CUTT
カットシステム

■**サンプルファイルのダウンロードについて**

本書掲載のサンプルファイルは、下記 URL からダウンロードできます。

https://----------

・本書の内容についてのご意見、ご質問は、お名前、ご連絡先を明記のうえ、小社出版部宛文書（郵送または E-mail）でお送りください。
・電話によるお問い合わせはお受けできません。
・本書の解説範囲を越える内容のご質問や、本書の内容と無関係なご質問にはお答えできません。
・匿名のフリーメールアドレスからのお問い合わせには返信しかねます。

はじめに

画像の加工や補正が必要なシーンは増えています。GIF から PNG へなどのファイルフォーマット変更、ページレイアウトに合わせたリサイズや切り出し、色調の調整、画像をレイヤ化してからの重畳、あるいはキャプションやウォーターマークの焼き込みといった作業です。数点なら画像編集アプリケーションで手作業もできますが、枚数が増えれば、手に負えません。

そこで自動化を考えます。使用する言語は、汎用的でありながら手軽に使え、利用者も情報も多く、実行環境を問わない Python が昨今では一般的でしょう。各種追加機能（外部のライブラリ）も豊富なので、いろいろな目的に利用できます。

しかし、画像プログラミングは数式が多かったりと、敷居が高いのが問題です。

その点、本書で紹介する Pillow は簡単です。他の画像処理ライブラリと比較すると、次のような特徴があります。

- 用途別に関数が用意されており、調整するパラメータが他より少ない。たとえば、リサイズだけで 8 種類あり、コントラストや輝度の調整もテレビのスライダ風にそれぞれの目的に応じて関数化されています。
- 画像表現に行列（Numpy）を使わなくてもよい（便利ですが、必須ではありません）。
- 対応可能な画像ファイルフォーマットが多い（現在 40 種類）。
- 他の画像処理ライブラリとの画像交換方法が完備している。
- 軽い（インポートが早い）。

半面、高度な画像処理機能は用意されていません。画像中の文字や顔の所在を特定したり、数値のリストからグラフを描くなどはできません。しかし、そうした処理はそれを得意とするライブラリに任せればよいのです。根幹部分は簡単な Pillow で記述し、高機能部分はネットに置かれているコードをコピペすれば、目的は達成できます。本書ではそうした使い方を念頭に、以下のライブラリとの連携方法に 1 章を割きます。

- tkinter ― Python にバンドルされている GUI ライブラリ。
- OpenCV ―画像処理ライブラリ。
- matplotlib ―グラフ描画ライブラリ。
- pywin32（win32api）― Windows API ライブラリ（の画像関連のごく一部）。

Pillow はもとは PIL（Python Imaging Library）と呼ばれていました。しかし、PIL は 1995 年の最初のリリースから 6 年後に開発が停止してしまいました。以降、同機能は Alex Clark がフォークし、修正と機能追加を行っている Pillow に移行しました。ネットで PIL とあるものは、（今は廃れた Python 2 関係の古い情報でなければ）ほぼ間違いなくこの Pillow を意味しています。

本書は Python を用いて画像の加工、補正、編集のスクリプト（プログラム）を書こうと考えるプログラマや Web デザイナ、あるいは簡単なところから画像処理を学ぼうと考える学生を対象にしています。Python については、ある程度なら書ける知識と経験があることを前提にしています。リスト内包表記やラムダ式などやや凝った記法も本書では使いますが、複雑なものは付録 B にまとめて概説を加えました。トリビアかも知れませんが、不如意なときは参照してください。

2022 年 7 月

豊沢 聡

注意事項

本書の表記や使用するサンプルファイルなどで注意すべき点を説明します。

パッケージ名

Pillow のパッケージ名は PIL ですが、本書では省きます。たとえば、マニュアルの関数定義は PIL.Image.open() のように PIL から書き起こしていますが、from PIL import Image, ImageOps のようにインポートすることを念頭に、本書では Image.open() や ImageOps.scale() とモジュール名から書きます。

Numpy も、慣例に従って import numpy as np と別名でインポートするので、それに合わせて np.ndarray のように np からスタートします（マニュアルは numpy.ndarray と書いています）。tkinter も同様で tk.Canvas() のように書きます。

関数定義

関数定義は初出の時点で、次のように示します。

```
Image.show()                                    # 戻り値なし
Image = Image.open(fp, formats=None)            # Imageオブジェクトを返す
tuple[str] = Image.getbands()                   # 文字列のタプルを返す
```

マニュアルに記載された関数引数には、用途によっては使わないものも数多くあります。本書では、説明範囲内のもののみ記載します。たとえば、上記の Image.open() 関数のマニュアル上の引数は (fp, mode='r', formats=None) ですが、mode はデフォルトの r 以外は指定できない仕様なので省きます。

左辺は戻り値の型を示します。1 行目のように未記載の関数には戻り値はありません（強いて代入すると None が得られる）。2 行目は Image オブジェクト、3 行目は文字列を要素としたタプルをそれぞれ返します。[] はコンテナタイプの内側の型を示します。たとえば、list[list] はリストのリスト、list[tuple[int]] は整数を要素としたタプルのリストです。

Python 本体の組み込み関数や標準ライブラリの定義は示しません。

Numpy や OpenCV など Pillow 以外のライブラリについては、関数定義は掲載しますが、必須引数のみ記述します。オプション引数はとくに断りがない限りデフォルトで使用します。

詳細はそれぞれのマニュアルを参照してください。マニュアルのURLは付録Dにまとめて示しました。

用例

本書では、簡単な用例はPythonのインタラクティブモードから示します。

```
>>> from PIL import Image
>>> img = Image.open('Dachshund.jpg')                    # テスト用画像を読み込み
>>> hex(id(img))                                         # オブジェクトIDを確認
'0x176abdd6110'
```

>>> はインタラクティブモードのプロンプトです。# から始まる文は、筆者が原稿に加えたコメントで、実行時に記述が必要なものではありません。

要素数の多いリストや行数のある出力結果は一部省略（… や ⋮）、あるいは読みやすい箇所で改行しています。次の例ではリストの要素を省略しています。

```
# web_mediumの中身
>>> JpegPresets.presets['web_medium']
{'subsampling': 2, 'quantization': [[16, 11, 11, 16, 23, 27, 31, 30, ..., 64]]}
```

この例では、横に長いエラー出力に改行を入れています。

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\tools\python3.10.2\lib\site-packages\PIL\Image.py",   # ここと
   line 519, in __getattr__                                      # ここはもとは1行
    raise AttributeError(name)
AttributeError: filename
```

コンソール（あるいはコマンドプロンプトやターミナル）からの実行例はWindowsのものを掲載しています。他OSのユーザは当該操作に該当する操作に読み替えてください。

```
C:\temp>dir new*                                          # Windowsコマンドプロンプト
 ⋮
2022/03/13  11:25         1,157,653 new.gif
2022/03/13  11:29           324,934 new.webp              # 約2/7
```

ファイルパスの区切り文字は Unix スタイルの /（フォーワードスラッシュ）を用います。ただし、Windows のコマンドプロンプトでは（当然ですが）Windows の \（バックスラッシュ）を用います。

日本語環境の Windows コマンドプロンプトでは、\ は ¥ で表示されます。

サンプルスクリプト

まとまった用例はスクリプトにまとめました。文中では、次のように「リスト●ファイル名」という見出しをつけて記載しました。

リスト 0.1 ● pil_paste_mask.py

```
1 from PIL import Image
2 from sys import argv
    ⋮
```

本書掲載のスクリプトは、すべて本書扉裏に示した URL からダウンロードできます。

サンプル画像

本書で用いたサンプル画像は、筆者自作のものを除いて、いずれも Pixabay から入手した「商用利用無料、帰属表示は必要ありません」のものです。これら画像は付録 C にまとめて掲載しました。

サンプル画像はサンプルスクリプトを置いたディレクトリ下の Images ディレクトリに収容してあります。本書扉裏に示した URL からダウンロードできます。

開発環境

Python も Pillow も、そして本書で取り上げる数点のライブラリもマルチプラットフォーム対応なので、開発環境は問いません。

ただし、OS 依存性のあるものについては、本書の記述はすべて Windows を対象とします。たとえば、画像を表示する `Image.open()` 関数のデフォルト画像表示アプリケーションは Windows の「フォト」アプリだと述べ、このデフォルトを変更するためのクラスは `Image.WindowsViewer` だけを説明します。

Python およびライブラリの導入方法は付録 A で説明します。

WSL

Windows にデフォルトで搭載されている Unix 仮想マシンの WSL（Windows Subsystem for Linux）でも Pillow は動作しますが、画像表示はできません。仮想マシンには物理的なディスプレイがないからです。

Windows 側に X Window System のディスプレイを提供するアプリケーションを用意すれば表示できますが、本書の範囲を超えています。WSL での開発では、画像を一時保存し、Windows のエクスプローラから開くのが迂路ですが、簡単です。もちろん、画像表示なしのバッチ処理なら、WSL も有効です。

オンライン環境

Google Colaboratory や replit など、オンライン Python 実行環境でも Pillow は実行できますが、本書で多用するコマンドライン引数からのファイルの指定、ファイル読み込みおよび描き出し、画像表示には、その環境にのっとった操作を行わなければなりません。オンライン環境は本書では扱わないので、利用に際してはそれぞれのマニュアルを参照してください。

オンライン環境では、Pillow がインストールされていたとしても、バージョンは古いのが通例です（2022 年 7 月 22 日現在、Google Colaboratory は 7.1.2、replit は 8.3.2 でした。どちらの環境でも win32api はインストールされていません）。

バージョン

対象は Python 3 です。マイナーバージョンは問いません。ただし、3.7 未満はサポートがすでに終了（End of life）しています。Python 2 も 2020 年 1 月に退役しています。

Pillow は年に 4 回、1 月、4 月、7 月、10 月の月初めにアップデートされます。アップデートされると、基本的にはマイナーバージョン（x.y.z の y の数字）が 1 つ繰り上がります。これまでの仕様（API）との互換性がなくなるとメジャーバージョン（x）の数字が繰り上がります。パッチリリース（z）はバグやセキュリティ上の問題が生じたときに不定期にリリースされます。

本書執筆時点の最新バージョンは 9.2.0（2022 年 7 月 1 日）です。本書は、バージョン 9 を対象としています。バージョン 9 の最初のリリースである 9.0.0 は 2022 年 1 月 3 日リリースなので、それ以前にインストールしているのなら、アップグレードを推奨します。

バージョン 9 内では、特に次の 2 点の変更点に注意が必要です。

定数の指定方式

Pillow 9.1.0（2022 年 4 月 1 日リリース）から定数の指定方法が変わりました。たとえば、`Image.NONE` は `Image.Dither.NONE` に、`Image.NEAREST` は `Image.Resampling.NEAREST` となりました。旧版で新しい定数を用いると、`Attribute` エラーが上がります。

```
>>> import PIL
>>> from PIL import Image
>>> PIL.__version__                                    # 2022年2月13日の版
'9.0.1'
>>> Image.NONE                                         # 古い定数はある
0
>>> Image.Dither.NONE                                  # 新しいのはまだない
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/foo/.local/lib/python3.8/site-packages/PIL/Image.py",
   line 65, in __getattr__
    raise AttributeError(f"module '{__name__}' has no attribute '{name}'")
AttributeError: module 'PIL.Image' has no attribute 'Dither'
```

逆に、最新版で古い定数を指定すると、`DeprecationWarning` が上がります。警告なので、値は得られ、スクリプト等は続行できます。

```
>>> import PIL
>>> from PIL import Image
>>> PIL.__version__                                    # 2022年4月1日の版
'9.1.0'
>>> Image.Dither.NONE                                  # 新しい定数はある
```

```
<Dither.NONE: 0>
>>> Image.NONE                                          # 古いのは警告
<stdin>:1: DeprecationWarning: NONE is deprecated and will be removed
  in Pillow 10 (2023-07-01). Use Resampling.NEAREST or Dither.NONE instead.
0
```

定数の変更は、マニュアルの［Deprecations and removals］セクションの Constants の箇所に書かれています。

本書では、本文および関数定義では新しい記述方法を用います。スクリプトファイルやインタラクティブモードでの用例では、相互運用性を鑑みて直値（たとえば Image.Dither.NONE と Image.NONE なら 0）を指定します。

フォントメトリックス関数

Pillow 9.2.0（2022 年 7 月 1 日リリース）から、フォントのサイズ等メトリックス関係の関数が一部非推奨となりました。たとえば、ImageFont.getsize() 関数は非推奨となったので、ImageFont.getbbox() または ImageFont.getlength() を使います。

```
>>> import PIL
>>> PIL.__version__                                    # 2022年7月1日の版
'9.2.0'
>>> from PIL import ImageFont
>>> f = ImageFont.truetype('C:/Windows/Fonts/Arial.ttf', size=18)
>>> f.getsize('Hello World')                           # 非推奨警告が上がる
<stdin>:1: DeprecationWarning: getsize is deprecated and
  will be removed in Pillow 10 (2023-07-01). Use getbbox or getlength instead.
(93, 17)
>>> f.getbbox('Hello World')                           # こちらは使える
(0, 4, 93, 17)
>>> f.getlength('Hello World')                         # 同上
92.71875
```

フォントメトリックス関数の変更は、マニュアルの［Deprecations and removals］セクションの Font size and offset methods の箇所に書かれています。

9.2.0 以前のバージョンでも ImageFont.getbbox() は利用できるので、本書ではそちらを使います。

その他ライブラリ

他の画像関連ライブラリは Pillow との関連でのみ扱います。主として Numpy は第 4 章、その他は第 9 章で触れます。ただ、説明は最低限に留めているので、詳細は公式ドキュメントあるいは関連書籍を参照してください（付録 D）。

これらについてもバージョンは問いませんが、今からインストールするのなら最新版がお勧めです。

文字エンコーディング

Python のデフォルトエンコーディングはバージョン 3.0 から UTF-8 です（PEP 3120）。掲載したスクリプトファイルの文字エンコーディングも BOM（Byte Order Marker）なしの UTF-8 です。

Windows コマンドプロンプトのデフォルト文字エンコーディングは Shift-JIS なため、UTF-8 日本語文字列を表示すると文字化けします。正しく表示させるには、chcp コマンドから、コマンドプロンプトのコードページを Shift-JIS（932）から UTF-8（65001）に変更します。

```
C:\temp>chcp
現在のコード ページ: 932                                    # Shift-JIS

C:\temp>chcp 65001
Active code page: 65001                                     # UTF-8
```

コードページは Windows（もともとは IBM）の文字コーディングの管理方法で、上記のように文字エンコーディング名が数値に対応しています。コードページの一覧は次の Microsoft Docs から確認できます。

```
https://docs.microsoft.com/en-us/windows/win32/intl/code-page-identifiers
```

Pillow の作者について

Pillow の保守管理者である Alex Clark はロックンローラーでもあるそうです。彼のミュージックビデオは次の YouTube チャネルから視聴できます。

```
https://www.youtube.com/user/aclark4life
```

彼のブログはこちらです。

```
https://blog.aclark.net/
```

目次

1

用 語

本章では、画像関連の用語の定義を示します。

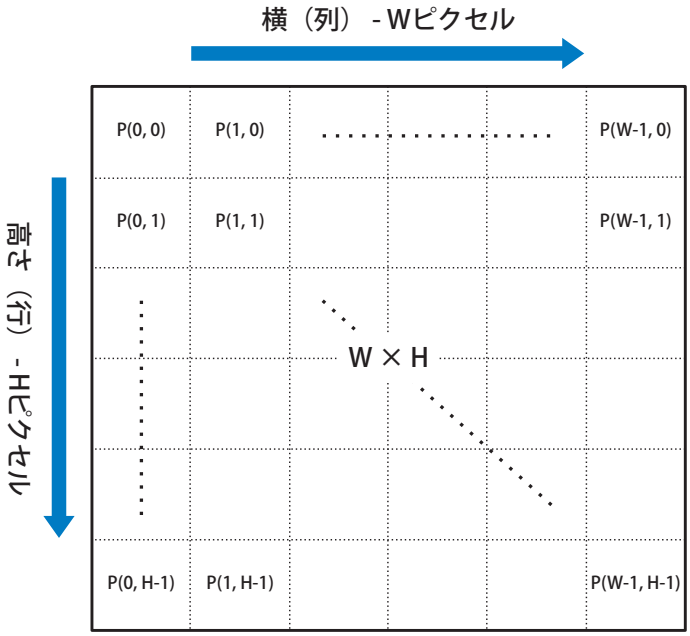
Pillow に固有な用語は、通常ならチャネル（channel）と呼ぶモノクロ画像の重なりをバンド（band）と言い換えている以外はほとんどないので、画像の扱いに慣れた読者は飛ばしても構いません。

用例は Pillow のクラスや関数から示します。これらの利用方法は第 2 章以降で順次取り上げるので、ここではそんな機能があるとだけ思って、コード部分は斜め読みしてください。

1.1 画像の構成

1 枚の画像を構成する 1 つ 1 つの点をピクセル（pixel）と言い、次の図のように横に W 個、縦に H 個が格子状に並びます。

図 1.1 ●画像の構成



このとき、画像のサイズ（size）が W × H、あるいはタプルで (W, H) であると言います。

この格子状の画像座標系では、左上の隅が位置 (0, 0) で、横方向の x 軸は左から右に画像の横幅 −1（W − 1）まで、縦方向の y 軸は上から下に高さ−1（H − 1）まで伸びています。座標系の個々のピクセルの位置は (x, y) のタプルで示されます。

Pillow の画像オブジェクト Image の横幅 W と高さ H は Image.width および Image.height 属性から、サイズは Image.size 属性からそれぞれ取得できます（3.1 節）。

```
>>> from PIL import Image                    # Imageモジュールの導入
>>> img = Image.open('Goat.jpg')             # 画像を読む（モノクロ）
>>> img.width                                # 横幅のピクセル数
1280
>>> img.height                               # 高さのピクセル数
961
>>> img.size                                 # タプル表記のサイズ
(1280, 961)
```

画像処理では、数値計算モジュールである Numpy をしばしば使います（4.3 節）。格子状に並んだ数値の羅列である画像は、数学の行列で表現できるからです。ただ、行列では縦横の順序が画像と入れ替わるところには注意が必要です。画像では横幅が先で高さが続きます。これに対し、行列ではその名称の通り行（縦方向）、続いて列（横方向）です。そのため、サイズが (1280, 961) の画像は、Numpy では (961, 1280) と表現されます。

```
# 上記インタラクティブモードからの続き
>>> import numpy as np                    # Numpyパッケージの導入
>>> arr = np.array(img)                   # 上記の画像を行列に変換
>>> arr.shape                             # 行列のサイズを取得
(961, 1280)                               # 縦横の順序が入れ替わる
```

画像の回転などで用いる角度は、Pillow では角度（°）です。たとえば、π（3.14...）ではなく 180°です。

角度の正負は関数によって異なります。関数定義から方向を確認してください。

1.2 ピクセル値

ピクセルには、色や明るさなどの情報を示す値が書き込まれています。これをピクセル値と言います。しばしば P(x, y) = z のように位置 (x, y) の関数として表記されます。z がピクセル値です。

ピクセル値は 0 から 255 の 8 ビット符号なし整数で表現されます。Pillow は 32 ビットの整数あるいは浮動小数点数もサポートしていますが、本書では扱いません。

ピクセル値は、一般にそのピクセルの明るさ（光の強度）を示します。モノクロ画像なら、次の図に示すように 0 は黒（ブラックアウト）で、値が大きくなるにつれて明るく、つまり白っぽくなっていき、最大値の 255 に達したところで真っ白（ホワイトアウト）になります。

図 1.2 ●モノクロピクセルの値と明るさ

ピクセル値	0	32	64	96	128	160	192	224	255
明るさ									

所定の位置のピクセル値は Image.getpixel() 関数で取得が、Image.putpixel() で上書きができます（7.3 節）。上書きすれば、そこの点だけ画像が変化します（画像の 1 ピクセルが変わっても見た目にはわかりませんが）。

```
# ...続き
>>> img.getpixel((0, 0))                    # P(0, 0)の値を読む
0                                           # ピクセル値は0
>>> img.putpixel((0, 0), 255)               # P(0, 0)を255で上書きする
```

画像外側の位置の操作には IndexError エラーが上がります。たとえば、サイズ (1280, 961) の画像に対して、(2000, 1000) の位置を指定すると次のようにエラーとなります。

```
# ...続き
# 画像の外側の位置(2000, 1000)からピクセル値を読み出そうとすると例外が上がる
>>> img.getpixel((2000, 1000))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\tools\python3.10.2\lib\site-packages\PIL\Image.py",
   line 1411, in getpixel
    return self.im.getpixel(xy)
IndexError: image index out of range
```

ただし、ImageDraw モジュール（第 6 章）から利用する各種の描画関数は、範囲外に描画を試みてもエラーにはなりません。その代わり、範囲外の操作は無視されます。

```
# ...続き
>>> from PIL import ImageDraw               # ImageDrawのインポート
>>> draw = ImageDraw.Draw(img)              # 描画準備
>>> draw.line((0, 0, 2000, 1000), fill=255) # 範囲外に線を描く。エラーなし。
```

1.3 カラーピクセル

モノクロ画像のピクセルに割り当てられる数値は 1 つだけですが、カラー画像では 3 つの値が割り当てられます。P(x, y) の値はこのとき (r, g, b) のようにタプルで表現されます。

これら 3 つの値の解釈は用いるカラーモードで異なります。どのカラーモードで画像を表現していても、Pillow は画像オブジェクトを適切に表示できます。

以下、個々のカラーモードを説明します。

1.3.1 RGB

RGB モードは最も一般的なカラーモードです。ピクセル値は (r, g, b) の 3 要素タプルで示され、1 番目の要素が赤の、2 番目が緑の、3 番目が青の明るさ（強度）をそれぞれ示します。モノクロ同様、値の範囲はいずれも 0 から 255 です。

```
# ...続き
>>> img_color = Image.open('BlackCat.jpg')          # カラー画像を読み込む
>>> img_color.getpixel((0, 0))                      # (0, 0)のピクセル値はタプル
(105, 100, 97)                                      # 赤=105、緑=100、青=97
```

それぞれの強度を示す赤、緑、青の値を混ぜれば、いろいろな色が表現できます。下図に示すように、赤と青を混ぜれば紫が、赤と緑を混ぜれば黄色が、全部混ぜれば白が得られます。混色のパターンは全部で 256 × 256 × 256 あるので、これで 16,777,216（約 1,600 万）色が表現できます。これを TrueColor と言います。

図 1.3 ● RGB の混色

Pillow の RGB モードでは、整数要素 3 つのタプル以外にも、HTML で用いる色名や #123456 などの # で始まる 16 進数表記の文字列からも色を指定できます（5.1 節）。タプル表記と 16 進表記は、次のようにリスト内包表記を使えば 1 行で交互に変換できます。

```
# 16進表記からタプル
>>> html = '#F0F8FF'
>>> tuple([int(html[i:i+2], 16) for i in range(1, len(html), 2)])
(240, 248, 255)

# タプルから16進表記
>>> rgb = (240, 248, 255)
>>> '#' + ''.join([f'{c:02X}' for c in rgb])
'#F0F8FF'
```

リスト内包表記はループを簡潔に書けるのでとても便利です（付録 B.1）。

f'...' はフォーマット済み文字リテラルで、変数を含んだ文字列を生成する表記です（付録 B.2）。

1.3.2 HSV

HSV モードのピクセル値タプルは (h, s, v) です。1 番目の要素が色相（Hue）の円環上の角度を、2 番目と 3 番目がそれぞれ彩度（Saturation）と値（Value）の割合を示します。

色相は、0°から 360°の角度で示されます。色の配分は下図に示すように、時計で 12 時に位置する 0°の赤から始まり、角度が進むにつれ橙、黄、緑、シアン（この辺で 180°）、青、紫と変遷し、360°でもとの赤に戻ります。虹と同じ順番です。

図 1.4 ● HSV の色相

ただし、角度値は 0 ～ 255 にスケーリングされます。たとえば、180°のシアンはちょうど中間なので 127（255*180//360）です。

除算結果の小数点以下の切り捨ては int() もよいですが、整数除算の // もシンプルで素敵です（付録 B.3）。

彩度と値で用いられる割合は 0% ～ 100% の範囲ですが、こちらも 0 ～ 255 にスケーリングされます。たとえば、50% は 127 です。

```
# ...続き
# 彩度、値共に100%のシアンで全体を染めた100×100の画像を生成。
# この色はRGBモードでは(0, 255, 255)。
>>> img_RGB = Image.new('RGB', (100, 100), color='cyan')
>>> img_RGB.getpixel((0, 0))
(0, 255, 255)

# 上記をHSVに変換し、同じ位置のピクセル値を確認。
# (180°、100%、100%)は(127, 255, 255)になる。
>>> img_HSV = img.convert('HSV')
>>> img_HSV.getpixel((0, 0))
(127, 255, 255)
```

新規に画像を生成する Image.new() 関数は 4.1 節で、カラーモードを変換する Image.convert() 関数は 5.4 節でそれぞれ説明します。

色名は利用できません。名称に対応するRGBのタプルの並びを、そのままHSVと解釈するからです。たとえば、シアンはRGBでは(0, 255, 255)ですが、この並びをHSVで解釈すると色相が0°（つまり赤）、彩度と値が100%でとても明るくて鮮やかという意味になります。

```
# ...続き
# いきなりHSVを作成。色はCyanなので（0，255，255）のタプル
>>> img_wrongcolor = Image.new('HSV', (100, 100), color='cyan')
# H=0，S=255，V=255 は赤。試してください。
>>> img_wrongcolor.show()
```

1.3.3 CMYK

CMYKモードのピクセル値のタプルは(c, m, y, k)の4要素です。それぞれC（シアン）、M（マゼンタ）、Y（イエロー）、K（キー）です。値はRGBと同じく強度を示します。CMYがいずれも強度0なら白、すべて255なら黒が得られます（RGBと逆です）。

図1.5 ● CMYKの混色

CMYKは実際にインクを混ぜる印刷関係で用いられます。カラープリンタのインクカートリッジにシアンとマゼンタと黄色と黒があるのは、CMYKカラーモデルを採用しているからです。理論上、CとMとYの3要素をすべて混ぜれば、上図の3色の重なりが示すように漆黒が得られます。しかし、実際の印刷では、いろいろな絵の具を混ぜたとき同じように、汚い濃い色になるだけでピュアな黒にはなりません。そこで、黒の値だけは第4の色のKで表現するようになりました。

バーチャル世界のコンピュータグラフィックスの混色式にはKが含まれていないため、RGBやHSVからCMYKに変換しても、Kバンドにデータは生成されません。

```
# ...続き
# 1.3.1節のRGBカラー画像のimg_colorをCMYKに変換
>>> img_cmyk = img_color.convert('CMYK')

# CMYKのうちKだけを取り出し、その最小最大値を見る。
# どちらも0なので、すべてのデータが0ということがわかる。
>>> img_cmyk.getchannel('K').getextrema()
(0, 0)
```

上記の例で、カラー画像から K のデータだけを抜き出す Image.channel() 関数は 5.5.2 節で、画像からピクセルの最小値と最大値を取得する Image.getextrema() 関数は 3.3 節でそれぞれ説明します。

グラフィックス的には、K は HSV の V と同じような働きをします。K が小さいほど色が明るく、大きくなるにつれ次第に濃くなっていき、最後の 255 ではどんな色が指定されていても真っ黒になります（全体を黒インクで塗りつぶすから）。

1.3.4 アルファ

前景画像を背景画像に重畳したとき前景から背景が透けて見える透過画像には、前景画像に透過情報が備わっています。透過情報もピクセル単位なので、それだけで 1 色と同じ情報が必要です。したがって、透過付きカラー画像のピクセル値 P(x, y) に割り当てられるタプルは、(r, g, b, a) のように 4 要素です。

```
# ...続き
>>> img_alpha = Image.open('Smily-RGBA.png')              # 透過PNGを読み込む

# 4要素のタプルが得られる。
# 末尾の要素の値が0なので、位置(0, 0)のピクセルはもとが白でも完全に透過。
>>> img_alpha.getpixel((0, 0))
(255, 255, 255, 0)
```

透過情報は一般にアルファチャネルと呼ばれますが、後述の事情から、Pillow の用語ではアルファ「バンド」となります。

値は透過の度合を割合から示します。最小値の 0% は完全な透過（下が完全に透けて見える）、最大値の 100% が完全な不透過（下が完全に覆われる）です。値が大きいほど下が見えないので、

透過率（transparency）よりは不透過率（opacity）と言った方がしっくりくるでしょう。タプル中の値は HSV の割合同様、0 から 255 にスケーリングされます。

1.3.5 カラーパレット

カラー画像は 1 ピクセルあたり (r, g, b) のように 3 要素、1 要素 1 バイトを占有するので、計 3 バイトを消費するのが一般的です。画像全体分なら `image.width * image.height * 3` バイトです。

GIF はこれとは異なる方法でピクセル値を管理することでデータ量をおよそ 2/3 に削減します。

これには、まず色 (r, g, b) とそれを指し示すインデックス番号のテーブルを用意します。そして、ピクセルには色そのものではなくこのインデックス番号を書き込みます。表示時にはインデックス番号に対応する色を参照して描画します。

テーブルの例を示します。

表 1.1 ● GIF カラーパレットの例

番号	色（RGB 順）
0	(2, 2, 2)
1	(127, 129, 22)
2	(137, 137, 137)
⋮	⋮
254	(255, 245, 218)
255	(154, 154, 154)

インデックス番号は値 1 つだけ、1 ピクセルあたり 1 バイトしか使わないので、通常の 3 色 3 バイトの 1/3 しかメモリを占有しません。その代わり、使える番号が 0 から 255 までなため、表現できる色が（どのような色の組み合わせでも構いませんが）最大 256 色に制限されます。

この「インデックス番号 = 色」の対応表をカラーパレット（color palette）と言います。他所ではカラーテーブル（color table）とも呼ばれます。GIF 以外では、PNG にもカラーパレットを用いるモードがあります（インデックス番号が 8 ビットものはしばしば PNG-8 と呼ばれます）。

Pillow にはカラーパレット操作の属性や関数がいくつか用意されています。これらを使えば、背景を透過にしたり、アルファバンドに頼らずとも背景色だけ入れ替えることができます（5.2 節）。