features to add.

It is advised that you always use the latest version of both this module and Pillow. As mentioned on the Pillow docs functionality and security fixes should not be expected to be backported to earlier versions of both modules.

1.2.1 Version 0.2

Changes:

ImageTools class removed

Into development of version 0.2 I realised that the ImageTools class doesnt fully follow the same style of PIL and also had a few problems when trying to add new features to.

To solve this I have remade the structure of both the class and the ImageTools module as a whole and now to access the class you will need to use this method:

```
# From base module
import PILTools
image_draw = PILTools.ImageTools.Draw(image)

# From the submodule
from PILTools import ImageTools
image_draw = ImageTools.Draw(image)
```

Function docstrings

Removed descriptions of arguements from the docstrings of functions to help keep it look clean.

All descriptions of both functions and their arguements can be found in the documentation.

Additions:

Alias functions added to docs

Any (currently just `greyscale()`) alias functions have been added to the documentation.

ImageTools.open_online

Added the ability to open images directly from the internet into PIL making use of the requests module.

This can be done via the use of `open_online()`.

Example:

```
from PILTools import ImageTools

image = ImageTools.open_online(url="http://example.com/image.png", mode="RGB")
# Note: http://example.com/image.png isn't actually an image it is just there for␣
↪example purposes
```

Rainbow text rendering

With this addition comes 2 functions `rainbow_text()` and `rainbow_multiline_text()`.

These functions allow you to render text in the same way PIL.ImageDraw does but with a few changes, the main one being that it renders each letter as a different colour of either the rainbow or a list of user defined colours.

Note: Parsing in a string with new line characters into the `rainbow_text()` function will cause the module to automaticly pass it onto the `rainbow_multiline_text()` function. Do not rely on this to often though as using the `rainbow_multiline_text()` function gives you access to more customisation arguments that you can not pass though the `rainbow_text()` function.

Example:

```
from PIL import Image, ImageFont
from PILTools import ImageTools

image = ImageTools.open_online(url="http://example.com/image.png", mode="RGB")
font = ImageFont.truetype("test.ttf", 25)

image_draw = ImageTools.Draw(image)

# Single line text
image_draw.rainbow_text((0, 0), text="Hello world!", font=font)

# Multiline text
image_draw.rainbow_multiline_text((0, 200), text="Hello\nworld!", font=font)

# Defining custom colours (same method on both)
colours = [(0, 0, 0), (50, 50, 50), (100, 100, 100), (150, 150, 150), (200, 200, 200),
↪ (250, 250, 250)]
image_draw.rainbow_text((0, 100), text="Hello world!", font=font, fill=colours)
```

1.2.2 Version 1.0

Changes:

async support for open_online()

I planned to add this feature before but never got around to doing it.

Now it is here! Async support for the module, currently this only affects the *open_online()* function as it adds the new *async_open_online()* method.

With this method you can now download and load images into PIL but using aiohttp with async support instead of requests without it.

```
import PILTools

image = await PILTools.async_open_online(url="http://example.com/image.png", mode="RGB
↪")
# Note: http://example.com/image.png isn't actually an image it is just there for␣
↪example purposes
```

For more info about how to use this function please refer to the library reference.

Moved online image open functions

Moved the *open_online()* and *async_open_online()* to the base of the module to make things cleaner and easier to access.

```
import PILTools

# Old (before ver 1.0)
image = PILTools.ImageTools.open_online(url="http://example.com/image.png", mode="RGB
↪")

# New (ver 1.0 and after)
image = PILTools.open_online(url="http://example.com/image.png", mode="RGB")

# Note: http://example.com/image.png isn't actually an image it is just there for␣
↪example purposes
```

Added custom errors

Added better error messages to some functions.

Fixed transparency issues

Noticed some of the functions in the ImageTools *Draw()* class were not fully supporting transparency so I fixed them up.

Now fixed functions:

- *Draw.black_and_white()*
- *Draw.grayscale()* / *Draw.greyscale()*

Additions:

async support for opening online images

New function *async_open_online()* added using aiohttp to asynchronously download images into PIL.

Chroma key functions

Added 3 new functions to the ImageTools *Draw()* class:

- *Draw.blue_screen()*
- *Draw.green_screen()*
- *Draw.red_screen()*

With these functions you are able to filter out one of either red, green or blue from an image similar to a chroma key effect.

You can customise the thresholds used in the filter to get better results, please refer to the function's docs for more info.

1.2.3 Version 1.1.0

Additions:

- New function *Draw.create_mask()* added allowing you to easily create image masks of defined images.

Changes:

- Changed version naming scheme from *major.minor/fix* to *major.minor.fix*.

CHAPTER

TWO

INSTALLATION

2.1 Install with pip

```
$ pip install PIL-Tools
```

2.2 Update with pip

```
$ pip install PIL-Tools --upgrade
```

2.3 Install from source

```
$ python setup.py install
```

2.4 Install async support:

Refer to the aiohttp install docs found [Here](https://docs.aiohttp.org/en/stable/#library-installation)

CHAPTER

THREE

SUPPORT

If you have any problems with PIL-Tools please let me know via the GitHub issue tracker.

CHAPTER

FOUR

SOME USEFUL LINKS

- Issue Tracker: https://github.com/stshrewsburyDev/PIL-Tools/issues
- Source Code: https://github.com/stshrewsburyDev/PIL-Tools

CHAPTER

FIVE

LICENSE

The project is licensed under the MIT license.

INDEX