Python Imaging Library

[Python]

- http://www.pythonware.com/library/pil/handbook/overview.htm

Pillow Python Imaging Library (Fork)

- Pillow
- https://pypi.python.org/pypi/Pillow/2.2.1
- http://pillow.readthedocs.org/

概要

- Python Imaging Library は、イメージ処理機能を Python に追加する。
- 様々なファイルフォーマットのサポート、高効率な内部描画、強力なイメージ処理機能を提供
- コア・イメージライブラリは基本的ないくつかのピクセルフォーマットで保存されたデータに高速にアクセスできるようにデザインされている
- 一般的なイメージ処理ツールとして安定した土台を提供する。

■イメージアーカイブ

- Python Imaging Library はイメージをアーカイブするバッチ処理アプリケーションに理想的である
- サムネイルを作成する、ファイルフォーマットを変換する、イメージをプリントするなど。

■イメージ表示

- 現在のリリースは、Windows DIB インターフェースと同様の、Tk PhotoImage、BitmapImage インターフェースを含む
- それは、PythonWin および 他の Windows ベースのツールキットとともに利用できる。
- デバッグ用途にて、イメージをディスクに書き込み、外部の表示ユーティリティを呼び出すメソッドも備える

■イメージ処理

- ライブラリは、基本的なイメージ処理機能 ([点演算 http://typea-mixi01.appspot.com/yh_s?q=%E7%82%B9%E6%BC%94%E7%AE%97]、畳み込みフィルタ、色空間変換) を含んでいる
- イメージのリサイズ、回転、任意のアフィン変換もサポートしている。
- ヒストグラムメソッドは統計からイメージを作成する。これは、自動コントラスト調整の強化に使える

モジュール群

- http://www.pythonware.com/library/pil/handbook/index.htm
- Python Imaging Library Modules

チュートリアル

- http://www.pythonware.com/library/pil/handbook/introduction.htm

・http://pillow.readthedocs.org/handbook/tutorial.html

■ Image クラス

・最も重要なクラス
・モジュールと同じ名前で定義されている
・いくつかの方法でインスタンスを生成できる
　・ファイルからイメージをロード
　・1 からファイルを生成する

ファイルからイメージをロードするには Image モジュールの open 関数を利用する

・成功すると Image オブジェクトを返す

```
>>> import Image
>>> im = Image.open(r'c:\work\py\pil01.jpg')
>>> print im.format, im.size, im.mode
JPEG (682, 453) RGB
```

・属性の内容

属性	内容
format	イメージのソースを特定する。ファイルから読めなかった場合は、None がセットされる。
size	幅と高さ (ピクセル) を持つタプル
mode	イメージのバンド番号と名前、ピクセルのタイプおよび深さを定義。"L" (luminance) グレースケール、"RGB" トゥルーカラー、"CMYK" CMYK カラーモデル

一旦 Image クラス を作成するとイメージの処理や操作ができる

・イメージを表示する例

```
>>> im.show()
```

■イメージの読み書き

イメージを読み込む

・ライブラリは多数のフォーマットをサポートしている
・ファイルフォーマットについて知らなくてもよい。
・ライブラリが自動的にファイル内容に基づいて決定する。

イメージの書き込み

・Image クラスの save メソッドでファイルを保存する。
・ファイルを保存するときに、ファイル名は重要

・ファイル名の拡張子に基づいた変換をライブラリは行う。

```
>>> import Image
>>> im = Image.open(r'c:\work\py\pil01.jpg')
>>> im.save(r'c:\work\py\pil01.bmp')
```

標準の拡張子を使わない場合など、save メソッドの 2 つ目の引数にフォーマットを指定することもできる

```
>>> im.save(r'c:\work\py\pil01.xxx','gif')
```

■サムネイルを作成する

```
>>> import Image
>>> size = (100, 100)
>>> im = Image.open(r'c:\work\py\pil01.jpg')
>>> im.thumbnail(size)
>>> im.save(r'c:\work\py\pil01_thum.jpg')
```

■イメージファイルの識別

・ライブラリは、ラスタデータのデコードやロードを本当に必要な場合以外行わない。
・ファイルを開いたとき、ファイルヘッダーはファイルフォーマット、モード、サイズなど、ファイルをデコードするためのプロパティを決定するために読み込まれるが、ファイルの残りの部分は後まで処理されない。
・これは、ファイルサイズや圧縮方法によらず、イメージファイルをすばやく処理できることを意味する。

```
>>> import Image
>>> im = Image.open(r'c:\work\py\pil01.jpg')
>>> print im.format, im.size, im.mode
JPEG (682, 453) RGB
```

■切り取り、貼り付け、マージ

・Image クラスはイメージの領域を操作するメソッドを含んでいる
・crop メソッドで、矩形を抽出できる
・左上隅を (0,0) としたタプルで (left, upper, right, lower) を指定して矩形を選択する

矩形を切り取り

```
>>> import os
>>> import Image
>>> os.chdir(r'c:\work\py')
>>> im = Image.open(r'.\pil01.jpg')
>>> box = (100,100,400,400)
>>> region = im.crop(box)
>>> region.save(r'.\region01.jpg')
```

矩形を貼り付けなおす

・矩形を貼り付け戻すとき、矩形のサイズが一致する必要があります。
・また、矩形はイメージの範囲外に拡張されません
・しかしながら、イメージのモードは一致する必要はありません。
・貼り付け前に自動的に変換されます

```
>>> region = region.transpose(Image.ROTATE_180)
>>> im.paste(region, box)
>>> im.save(r'.\pil01.jpg')
```

カラーバンドの分割とマージ

・RGB イメージのような、マルチバンドイメージのカラーバンドを独立して扱うことができる。
・split メソッドは、元の複数のカラーバンドからひとつの色を抜き出した新しいイメージのセットを作成する。
・merge 関数は、モードとイメージのタプルからそれらを混ぜ合わせ新しいイメージを作成する。

```
>>> import Image
>>> im = Image.open(r'c:\work\img\test01.jpg')
>>> r, g, b = im.split()
>>> im = Image.merge("RGB", (b, g, r))
>>> im.save(r'c:\work\img\test02.jpg')
```

・test01.jpg

・test02.jpg

■幾何学的な変形

・Image クラスは、resize および rotate メソッドを持つ

```
>>> im = Image.open(r'c:\work\img\test01.jpg')
>>> out = im.resize((50,50))
>>> out = out.rotate(45)
>>> out.save(r'c:\work\img\test03.jpg')
```

・test03.jpg

・transpose メソッドを使うと、rotate と同様に回転することや、垂直や水平に反転できる。

```
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)
```

```
>>> im = Image.open(r'c:\work\img\test01.jpg')
>>> im = im.transpose(Image.FLIP_LEFT_RIGHT)
>>> im.save(r'c:\work\img\test06.jpg')
```

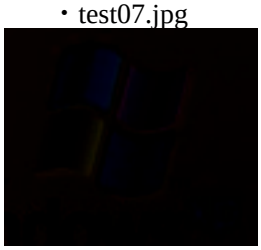
・test06.jpg

■色の変換

・convert 関数で異なるピクセル表現にイメージを変換できる
・カラーモードを変換する
  ・RGB
  ・CMYK

```
>>> import Image
>>> im = Image.open(r'c:\work\img\test01.jpg').convert('CMYK')
>>> im.save(r'c:\work\img\test07.jpg')
```

・ブラウザと CMYK の問題 (↓の画像が表示されない、色がおかしい)
・test07.jpg



■イメージ拡張機能

・多くのメソッドやモジュールが拡張機能として提供されている

フィルタの適用

・ImageFilter モジュールは、多くの事前定義されたフィルターを提供
  ・BLUR
  ・CONTOUR
  ・DETAIL
  ・EDGE_ENHANCE
  ・EDGE_ENHANCE_MORE
  ・EMBOSS
  ・FIND_EDGES
  ・SHARPEN
  ・SMOOTH
  ・SMOOTH_MORE

```
>>> import ImageFilter
>>> im = Image.open(r'c:\work\img\test01.jpg')
```

```
>>> out = im.filter(ImageFilter.EMBOSS)
>>> out.save(r'c:\work\img\test08.jpg')
```

・test08.jpg

■ポイント操作

・point メソッドは、イメージのピクセル値変換に利用できる (たとえばコントラスト操作)
・多くの場合、関数オブジェクトを引数としてこのメソッドに渡されることを期待する。
・すべてのピクセルはこの関数にしたがって処理される。

コントラスト変換

```
>>> im = Image.open(r'c:\work\img\test01.jpg')
>>> out = im.point(lambda i: i / 3.0)
>>> out.save(r'c:\work\img\test09.jpg')
```

・test09.jpg

カラーバンドを独立して操作

```
>>> im = Image.open(r'c:\work\img\test01.jpg')
>>> source = im.split()
>>> r, g , b = 0, 1, 2
>>> mask = source[r].point(lambda i: i < 100 and 255)
>>> out = source[g].point(lambda i: i * 0.7)
>>> source[g].past(out, None, mask)
>>> source[g].paste(out, None, mask)
>>> im = Image.merge(im.mode, source)
>>> im.save(r'c:\work\img\test10.jpg')
```

・test10.jpg

マスクを生成する書式

・Python は論理演算に必要な部分しか評価しない。そして最後に検査した値を返す。
・なので、以下の 式 (expression) が 偽なら false(0) を返し、真なら最後に評価した 255 を返す。

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

■強化機能 (ImageEnhance モジュール)

・イメージの強化機能クラスを、ImageEnhance モジュールで利用可能。
・いったんイメージから enhancement オブジェクトを作成したら、すばやく異なる設定を試すことができる。コントラスト調整、輝度、カラーバランス、シャープネスなど。

```
>>> import Image, ImageEnhance
>>> im = Image.open(r'c:\work\img\test01.jpg')
>>> enh = ImageEnhance.Contrast(im)
>>> out = enh.enhance(2.0)
>>> out.save(r'c:\work\img\test11.jpg')
```

・test11.jpg

■イメージシーケンス (アニメーションフォーマット)

・PIL はいくつかの基本的なイメージシーケンス (アニメーションフォーマットとも呼ばれる) を含んでいる。
・サポートしているのは、FLI/FLC、GIF、あといくつか実験的なフォーマット。
・TIFF ファイルは 1 つ以上のフレームを含むことができる。
・このようなファイルを開くと、PIL は自動的に最初のフレームをロードする。
・seek および tell メソッドによりフレーム間を移動することができる。

アニメーション GIF をフレームに分解

```
>>> import Image
>>> im = Image.open(r'c:\work\img\anime.gif')
>>> try:
...     while True:
...             im.save(r'c:\work\img\anime' + str(im.tell()) + '.gif')
...             im.seek(im.tell()+1)
... except EOFError:
...     pass
...
```

分解前

・anime.gif

分解後 1 フレーム目

・anime0.gif

分解後 4 フレーム目

・anime3.gif

分解後 8 フレーム目

・anime7.gif

■ Postscript 表示

・PIL は、テキストやグラフィックを Postscript プリンタへ印刷する関数を持つ。

■イメージを読むその他の方法

・上記までは、単純にファイル名のみを引数に Image モジュールの open 関数を利用していた。
・成功すれば、Image オブジェクトが生成され、失敗すれば IOError が発生。

バイナリモードで開いたファイルから Image を作成

```
>>> import Image
>>> fp = open(r'c:\work\img\test01.jpg', 'rb')
>>> im = Image.open(fp)
>>> im.show()
```

文字列 IO から Image を作成

```
>>> import Image, StringIO
>>> fp = open(r'c:\work\img\test01.jpg', 'rb')
>>> buf = fp.read()
>>> im = Image.open(StringIO.StringIO(buf))
>>> im.show()
```