訳者まえがき

本書は『Automate the Boring Stuff with Python ── Practical Programming for Total Beginners』の日本語訳です。原書の副題にあるように、まったくの初心者にもわかるように、Pythonの基本的な文法から説明し、実用的なプログラミングを紹介しています。

本書の特徴は、まずその実用性です。WordやExcel文書の処理のほかに、PDF文書の処理、Webサイトからのダウンロード、メールやSMSの送受信、画像処理といった、日常業務でよく直面する面倒で退屈な作業を事例にしています。最後には汎用性の高いGUIオートメーションについても説明しています。

本書を読むと「こんなことも自動化できるんだ！」と気づくことが多いでしょう。劇的な「業務効率化」「コスト削減」「生産性向上」を達成するには、単純な繰り返し作業を自動化するにかぎります。Pythonと豊富なモジュールを使えば、簡単なプログラミングで作業を自動化できるのです。

ワンランク上のハイパフォーマンスなホワイトカラーを目指すには、プログラミングを覚え、有名なツールやモジュールの使い方を覚え、業務を分析して、自動化できる作業は可能なかぎり自動化するという意識をもつことが重要です。人をつぎ込めばよいといった安易な人海戦術は避け、自動化が困難な人間臭い作業にこそ貴重なヒューマンリソースをつぎ込み、仲間の仕事のやりがいを高め、仕事全体の質を高めていく。本書はそのような意識をもつ方に、具体的な手順と事例を提供する良書です。本書掲載のサンプルプログラムは、いずれも短くて理解しやすいので、自分の業務に合わせて簡単に改造できるでしょう。既存のものを模倣するのも、効率化のひとつの方法です。ぜひ実際にコードを打ち込んで体験しながら読み進めてください。

本書を手に取る方はノンプログラマーが多いと思いますが、Pythonに詳しいプログラマーにも役立つ面があります。プログラマーはプレーンテキストが好きで、シンプル

なコマンドラインツールを愛し、WordやExcelのようなぶくぶく太ったアプリを毛嫌いしているものです。けれども業務上やむをえず、WordやExcel文書でデータをやり取りしなければならないこともあるでしょう。「ネ申（神）Excel」に代表されるように、社外や他部署から提供されるデータファイルや、提出すべきレポートや申請書が、業務効率を配慮した形式になっていないことがよくあります。そんなとき、本書で紹介しているモジュールを利用してWordやExcel文書からデータを取り出し、自分の好きなデータ形式に変換したりデータベースに登録してしまえば、仕事がやりやすくなります。そして最後にまたWordやExcel文書として書き出せばよいのです。

なお、本書には、クラスや、ジェネレータ、リスト内包表記といったPythonの高度な機能の説明はありません。再利用可能で巨大で複雑なプログラムより、その場限りかもしれないけれど、シンプルで短いプログラムを作って、さっさと仕事を終わらせてしまうことが目的だからです。

また、開発環境として最もシンプルなIDLEを採用しています。本書のサンプルプログラムのように1つのファイルで完結する短いプログラムなら、IDLEで十分だと思います。Pythonの開発環境には、他にPyDev、PyCharm、Spyderなどさまざまありますが、どちらかというと大規模なプログラムを組むプログラマー向けのものです。本書の読者には、次のステップとしてJupyter notebookをお勧めします。Jupyter notebookは、Webブラウザ上で対話的にプログラミングができます。コードと実行結果の記録をとっておけるので、レポート作成にも使えるし、再編集も容易です。

翻訳にあたっては、原書にあったミスを訂正したほか、コーディングスタイルをキャメルケースから標準的な下線つなぎに変更したり、%による文字列フォーマットを`format()`メソッドに書き換えたりしています。また、各種モジュールの最新バージョンの仕様に合うように修正しています。特にExcel文書を読み書きするOpenPyXLのバージョンアップは激しく、機能がますます充実しています。Twilioは日本のサイトでの申し込みと日本国内での動作を確認しています。

サンプルコードの文字列の一部を日本語化しています。訳者個人としては基本的にコードの中には英語しか書かない方針なのですが、ノンプログラマーの方により身近に感じていただくためと、サンプルプログラムの日本語対応状況を調べるために日本語化しました。コード中に日本語を用いる場合の注意点として、うっかり全角スペースで字下げしたり、文字列を囲むのに全角のシングルクォートやダブルクォートを使ったり、引数の区切りに全角のカンマを使ってしまうことがあります。エラーの原因が見た目に

わかりにくいので注意してください。また、ファイルの文字コードはUTF-8である必要があります。IDLEで編集する場合は自動でUTF-8が選ばれますが、メモ帳など他のエディタで編集する場合には文字コードに注意してください。

それでは、本書を通じてより多くの人が自動化の恩恵を受けられ、退屈な作業から解放されることを期待します。

2017年4月19日
相川 愛三

まえがき

「3人がかりで2日かかっていた作業を、2時間でやっちゃったね。」

筆者の大学時代のルームメイトは、2000年代初頭に家電量販店で働いていました。お店はときどき競合店の販売価格が何千件も掲載された表を受け取っていました。それを印刷すると紙の山ができました。店員3人で分担して、商品ひとつずつについて自店舗の価格を探し、競合店のほうが安い商品をすべてリストアップしていました。通常、この作業に2日かかっていました。

床の上に座って膨大な紙の山と格闘している彼らを見ながら、ルームメイトは言いました。

「もしその印刷物の元ファイルがあれば、ぼくならプログラムを組んで処理しちゃうんだけど。」

2時間後、競合店の価格を読み込み、自店舗のデータベースから商品を検索し、競合店のほうが安いかどうかを記録する短いプログラムができました。彼はプログラミングの初心者だったので、プログラミング本から説明を探すのに多くの時間を費やしていました。プログラムを実行してみると、たった数秒で処理が終わりました。その日はルームメイトとお店の仲間は昼食時間をたっぷりとることができたそうです。

これがコンピュータプログラミングの威力です。コンピュータは、スイス・アーミーナイフのように、ありとあらゆる仕事に合わせて仕立て上げることができます。多くの人が何時間もクリックやキー入力をして反復作業をしていますが、目の前の機械に正しく指示すれば数秒で作業を完了できることに気づいていないのです。

本書の対象読者

今日用いられる多くのツールの中心にあるのはソフトウェアです。多くの人がSNSを使ってやり取りし、インターネットに接続可能な電話を持ち、オフィスではコンピュータを使って仕事をしています。その結果、プログラムを書ける人材の需要は急騰しています。数多くの本や、インタラクティブなWebチュートリアル、開発者のブートキャンプがあり、大志を抱いた初心者を年収1,000万円のソフトウェアエンジニアに育成すると謳っています。

しかしながら、本書はそういう一部の人のためのものではなく、ふつうの人のための本です。

本書は読者をプロのソフトウェア開発者に養成するものではありません。ギターのレッスンを数回受けたところで、ロックスターになれるわけではないのと同じです。けれども、事務職や管理職の人、学校、会社、自宅で仕事や娯楽のためにコンピュータを使う人が本書を読めば、プログラミングの基本を覚えて、次のような単純作業を自動化できるようになるでしょう。

- 何千個ものファイルを移動し、名前を変え、フォルダに分ける
- オンラインのフォーム（申し込み画面）を、キーボードを使わずに入力する
- Webサイトが更新されたら、自動的にファイルをダウンロードしたりテキストをコピーする
- コンピュータから通知メールを送ってもらう
- Excelのスプレッドシートを更新・整形する
- 電子メールをチェックして、事前に用意しておいた返事を送信する

こういった作業は単純ですが、人手で処理すると時間がかかるものです。また、単純すぎたり用途が限られたりするので、こういった作業をやってくれる市販ソフトウェアはなかなかありません。しかし、プログラミングの知識を少し身につければ、面倒な作業をコンピュータにやらせることができるようになるのです。

本書の構成

本書はリファレンスマニュアルではなく、初心者のための入門書です。推奨のコーディングスタイルに反することがありますが（例えば、グローバル変数の使用）、その代

わりコードが簡単で習得しやすくなっています。その場限りのコードを書く人向けの本なので、コードのスタイルや美しさにはあまり配慮していません。オブジェクト指向プログラミングやリスト内包、ジェネレータなど、洗練されたプログラミングの考え方は、複雑になるので説明していません。ベテランのプログラマーなら本書のコードを効率のよいものに改良できるでしょうが、本書は最小限の労力でプログラムを動作させることを重視します。

プログラミングとは何か？

プログラマーといえば、ドラマや映画では、不可解な0と1の並びをものすごい勢いで画面いっぱいに入力しているのを見かけますが、現実のプログラマーはそんな変な生き物ではありません。**プログラミング**とは、コンピュータに処理をさせるための命令を入力する行為にすぎません。命令によって、数字を高速に処理し、テキストを変換し、ファイルの中の情報を検索し、インターネットを通じて他のコンピュータと通信するわけです。

プログラムは、構成要素として基本的な命令群を用います。最も一般的な構成要素を言葉で書き下すと、こうなります。

「これを実行せよ。次にあれを実行せよ。」
「もしこの条件が成り立つなら、これを実行せよ。そうでなれけば、あれを実行せよ。」
「指定の回数だけ、これを実行せよ。」
「この条件が成り立つまで、あれを繰り返せ。」

以上の構成要素を組み合わせれば、より複雑な処理を実装することができます。例として、**ソースコード**というプログラミングの命令群を示します。これはPythonというプログラミング言語で書かれた簡単なプログラムです。Pythonのソフトウェアは、ソースコードの先頭から末尾まで、1行ずつ順番に実行していきます（ただし、条件式が成り立つときにのみに実行される部分や、成り立たないときに実行される部分もあります）。

```
password_file = open('SecretPasswordFile.txt')  # ❶
secret_password = password_file.read()  # ❷
print('パスワードを入力して下さい。')  # ❸
typed_password = input()  # ❹
if typed_password == secret_password:  # ❺
    print('認証されました。')  # ❻
    if typed_password == '12345':  # ❼
        print('パスワードは脆弱です。')  # ❽
else:  # ❾
    print('アクセスが拒否されました。')  # ❿
```

プログラミングについて何も知らなくても、上記のコードを読めば、何が起こるのかおぼろげながらわかるのではないでしょうか。まず、❶SecretPasswordFile.txtというファイルを開き（open）、❷秘密のパスワード（secret_password）を読み出し（read）ます。次に、❸ユーザーにパスワードをキーボード入力するように表示（print）し、❹入力（input）されたパスワードをtyped_passwordに入れます。❺2つのパスワードを比較し、❻もし（if）同じであれば、「認証されました。」と表示します。❼続いてパスワードが「12345」かどうかを調べ、❽もしそうならパスワードが不適切であるメッセージを表示します。❾パスワードが同じでなければ（else）、❿「アクセスが拒否されました。」と表示します。

Pythonとは何か

Pythonとは、プログラミング言語Pythonのことであり、正しいPythonのコードとみなすための文法が定められています。Pythonのインタプリタというソフトウェアは、Python言語で書かれたソースコードを読み込んで、命令を実行します。Pythonインタプリタは無料でダウンロードでき、Linux、Mac、Windows版が存在します。

Pythonという名前は、イギリスの喜劇集団「モンティ・パイソン」に由来し、蛇の名前に由来するものではありません。Pythonのプログラマーは、愛情を込めて「パイソニスタ」と呼ばれます。モンティ・パイソンや蛇にちなんだ表現は、Pythonのチュートリアルやドキュメントのあちこちにちりばめられています[*1]。

*1　訳注：本書でも、42やSPAMなど、モンティ・パイソンに関連する言葉が使われています。

プログラマーは数学にそれほど詳しくなくてもよい

プログラミングを習得するうえでよく耳にする懸念点は、数学を多用するのではないか、と思われていることです。実際のところ、ほとんどのプログラミングにおいては、算数レベルの簡単な数学で十分です。プログラミングが得意であることと、数独パズルを解くのが得意であることは、それほど違いがありません。

数独は、9×9のマスを1から9の数字を埋めるパズルですが、各行と各列と3×3のマスの中では、同じ数字を使ってはいけません。最初に与えられた数字から、推論を重ねて解いていきます。**図1**の例では、左上に5があるので、一番上の行と、一番左の列と、左上の3×3の太枠の正方形の中では、他に5を使うことができません。1行、あるいは、1列、あるいは、1つの太枠正方形の中をひとつ完成させると、パズルの残りの部分のヒントになっていきます。

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

図1　数独の問題（左）と、答え（右）。数字を使うが、数学はあまり使わない（出典：©Wikimedia Commons）

数独には数字が使われているからといって、数独を解くのに数学が得意でなければならないわけではありません。プログラミングも同じです。数独を解くときのように、プログラムを書くときには、問題を細かな1つずつのステップに分解していきます。同様に、プログラムを**デバッグ**するとき、すなわち、プログラムの間違い（バグ）を見つけて直すときにも、プログラムが何をしているのかをひとつずつ辛抱強く観察してバグの原因を見つけます。どんなスキルでもそうですが、練習すればそれだけ上達します。

プログラミングは創造的な活動

プログラミングは、レゴブロックでお城を作るような創造的な活動です。どんな外観のお城を作りたいか基本構想を抱き、手持ちのブロックを確認します。それから制作に取りかかります。プログラムを作り上げたら、お城と同じように、ちょっとおめかしすることもあります。

プログラミングが他の創造的な活動と異なる点は、プログラミングに必要な材料がコンピュータの中にすべて用意されているということです。キャンバスや絵の具やフィルムや糸やレゴブロックや電子部品を買いに行く必要がありません。プログラムを書いたら、オンラインで世界中に共有することもできます。プログラミングをしていると、間違えることもたくさんあるでしょうが、とても楽しい活動なのです。

本書について

本書の前半では、Pythonプログラミングの基本的な考え方について説明し、後半では、コンピュータを使って自動化できるさまざまな業務について説明します。後半の各章では、プログラム開発のプロジェクトに従って学べるようになっています。各章の概要を次に示します。

第Ⅰ部 Pythonプログラミングの基本

1章 Pythonの基本

計算式と、Pythonの最も基本的な命令と、コードを試すために必要なPythonのインタラクティブシェルというソフトウェアの使い方を説明します。

2章 フロー制御

条件の変化に賢く追従できるように、どの命令を実行するのかをプログラムに判断させる方法を説明します。

3章 関数

コードを管理しやすくまとめるために、自分で関数を定義する方法を説明します。

4章 リスト

リストというデータ型を使ってデータをまとめる方法を説明します。

5章 辞書とデータ構造

辞書というデータ型を使って、データを構造化する非常に強力な方法を説明します。

6章 文字列の操作

テキストデータ（Pythonでは**文字列**といいます）の処理方法を説明します。

第Ⅱ部 処理の自動化

7章 正規表現を使ったパターンマッチ

Pythonでの文字列操作法と、正規表現を使ったテキストのパターンの検索について説明します。

8章 ファイルの読み書き

プログラムからテキストファイルの内容を読み込んだり、ハードディスク上のファイルに情報を保存したりする方法を説明します。

9章 ファイル管理

Pythonを使って人間よりもはるかに速く、大量のファイルをコピー・移動・名前の変更・削除する方法を説明します。ファイルの圧縮と展開についても説明します。

10章 デバッグ

バグを見つけて修正するためのPythonのさまざまなツールの使い方を説明します。

11章 Webスクレイピング

Webページを自動的にダウンロードし、内容を解析して情報を抽出するプログラムを作ります。これを**Webスクレイピング**といいます。

12章 Excelスプレッドシートの操作

Excelのスプレッドシートをプログラムから操作する方法を説明します。Excelから読み込む必要がなくなるので、何百、何千ものExcel文書を処理するときに便利です。

13章 PDFとWord文書の操作

PDFとWord文書をプログラムから読み込む方法を説明します。

14章 CSVファイルとJSONデータの操作

CSVとJSONファイルをプログラムから操作する方法を説明します。

15章 時間管理、タスクのスケジューリング、プログラム起動

Pythonのプログラムでの日付と時間の扱い方と、ある時刻になったらコンピュータに処理を実行させるスケジューリング方法を説明します。Pythonプログラムから、Python以外のプログラムを起動する方法についても説明します。

16章 電子メールとSMSの送信

ユーザーに代わって電子メールとSMSを送信するプログラムの書き方を説明します。

17章 画像の操作

JPEGやPNGファイルなどの画像をプログラムから操作する方法を説明します。

18章 GUI自動化によるキーボードとマウスの制御

キーボードとマウスをプログラムから操作して、キー入力とクリックを自動化する方法について説明します。

Pythonのダウンロードとインストール

WindowsとMac、Ubuntu用のPythonは、本家http://python.org/downloads/から無料でダウンロードできます（**図2**）。

このWebサイトから最新版をダウンロードすれば、本書のすべてのプログラムを試せます。あるいは、さまざまなライブラリがパッケージ化されたディストリビューションを用いるのも便利です[*1]。

Python 3（例えば3.6.0）を使ってください。本書のプログラムはPython 3で動作するように書かれているため、Python 2では正しく、あるいは、まったく動作しないかもしれません。

＊1　訳注：特に64ビットのWindowsでPythonを使う場合は、Anacondaがお勧めです。本章の末尾に、Anacondaのインストール方法を示します。

図2　Pythonのダウンロードサイト

ダウンロードページには、64ビット用と32ビット用のインストーラがあるので、まず自分のコンピュータがどちらなのかを判断します。2007年以降に購入したコンピュータなら64ビットの可能性がありますが、それ以外は32ビットだと思われます。次のように確認しましょう。

Windows

コントロールパネルを開き、[システム]を開くと、[システムの種類]に64ビットか32ビットかが表示されています。

Mac

Appleメニューで［このMacについて］を選び、［詳しい情報］→［システムレポート］→［ハードウェア］→［プロセッサ名］を調べます。Intel Core SoloかIntel Core Duoであれば32ビットです。それ以外なら（Intel Core 2 Duoも含む）64ビットです。

Ubuntu Linux

ターミナルを開き、`uname -m`を実行します。`i686`なら32ビット、`x86_64`なら64ビットです。

Windowsでは、Pythonのインストーラ（ファイル名の拡張子が`.msi`）をダウンロードして開いてください。次のような画面指示に従ってPythonをインストールします。

1. ［Add Python 3.6 to PATH］にチェックを入れます。
2. ［Install Now］をクリックします。

MacではOSのバージョンに合った`.dmg`ファイルをダウンロードし、ダブルクリックします。表示されるインストーラ画面に従い、次のようにPythonをインストールします。

1. 新しいウィンドウにDMGパッケージが開いたら、`Python.mpkg`ファイルをダブルクリックし、管理者パスワードを入力
2. ［Continue］をクリックして［Welcome］セクションをスルーし、ライセンス条項を読んで［Agree］をクリック
3. ［Macintosh HD］（あるいはコンピュータのハードディスク名）を選択して、［Install］をクリック

Ubuntuでは、ターミナルを開いて次のようにPythonをインストールします。

1. ターミナルを開く
2. `sudo apt-get install python3`を入力
3. `sudo apt-get install idle3`を入力
4. `sudo apt-get install python3-pip`を入力

IDLEの起動

PythonインタプリタはPythonプログラムを実行するソフトウェアですが、IDLEという**インタラクティブ開発環境**は、ワープロのようにプログラムを入力するためのソフトウェアです。さっそくIDLEを起動してみましょう[*1]。

- Windows 7以前では、スタートメニューからIDLEを選択するか、IDLEを検索して実行してください。
- Windows 8以降では、IDLEを検索して実行してください。Win-Rを押し、「ファイル名を指定して実行」を開いて`idle`と入力して［OK］ボタンを押します。
- Macでは、Finderを開き、Applicationをクリックし、Python 3.6をクリックし、IDLEアイコンをダブルクリックします。
- Ubuntuでは、ターミナルを開いて、`idle3`と入力します。あるいは、画面左上のアプリケーションメニューをクリックして、プログラミングを選択し、IDLE3をクリックしてもよいです。

インタラクティブシェル

どのOSでもIDLEを起動すると、次のようなメッセージが表示された、ほとんど空白のウインドウが開きます。

```
Python 3.6.0 |Anaconda 4.3.0 (64-bit)| (default, Dec 23 2016, 11:57:41)
[MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

このウィンドウのことを**インタラクティブシェル**といいます。「シェル」とはコンピュータに命令を入力するプログラムのことでMacやWindowsのターミナルやコマンドプロンプトのようなものです[*2]。Pythonのインタラクティブシェルを使うと、命令を入力して、Pythonインタプリタに実行させることができます。コンピュータは入力された命令を読み込んで即座に実行するわけです。

例として、インタラクティブシェルの「`>>>`」（入力を促す「プロンプト」といいます）

＊1　訳注：IDLEは、モンテイ・パイソンのメンバーのEric Idleにちなんで名付けられています。

＊2　訳注：OSの「中心核＝kernel（カーネル）」に対して、カーネルを取り囲むインタフェースを「殻＝shell（シェル）」と呼びます。

の後に次のように入力してみましょう。

```
>>> print('Hello world!')
```

入力後にEnterキーを押すと、インタラクティブシェルは次のように表示するでしょう。

```
Hello world!
```

ヘルプの探し方

プログラミング上の問題を自力で解決するのは、意外と簡単です。信じられないでしょうから、わざとエラーを起こして確認してみましょう。インタラクティブシェルに`'42' + 3`と入力してください[*1]。これがどんな命令なのか、今は知る必要はありません。Enterキーを押すと、次のように表示されるでしょう。

```
>>> '42' + 3
Traceback (most recent call last):  # ❶
トレースバック(最新の呼び出しほど下にある)
  File "<pyshell#0>", line 1, in <module>
    '42' + 3
TypeError: Can't convert 'int' object to str implicitly  # ❷
型のエラー:int(整数)型のオブジェクトは文字列型に暗黙的に変換できない
>>>
```

❷のメッセージは、入力された命令をPythonが理解できなかったことを表しています。❶のトレースバック（問題の箇所まで関数の呼び出しをさかのぼった呼び出し履歴）には、問題が起こった箇所の命令と行番号が表示されています。エラーメッセージの内容がわからなければ、エラーメッセージを1文字たりとも変更しないでオンラインで検索しましょう。「`"TypeError: Can't convert 'int' object to str implicitly"`」と二重引用符（"）を付けて検索すれば、エラーメッセージの意味について大量の説明を検索することができます。例を**図3**に示します。

*1　訳注：「42」という数字は、『銀河ヒッチハイク・ガイド』というSFの「生命、宇宙、そして万物についての究極の疑問の答え」です。作者のダグラス・アダムスは、モンティ・パイソンの脚本家のひとりです。

図3　エラーメッセージをGoogleで検索するのはとても便利

だれか他の人が同じ質問をし、親切な人がそれに回答済みであることがよくあります。プログラミングについてありとあらゆることを知っている人などいません。ソフトウェア開発者の仕事といえば、技術的な問題の答えを探すことの毎日なのです。

望ましいプログラミングの質問とは

オンラインで答えが得られなかったときには、Stack Overflow（http://stackoverflow.com/）や、redditの「learn programming」（http://reddit.com/r/learnprogramming/）のようなWebフォーラムで質問してみましょう。けれども、回答してもらいやすくするには、望ましい質問のしかたがあります。Webサイトにある FAQ（よくある問答集）を読んで、正しい質問の方法を確認してください。

プログラミング上の質問をするには、次の点に気をつけましょう。

- 何をしたのかだけでなく、何をしたいのかを説明しましょう。間違った方法をとっているかどうかを判断しやすくなります。
- エラーの起こった箇所を特定しましょう。プログラムを実行するとすぐにエラーになるのか、あるいは、何らかの動作を行ったときにのみエラーになるのか。
- エラーメッセージとコードの全部を、http://pastebin.com/やhttp://gist.github.

com/にコピー&ペーストしましょう。

これらのWebサイトは、Webを通じて大量のコードを他人と共有でき、テキストの形式が崩れることもありません。投稿したコードのURLを、電子メールやフォーラムに記載します。例えば、筆者が投稿したコードのURLは、http://pastebin.com/SzP2DbFx/とhttps://gist.github.com/asweigart/6912168/です。

- 問題を解決しようとして試みたことを説明しましょう。自力で解決しようとしたことがわかります。
- 使っているPythonのバージョンを書きましょう。Pythonの2と3とでは、大きな違いがあるからです。OSの種類とバージョンも書きましょう。
- コードを変更後にエラーになったのなら、何を変更したのか正確に説明しましょう。
- プログラムを実行すると毎回エラーになるのか、あるいは、何らかの処理をした後だけにエラーが生じるのかを書きましょう。後者の場合は、どんな処理なのかも説明しましょう。

ネチケットには十分配慮しましょう。例えば、質問をすべて大文字にしたり、支援者に過大な要求をしたりするのはやめましょう。

Anacondaのインストール方法

本家のPythonサイトにあるWindows用のPythonは、32ビットWindows用です。64ビットWindowsでも動作はしますが、せっかくの広大なメモリー空間を活用できません。さらに、サードパーティーのパッケージをインストールするときに、別途C/C++コンパイラが必要になることがあり面倒です。そこで、前述のとおり、ライブラリとセットになったディストリビューションを用いると便利です。特に64ビットのWindowsではAnacondaの利用をお勧めします。以下にインストール手順を示します。

ダウンロード

Anacondaのダウンロードサイトhttps://www.continuum.io/downloads/をWebブラウザで開きます（**図4**）。

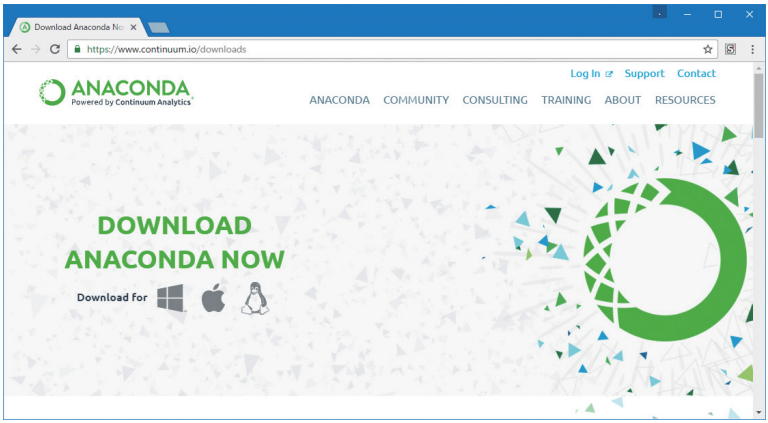

図4　Anacondaのダウンロードサイトを開きます

Windowsのアイコンをクリックすると、バージョンを選択できるので、Python 3.6の［64-BIT INSTALLER］をクリックすると、インストーラのダウンロードが始まります（**図5**）。

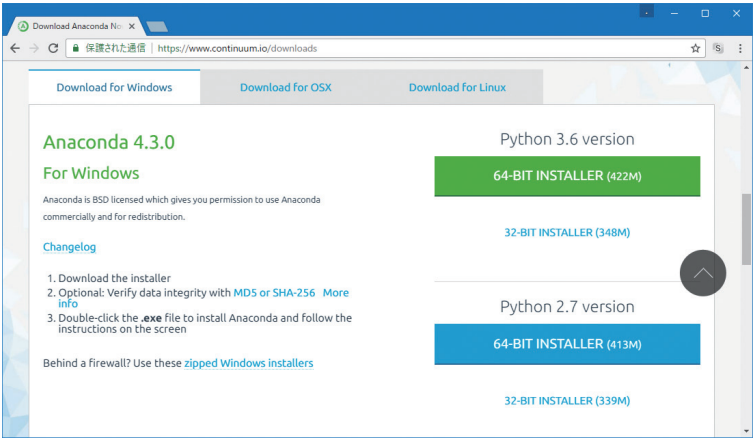

図5　バージョンを選びます

インストール

ダウンロードが完了したら、インストーラ`Anaconda3...exe`をクリックして実行します（**図6**）。

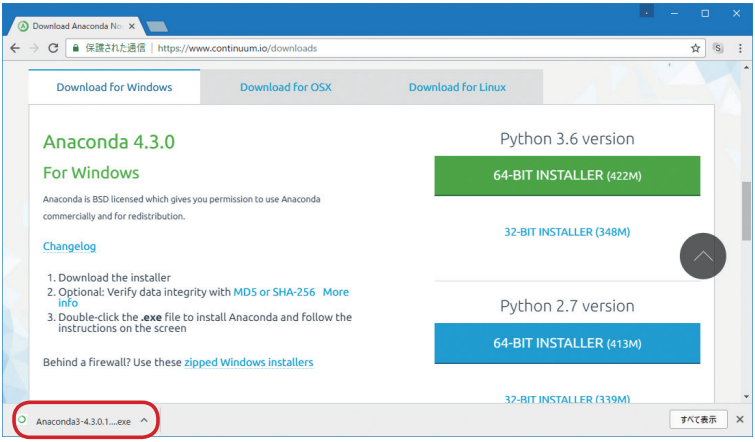

図6　インストーラを実行します

インストーラの起動画面で［Next］をクリックします（**図7**）。

図7　インストーラの起動画面

利用規約に目を通し、承諾できれば［I Agree］をクリックします（**図8**）。

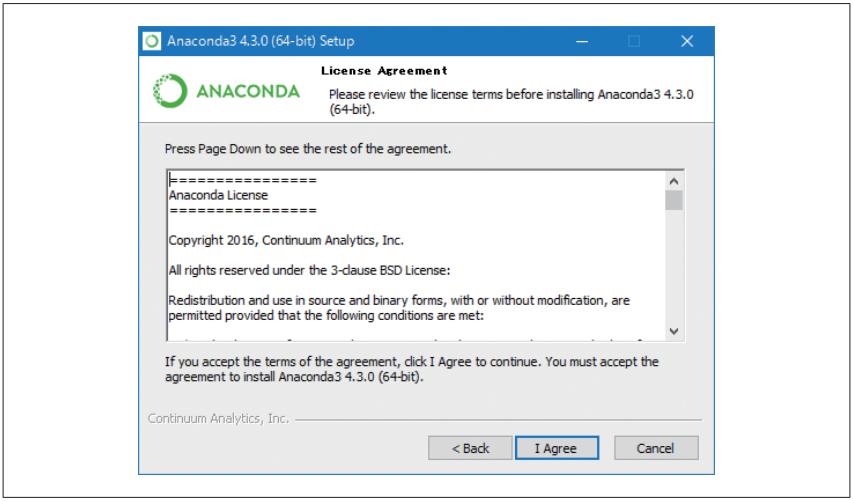

図8　利用規約を承認します

次に、利用者をだれにするのか決めます（**図9**）。パソコンの特権ユーザーであり、ひとりでしか使わない場合は、どちらでもかまいません。会社のパソコンであったり、共有パソコンの場合には、［Just Me］を選びます。［Next］をクリックします。

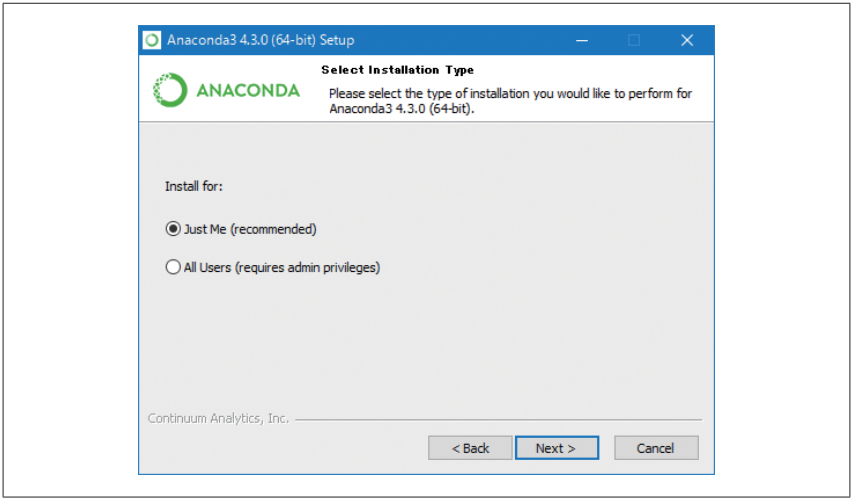

図9　利用者を決めます

インストール先のフォルダを決めますが、問題なければデフォルトのまま［Next］をクリックします（**図10**）。

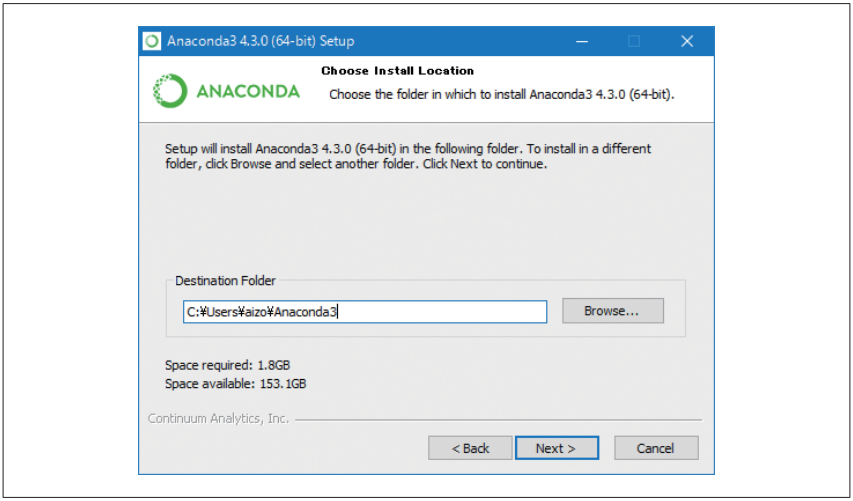

図10　インストール先フォルダを決めます

次の画面では、環境変数PATHを設定することと、AnacondaをデフォルトのPython処理系にすることを選べます。両方ともチェックを入れて［Install］をクリックします（**図11**）。

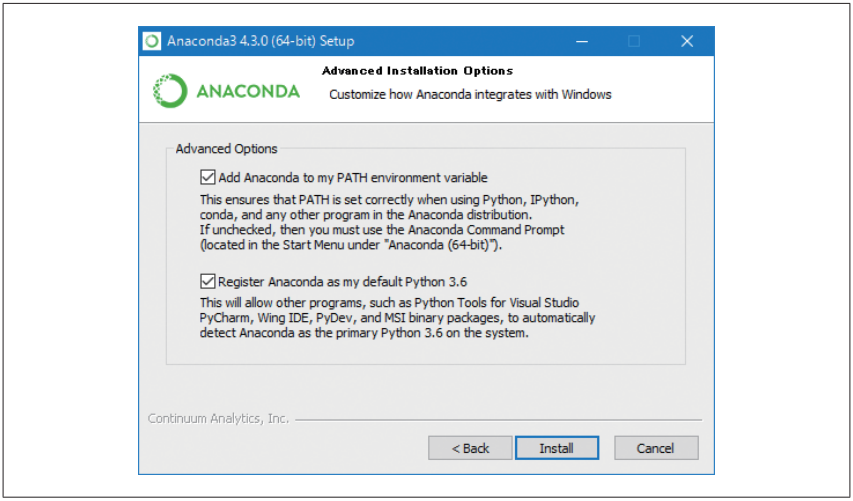

図11　オプションにチェックします

するとインストールが開始し、プログレスバーで進捗が表示されます（**図12**）。

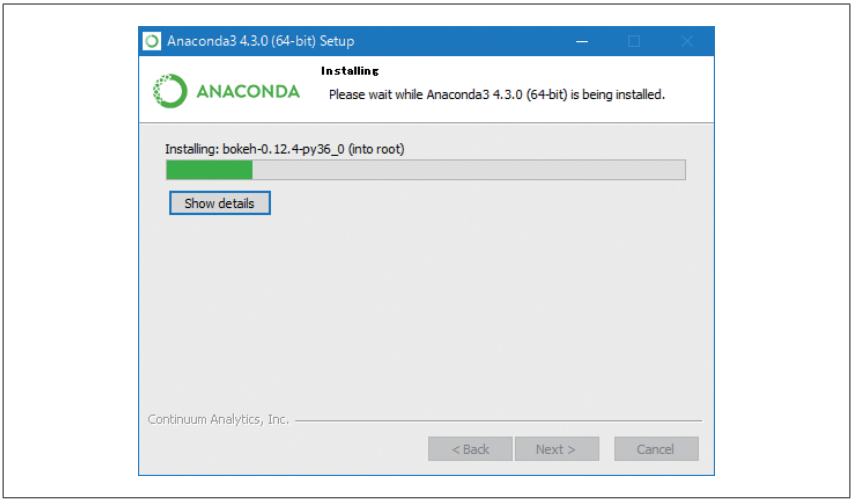

図12　インストール完了までしばらく待ちます

Anacondaには多数のライブラリが同梱されているので、完了までしばらく時間がかかります。完了したら［Next］ボタンを押します（**図13**）。

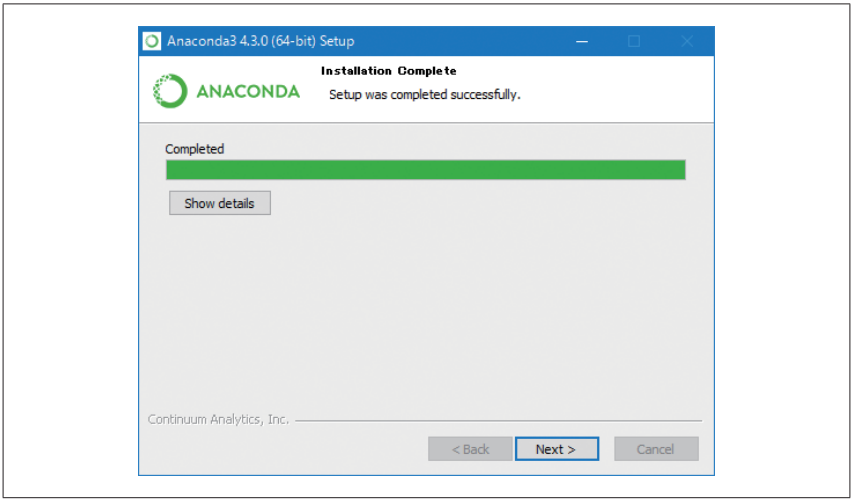

図13　インストール完了画面

最後の画面では、Anaconda Cloudの説明を開くかどうかのチェックボックスがありますが、不要であればチェックをはずして、［Finish］ボタンを押します（**図14**）。

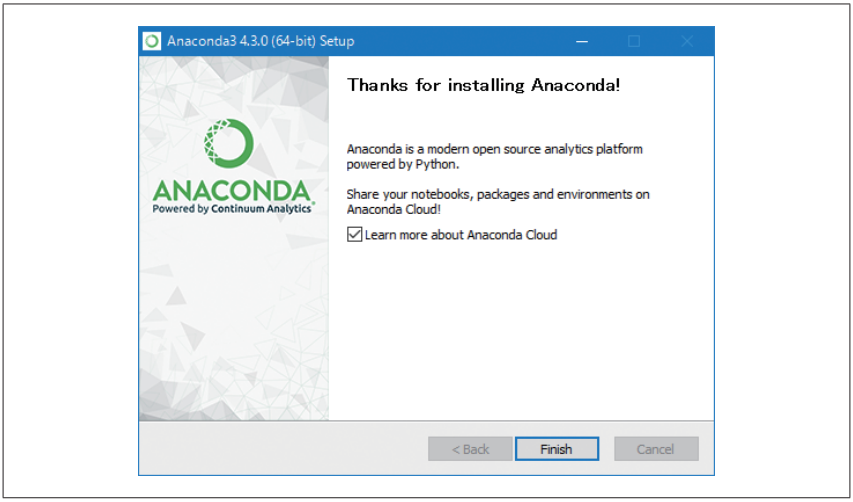

図14　完了画面

確認

正しくインストールできたかどうか確認してみましょう。Win−Rキーを押して「ファイル名を指定して実行」ダイアログを出します。そこに`idle`と入力して［OK］ボタンを押します（**図15**）。すると、黒いコマンドプロンプトのウィンドウが開き、しばらくすると白い`Python 3.6.0 Shell`というIDLEのインタラクティブシェルのウィンドウが開きます。

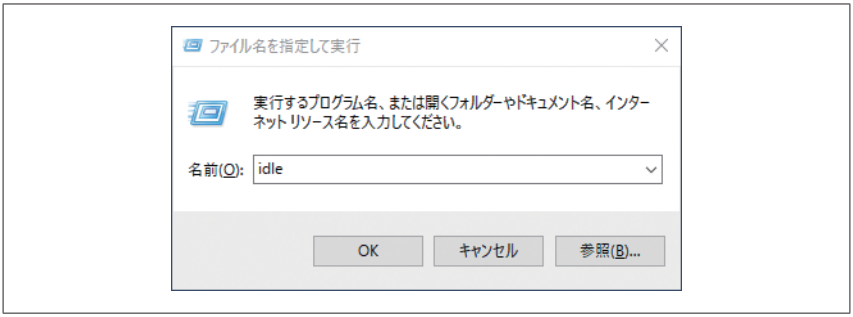

図15　Win-Rでファイル名を指定してidleを実行

IDLEのインタラクティブシェルは、スタートメニューからAnaconda Promptを実行して（**図16**）、Anacondaのプロンプトで`idle`と入力して起動することも可能です。

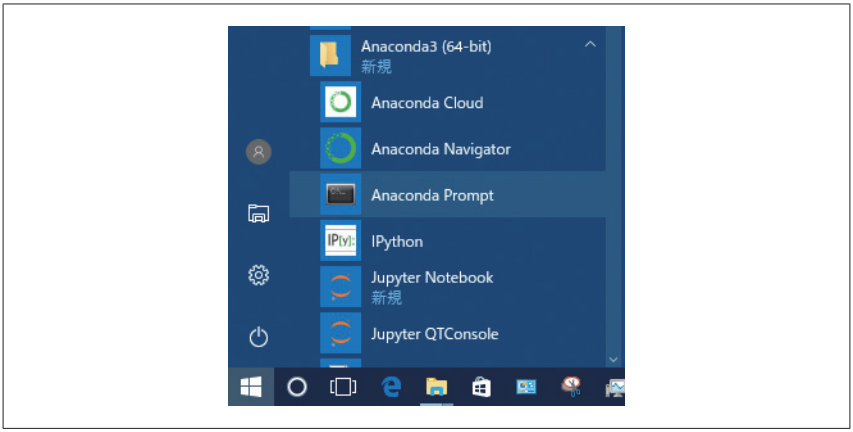

図16　スタートメニューからAnaconda Promptを実行

まとめ

ほとんどの人にとって、コンピュータは道具というより、ただの電化製品にすぎません。けれども、プログラミングを覚えれば、現代の最も強力な道具をもっと上手に使えるようになり、それが楽しくなることでしょう。プログラミングは脳の外科手術のような危険なものではありません。アマチュアにとって実験や試行錯誤ができる楽しいものなのです。

筆者はPythonのすばらしさをみんなにわかってほしいと思っています。筆者はブログ（http://inventwithpython.com/blog/）でプログラミングの入門記事を書いていますし、メール（`al@inventwithpython.com`）で質問も受け付けています。

本書は、プログラミングについての知識がゼロであることを想定していますが、本書に書かれていること以外にも質問したくなることがあるでしょう。効果的な質問をし、答えを見つける方法は、プログラミングを習得するうえで、とても重要であることを忘れないでください。

Let's begin!

意見と質問

本書（日本語翻訳版）の内容については、最大限の努力をもって検証、確認していますが、誤りや不正確な点、誤解や混乱を招くような表現、単純な誤植などに気がつかれることもあるかもしれません。そうした場合、今後の版で改善できるようお知らせいただければ幸いです。将来の改訂に関する提案なども歓迎いたします。連絡先は次のとおりです。

株式会社オライリー・ジャパン
電子メール japan@oreilly.co.jp

本書のWebページには次のアドレスでアクセスできます。

http://www.oreilly.co.jp/books/9784873117782
https://github.com/oreilly-japan/automatestuff-ja（日本語版サンプルコード）
https://www.nostarch.com/automatestuff（英語）
https://automatetheboringstuff.com（著者）

オライリーに関するそのほかの情報については、次のオライリーのWebサイトを参照してください。

http://www.oreilly.co.jp/
http://www.oreilly.com/（英語）

表記上のルール

本書では、次に示す表記上のルールに従います。

太字（Bold）

新しい用語、強調やキーワードフレーズを表します。

等幅（`Constant Width`）

プログラムのコード、コマンド、配列、要素、文、オプション、スイッチ、変数、属性、キー、関数、型、クラス、名前空間、メソッド、モジュール、プロパティ、パラメータ、値、オブジェクト、イベント、イベントハンドラ、XMLタグ、HTMLタグ、マクロ、ファイルの内容、コマンドからの出力を表します。

その断片（変数、関数、キーワードなど）を本文中から参照する場合にも使われます。

等幅太字（`Constant Width Bold`）

ユーザーが入力するコマンドやテキストを表します。コードを強調する場合にも使われます。

等幅イタリック（*`Constant Width Italic`*）

ユーザーの環境などに応じて置き換えなければならない文字列を表します。

ヒントや示唆、興味深い事柄に関する補足を表します。

ライブラリのバグやしばしば発生する問題などのような、注意あるいは警告を表します。

訳者による補足説明を表します。

謝辞

本書は多くの方々に支えられて執筆することができました。編集者のBill Pollock氏や、Laurel Chun、Leslie Shen、Greg Poulos、Jennifer Griffith-Delgadoなど、No Starch Press社のスタッフ各氏の並々ならぬ支援に感謝申し上げます。技術レビューアのAri Lacenski氏には、アドバイスや修正などの支援をいただき感謝します。

優しい終身の独裁者Guide van Rossum氏と、Python Software Foundationに関わる各氏には、優れた仕事に対して心より感謝いたします。Pythonのコミュニティーは、ハイテク業界で見つけた最高のものです。

最後に、家族と友人、そして、本書の執筆で忙しい生活のなか、おかまいなしで遊んだShotwellのお友達に感謝します。ありがとう！

目次

コラム目次

第Ⅰ部

Pythonプログラミングの基礎

1章
Python入門

プログラミング言語Pythonには、さまざまな構文や標準ライブラリ関数やインタラクティブな開発環境があります。本書ではあまり複雑なことに深入りせず、手軽で小さなプログラムを書くのに十分な知識を身につけることにします。

とはいえ、何をするにせよ基本的なプログラミングの考え方を覚えておく必要があります。こういった考え方は不可解で退屈に思われるかもしれませんが、あらかた知識を身につけて練習すれば、まるで魔法の杖のようにコンピュータに命令して、信じられないくらいの仕事をやってもらえるようになるでしょう。

本章では、いくつかの例をインタラクティブシェルに入力してみましょう。Pythonの命令をひとつずつ実行すると、すぐに結果がわかります。インタラクティブシェルを用いれば、Pythonの基本命令の働きを覚えるのに便利ですので、試しながら学んでいきましょう。本書を読むだけでなく実際に動かしてみるほうが、ずっと忘れにくいです。

1.1 式をインタラクティブシェルに入力する

IDLEを実行してインタラクティブシェルを起動しましょう。IDLEは序章でPythonといっしょにインストール済みです。Windowsでは、Win−Rを押し「ファイル名を指定して実行する」ダイアログで`idle`と入力してください。Macでは、［アプリケーション］→［MacPython 3.6］→［IDLE］を選択します。Ubuntuでは、端末を開き`idle3`と入力します。

`>>>`というプロンプトが表示されていれば、それがインタラクティブシェルです。Pythonに簡単な計算をさせてみましょう。`2 + 2`と入力します。

```
>>> 2 + 2
4
```

IDLEのウィンドウには次のように表示されるでしょう。

```
Python 3.6.0 |Anaconda 4.3.0 (64-bit)| (default, Dec 23 2016, 11:57:41)
[MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2 + 2
4
>>>
```

Pythonでは、`2 + 2`を**式**といい、最も基本的なプログラミング命令となっています。式は、2などの**値**と、+などの**演算子**から構成され、ひとつの値に**評価する**（つまり、式を計算して簡単にする）ことができます。したがって、Pythonで値を使えるところならどこでも式を用いることができるのです。

前述の例では、`2 + 2`を評価すると4というひとつの値になりました。演算子のないひとつの値は、評価しても変わりませんが、これもまた式とみなされます。

```
>>> 2
2
```

間違っても気にしない！

プログラムの中にコンピュータが理解できないコードが含まれていると、Pythonはエラーメッセージを表示して止まります。エラーメッセージが出ても、コンピュータが壊れたわけではないので、間違えても気にしないでください。単にプログラムが予期せず停止しただけですから。

エラーメッセージの内容を詳しく知りたければ、メッセージをそのものをWebで検索しましょう。http://inventwithpython.com/appendixd.html（英文）にも、よくあるPythonのエラーメッセージとその意味の一覧があるので参照してください。

Pythonの式で使える演算子は他にもたくさんあります。**表1-1**にPythonの数学演算子を示します。

表1-1 数学演算子

優先順位	演算子	説明	例	評価後の値
1.	**	累乗	2 ** 3	8
2.	*	掛け算	3 * 5	15
2.	/	割り算	22 / 8	2.75
2.	//	整数の割り算。小数点以下切り捨て	22 // 8	2
2.	%	剰余（割り算の余り）	22 % 8	6
6.	+	足し算	2 + 2	4
6.	-	引き算	5 - 2	3

Pythonの演算子の順序（**優先順位**ともいいます）は、通常の数学と同じです。累乗の**演算子が最初に評価され、次に*、/、//、%演算子が左から右に評価され、最後に+、-演算子が左から右に評価されます。必要なら、丸カッコ()を使って順序を変えることもできます。インタラクティブシェルに次のように式を入力してみましょう。

```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565878 * 578453
28093077826734
>>> 2 ** 8
256
>>> 23 / 7
3.2857142857142856
>>> 23 // 7
3
>>> 23 % 7
2
>>> 2 + 2
4
>>> (5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

それぞれの例において、式を入力してやるだけで、Pythonは面倒な計算をしてひとつの値を求めてくれます。**図1-1**に示すように、Pythonはひとつの値になるまで式の部分を繰り返し評価します。

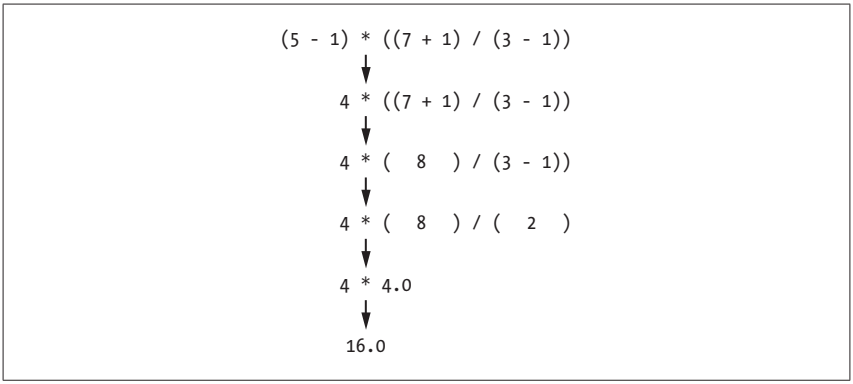

図1-1 ひとつの値になるまで式を評価する

演算子と値を組み合わせて式を作るルールは、プログラミング言語としてのPythonの基本です。人間どうしでもスムーズにコミュニケーションをとるには文法が大事なのと同じです。例えば、次の2文を見てください。

> これは文法的に正しい日本語の文です。
> これは文法的です正しくない文の日本語。

2文目は日本語の文法に合っていないので、理解するのが困難です。これとまったく同じように、Pythonの命令を間違って入力すると、Pythonは理解することができず、次のような`SyntaxError`（シンタックスエラー。文法エラー）のエラーメッセージを表示するのです。

```
>>> 5 +
  File "<stdin>", line 1      ファイル "<stdin>"、1行目
    5 +
      ^
SyntaxError: invalid syntax   文法エラー：不正な文法
>>> 42 + 5 + * 2
  File "<stdin>", line 1      ファイル "<stdin>"、1行目
    42 + 5 + * 2
             ^
SyntaxError: invalid syntax   文法エラー：不正な文法
```

インタラクティブシェルを使えば、命令を入力してうまく動作するかをいつでも確かめることができます。コンピュータが壊れてしまわないか心配する必要はありません。

最悪でもPythonがエラーメッセージを表示するだけです。プロのソフトウェア開発者であっても、コードを書いているときは、しょっちゅうエラーメッセージを見ているものです。

1.2 整数、浮動小数点数、文字列型

式とは、値と演算子が組み合わさったもので、必ずひとつの値に評価されるものでした。値のカテゴリーを**データ型**といい、すべての値はいずれかひとつのデータ型に属しています。Pythonの主なデータ型を**表1-2**に示します。例えば、`-2`や`30`の値は、**整数（int）型**の値です。整数型は、整数値を表すデータ型です。「イント」と読みます。一方、`3.14`のような小数点を含む数値は、**浮動小数点数（float）型**です。「フロート」と読みます。`42`という値は整数型ですが、`42.0`は浮動小数点数型になることに注意してください。

表1-2 主なデータ型

データ型	例
整数（int）	`-2`, `-1`, `0`, `1`, `2`, `3`, `4`, `5`
浮動小数点数（float）	`-1.25`, `-1.0`, `-0.5`, `0.0`, `0.5`, `1.0`, `1.25`
文字列（str）	`'a'`, `'aa'`, `'aaa'`, `'Hello!'`, `'11 cats'`

Pythonのプログラムは、**文字列（str）型**というテキストの値を持つこともできます（ストリング、ステア、ストルなどと読みます）。文字列は、Pythonが始まりと終わりを認識するように、シングルクォート（'）やダブルクォート（"）で囲んで記述します（例えば、`'Hello'`や`"さらば残酷な世界よ！"`と書きます）。1文字も使わない文字列を使うこともできます。`''`や`""`と書き、**空文字列**といいます。文字列については、4章で詳しく説明します。

「SyntaxError: EOL while scanning string literal（文字列リテラルを走査中にEOL（文の末尾）に到達した）」というエラーメッセージが表示されたなら、おそらく文字列のクォートの閉じ忘れが原因です。例を示します。

```
>>> 'Hello world!
SyntaxError: EOL while scanning string literal
```

1.3 文字列の連結と複製

演算子の意味は、それに続く値のデータ型によって変わります。例えば、+は2つの整数型や浮動小数点数型の値を足し合わせます。しかし、2つの文字列型の値に対しては、文字列をつなぐ**文字列連結**演算子として働くのです。インタラクティブシェルに次のように入力してください。

```
>>> 'Alice' + 'Bob'
'AliceBob'
```

式が評価されて、2つの文字列をつなげた文字列というひとつの値になります。では、もし+演算子を文字列型と整数の値に用いたらどうなるでしょうか？ Pythonはこの式の扱い方を知らないので、エラーメッセージを表示します[*1]。

```
>>> 'Alice' + 42
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    'Alice' + 42
TypeError: Can't convert 'int' object to str implicitly
型エラー:'int'型のオブジェクトをstrに暗黙的に変換できない
```

「Can't convert 'int' object to str implicitly ('int'型のオブジェクトをstrに暗黙的に変換できない)」というエラーメッセージは、整数と文字列`'Alice'`を連結しようとしたとPythonが考えたことを意味します。Pythonは自動的に型変換してくれないので、整数を文字列に明示的に変換する必要があります（「1.6 プログラムを分析する」で説明しますが、データ型の変換には、`str()`、`int()`、`float()`という関数を用います）。

*演算子は2つの整数や浮動小数点数の掛け算に用いますが、文字列と整数の間に用いると**文字列の複製**演算子になります。インタラクティブシェルに次のように入力して、文字列と数を掛け算してみましょう。

```
>>> 'Alice' * 5
'AliceAliceAliceAliceAlice'
```

式を評価すると、整数で指定した回数だけ元の文字列を繰り返した文字列になります。文字列の複製は便利な技ですが、文字列の連結ほどは使われません。

---

*1 訳注：JavaScriptなど、暗黙的に文字列型に変換されるプログラミング言語もあります。このことが見つけにくいバグの原因になることがあります。

*演算子は、2つの数値（掛け算）と、文字列と整数値（文字列の複製）だけに用いることができます。それ以外の場合には、Pythonはエラーメッセージを表示します。

```
>>> 'Alice' * 'Bob'
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    'Alice' * 'Bob'
TypeError: can't multiply sequence by non-int of type 'str'
型エラー：シーケンスと非整数型の'str'型と掛け算はできない [*1]
>>> 'Alice' * 5.0
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    'Alice' * 5.0
TypeError: can't multiply sequence by non-int of type 'float'
型エラー：シーケンスと非整数型の'float'型と掛け算はできない
```

Pythonはこれらの式を理解できないのも無理はありません。人間だって2つの単語を掛け算したり任意の文字列と小数を掛け算したりできないのですから。

1.4 変数に値を格納する

変数とはコンピュータのメモリー上にある箱みたいなもので、ひとつの値を格納することができます。プログラムにおいて、式の評価結果を後で使いたいときに、変数の中に保存しておけるのです。

1.4.1 代入文

変数に値を保存するには、**代入文**を用います。代入文は、変数名と等号=（**代入演算子**といいます）と、保存する値から構成されます。例えば、

```
spam = 42
```

のように代入文を入力すれば、spamという名前の変数に整数値の42が保存されるわけです[*2]。

図1-2のように、変数はラベルの付いた箱であり、そこに値が入っていると考えましょう。

---

*1 「シーケンス」とはデータが並んだデータ型であり、文字列やリスト、タプルなどの総称です。

*2 訳注：spamは、モンティ・パイソンのスケッチ（コント）にちなんでいます。spamを連呼することから、迷惑メールを表す「スパム」の由来といわれています。

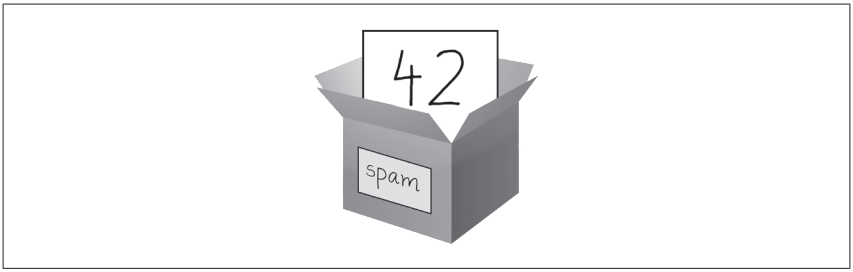

図1-2 spam = 2 とは「変数spamに整数値42を入れる」とプログラムに伝えるようなもの

例として、次のようにインタラクティブシェルに入力してみましょう。

```
>>> spam = 40  # ❶
>>> spam
40
>>> eggs = 2
>>> spam + eggs  # ❷
42
>>> spam + eggs + spam
82
>>> spam = spam + 2  # ❸
>>> spam
42
```

変数に初めて値が入ることを**初期化**（あるいは**生成**）といいます（❶）。その後は、式の中でこの変数を他の変数や値と組み合わせて使うことができます（❷）。変数に新しい値が代入されると（❸）、古い値は忘れ去られるので、この例ではspamは40の代わりに42という値になります。これを変数の**上書き**といいます。次のコードをインタラクティブシェルに入力に入力して、文字列を上書きしてみてください。

```
>>> spam = 'Hello'
>>> spam
'Hello'
>>> spam = 'Goodbye'
>>> spam
'Goodbye'
```

図1-3に示す箱のように、この例の変数spamには'Hello'が保存されていましたが、'Goodbye'に置き換わります。

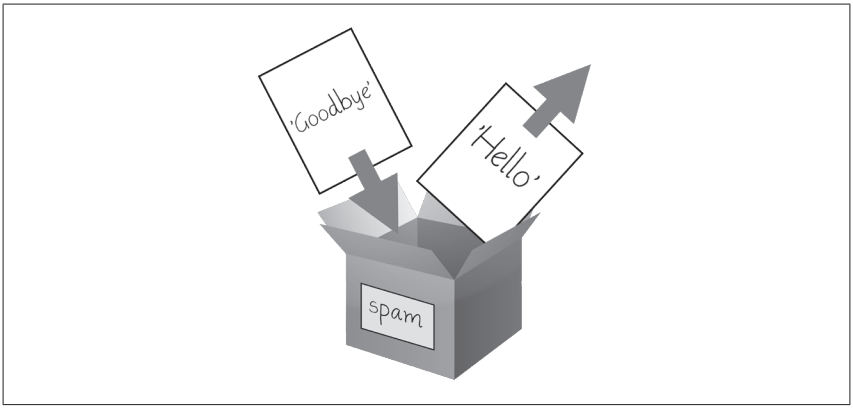

図1-3 新しい値が代入されると古い値は忘れ去られる

1.4.2 変数名

表1-3に正しい変数名の例を示します。以下のルールに従うかぎり、変数名は自由に付けられます。

1. ひとつながりの語であること。
2. 文字と数字と下線記号_のみを用いていること。
3. 数字から始まらないこと。

表1-3 正しい変数名と正しくない変数名

正しい変数名	正しくない変数名
balance	current-balance（ハイフン記号は使えない）
currentBalance	current balance（スペースは使えない）
current_balance	4account（数字から始めてはいけない）
_spam	42（数字から始めてはいけない）
SPAM	total_$um（$のような特殊文字は使えない）
account4	'hello'（'のような特殊文字は使えない）

変数名は大文字と小文字を区別するので、spam、SPAM、Spam、sPaMの4つは異なる変数として扱われます。Pythonの慣習では、変数名は小文字で始まります。

変数名の書き方には流儀があります。lookLikeThisというように、単語の先頭を大文字にしてつなげる書き方は、ラクダ（= Camel）のコブのように凸凹しているので、キャメルケースといいます。一方、look_like_thisのように単語を下線つなぎで書く流儀もあり、Pythonの公式コーディングスタイルを規定したPEP 8によれば、こちら

を推奨しています[*1]。

どちらを使うのかは好き嫌いの問題で、一貫していればいいのですが、PEP 8には次のようにも書かれています。「愚かな一貫性は心の狭い小鬼である」[*2]

> 「スタイルガイドに従うことは重要です。しかし、最も重要なことは、どんな場合に従わなくてもよいかを知っていることです。スタイルガイドを適用しない場合もあります。疑問に思ったら、自分で最良の判断をしてください。」

変数には、どんなデータを扱っているのかがわかるように名前を付けましょう。引っ越しのときに、箱のすべてに「がらくた」というラベルが貼ってあったらどうなるか想像してみてください。必要なものが見つからなくなってしまいます。変数名も中身がわかるようにすべきなのです。本書やPythonのドキュメントの多くで、`spam`、`eggs`、`bacon`いう変数名が汎用的な名前として使われています（モンティ・パイソンの「スパム」というスケッチ（コント）に由来します）。けれども、みなさんのプログラムでは、意味のある名前を付けてコードを読みやすくしましょう。

1.5 最初のプログラム

インタラクティブシェルはPythonの命令をひとつずつ実行するのに便利ですが、Pythonのプログラムを書くにはファイルエディタに命令を書きます。**ファイルエディタ**とは、ノートパッドなどのテキストエディタに似ていますが、ソースコードを入力するための特別な機能が備わっています。IDLEでファイルエディタを開くには、[File] → [New File] メニューを選択します。

ウィンドウが開き入力待ちのカーソルが表示されますが、インタラクティブシェルと違うのは、命令を入力しても実行されません。ファイルエディタを用いるときには、たくさんの命令を入力し、ファイルを保存し、それからプログラムを実行します。両者を区別するのは簡単です。

- インタラクティブシェルのウィンドウには`>>>`のプロンプトが常に出ています。
- ファイルエディタのウィンドウには`>>>`のプロンプトが出ません。

---

*1 訳注：原書のコードは、原著者の意向でキャメルケースで書かれていましたが、翻訳にあたって全面的に下線つなぎに書き換えています。

*2 訳注：ラルフ・ワルド・エマーソンの言葉。

それでは、最初のプログラムを書いてみましょう！ ファイルエディタのウィンドウが開いたら、次のコードを入力してください。

```
# このプログラムは挨拶を表示して名前と年齢を尋ねる ❶

print('Hello world!')  # ❷
print('What is your name?') # 名前を尋ねる
my_name = input()  # ❸
print('It is good to meet you, ' + my_name)  # ❹
print('The length of your name is:')  # 名前の長さを表示 ❺
print(len(my_name))
print('What is your age?') # 年齢を尋ねる ❻
my_age = input()
print('You will be ' + str(int(my_age) + 1) + ' in a year.') # 来年の年齢を表示
```

ソースコードを入力したら、ファイルに保存しておけば、次回IDLEを起動したときに再度入力する必要がなくなります。ファイルエディタのウィンドウのメニューから［File］→［Save As］を選択します。ファイル保存ダイアログが表示されたら、ファイル名として`hello.py`と入力して［Save］ボタンをクリックして保存します。

プログラムを変更したら、毎回保存するようにしましょう。コンピュータが異常終了したり、うっかりIDLEを終了させたりしても、コードを失わずに済みます。ショートカットキーとして、WindowsやLinuxではCtrl-S、Macでは⌘-Sを押せば、簡単にファイルを保存することができます。

プログラムをファイルに保存したら実行してみましょう。［Run］→［Run Module］メニューを選ぶか、F5キーを押してください。IDLEを起動したときに開いたインタラクティブシェルの中でプログラムが実行します。F5キーはインタラクティブシェルではなくファイルエディタのウィンドウで押すことに注意してください。

プログラムを実行すると、名前と年齢をきいてきます。インタラクティブシェルの出力は次のようになるでしょう（太字は入力する文字で、入力後にEnterを押します）。

```
Python 3.6.0 |Anaconda 4.3.0 (64-bit)| (default, Dec 23 2016, 11:57:41)
[MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello world!
What is your name?
Al
```

```
It is good to meet you, Al
The length of your name is:
2
What is your age?
4
You will be 5 in a year.
>>>
```

実行すべきコードがなくなるとPythonのプログラムは**終了**、つまり実行を停止します（Pythonのプログラムが**exitする**ともいいます）。

ファイルエディタはウィンドウ上部の［×］ボタンをクリックすると閉じることができます。保存したプログラムをもう一度読み込むには、［File］→［Open］メニューを選択します。ファイル選択ダイアログが開いたら、`hello.py`を選択して［Open］ボタンをクリックします。以前に保存した`hello.py`プログラムがファイルエディタのウィンドウに表示されます。

1.6 プログラムを分析する

ファイルエディタにプログラムを開いたら、使用されているPythonの命令を1行ずつ調べていくことにしましょう。

1.6.1 コメント

次の行は**コメント**といいます。

```
# このプログラムは挨拶を表示して名前と年齢を尋ねる ❶
```

#記号から行末までのテキストをコメントとして扱います。Pythonはコメントを無視するので、コードが何をしようとするものなのかを忘れないように注釈を付けておくのに使います。

プログラマーがプログラムをテストするときに、行の先頭に#を付けることで一時的に無効化することがよくあります。これをコードを**コメントアウト**するといい、なぜプログラムがうまく動作しないのか考えるときに役立ちます。コードを再び有効にするには、#を削除するだけでよいのです。

コメントの次の空行は無視されます。プログラムには好きなだけ空行を入れることができます。本の段落と同じように、空行を入れて間隔を空けるとプログラムが読みやすくなります。

1.6.2 print()関数

print()関数は、カッコの中の文字列を画面に表示します。

```
print('Hello world!')  # ❷
print('What is your name?') # 名前を尋ねる
```

print('Hello world!')という行は、「'Hello world!'という文字列を表示しなさい」を意味します。Pythonがこの行を実行することを、Pythonが**print()関数を呼び出し**、文字列の値が関数に**渡される**といいます。関数に渡される値のことを**引数**といいます。シングルクォート記号は表示されないことに注意しましょう。これは文字列の開始と終了を表す記号なので、文字列の値に含まれないのです。

この関数は画面に空行を出すためにも使えます。カッコの中に何も入れずにprint()と呼び出すだけです。

名前の後の開きカッコと閉じカッコの存在が関数名であることの証です。本書では本文中に関数名を書くときにはprintと書かずにprint()と書くようにしています。関数については2章で詳しく説明します。

1.6.3 input()関数

input()関数は、ユーザーが入力キーボードから何かテキストを入力してEnterキーを押すのを待ちます。

```
my_name = input()  # ❸
```

この関数を呼び出すと、ユーザーが入力した文字列を返します。上記の例ではmy_nameという変数にその文字列が代入されます。

input()関数の呼び出しを、ユーザーが入力した文字列へと評価される式とみなすこともできます。ユーザーが「Al」と入力したら、式を評価した結果my_name = 'Al'となるわけです。

1.6.4 ユーザー名を表示する

次のprint()の呼び出しで、カッコの中に'It is good to meet you, ' + my_nameという式が含まれています。

```
print('It is good to meet you, ' + my_name)  # ❹
```

式は常にひとつの値に評価されるのでしたね。前の行でmy_nameに'Al'という値が保存されていれば、この式を評価すると'It is good to meet you, Al'となります。このひとつの文字列の値がprint()に渡されて画面に表示されるのです。

1.6.5 len()関数

len()という関数に文字列の値（もしくは文字列を格納した変数）を渡すと、文字列に含まれる文字数を整数値で返します。

```
print('The length of your name is:')  # 名前の長さを表示 ❺
print(len(my_name))
```

インタラクティブシェルに次のコードを入力してみてください。

```
>>> len('hello')
5
>>> len('My very energetic monster just scarfed nachos.')
46
>>> len('')
0
>>> len('こんにちは')
5
```

これらの例からわかるように、len(my_name)は整数を返します。これをprint()に渡して画面に表示します。print()は整数でも文字列でも渡せますが、注意が必要です。次のようにインタラクティブシェルに入力するとエラーになります。

```
>>> print('I am ' + 29 + ' years old.')
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print('I am ' + 29 + ' years old.')
TypeError: Can't convert 'int' object to str implicitly
型エラー:'int'型のオブジェクトをstrに暗黙的に変換できない
```

エラーが起こっているのはprint()関数ではなく、print()に渡そうとしている式

です。式の部分だけをインタラクティブシェルに入力しても同じエラーが起こります。

```
>>> 'I am ' + 29 + ' years old.'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    'I am ' + 29 + ' years old.'
TypeError: Can't convert 'int' object to str implicitly
```

+演算子は2つの数の足し算か2つの文字列の連結にしか使えないのに、整数と文字列の間に使ったためにエラーが起こりました。整数を文字列に足し合わせるのは、Pythonの文法に反します。次に説明するように、整数を文字列に変換することでこの問題を解決できます。

1.6.6 str()、int()、float()関数

29のような整数を文字列に連結してprint()に渡したいときには、29を文字列化した'29'という値が必要です。次に示すように、str()関数に整数値を渡すと、文字列化した値を返します。

```
>>> str(29)
'29'
>>> print('I am ' + str(29) + ' years old.')
I am 29 years old.
```

str(29)が'29'になるので、'I am ' + str(29) + ' years old.'は、'I am ' + '29' + ' years old.'となり、最終的に'I am 29 years old.'という値が得られます。これがprint()関数に渡される値です。

str()、int()、float()関数は、それぞれ、渡された値を文字列、整数、浮動小数点数に変換します。次のようにインタラクティブシェルに入力して、これらの関数を用いて値を変換してみましょう。

```
>>> str(0)
'0'
>>> str(-3.14)
'-3.14'
>>> int('42')
42
>>> int('-99')
-99
>>> int(1.25)
```

```
1
>>> int(1.99)
1
>>> float('3.14')
3.14
>>> float(10)
10.0
```

str()、int()、float()関数に、異なるデータ型の値を渡すと、それぞれ、文字列、整数、浮動小数点数に変換してくれます。

str()関数は、整数や浮動小数点数を文字列に連結したいときに便利です。int()関数は、数字を文字列として保持しているときに、それを計算に用いたいときに使います。例えば、input()関数は、ユーザーが数字を入力しても常に文字列を返します。インタラクティブシェルにspam = input()と入力し、入力待ちになったら101と入力してみましょう。

```
>>> spam = input()
101
>>> spam
'101'
```

変数spamに保存された値は整数の101ではなく文字列の'101'です。spamの値を使って計算したいときには、int()関数を呼び出して整数化し、あらためてspamの新しい値として格納します。

```
>>> spam = int(spam)
>>> spam
101
```

これで変数spamを文字列ではなく整数として扱うことができるようになりました。

```
>>> spam * 10 / 5
202.0
```

int()に整数として評価できない値を渡すと、Pythonはエラーを表示します。

```
>>> int('99.99')
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    int('99.99')
ValueError: invalid literal for int() with base 10: '99.99'
```

```
値エラー:10を基数とするint()に不正なリテラルが渡された
>>> int('twelve')
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    int('twelve')
ValueError: invalid literal for int() with base 10: 'twelve'
```

int()関数は、浮動小数点数の小数点以下を切り捨てるのにも使えます。

```
>>> int(7.7)
7
>>> int(7.7) + 1
8
```

例題プログラムのhello.pyでは、最後の3行で関数int()とstr()を使って、適切なデータ型に変換しています。

```
print('What is your age?') # 年齢を尋ねる ❻
my_age = input()
print('You will be ' + str(int(my_age) + 1) + ' in a year.') # 来年の年齢を表示
```

変数my_ageにはinput()が返した値が格納されています。input()関数は（たとえそれが数字だとしても）常に文字列を返すので、int(my_age)というコードを使ってmy_ageに格納されている文字列の整数としての値を得ます。そしてint(my_age) + 1という式でその値に1を足します。

この足し算の結果をstr()関数に渡します。以上をまとめてstr(int(my_age) + 1)と書きます。その結果得られた文字列の値を、'You will be 'と' in a year.'を連結して、長い文字列の値とします。最終的にこの長い文字列がprint()に渡されて画面に表示されます。

例えばユーザーがmy_ageに'4'と答えたとします。文字列'4'を整数に変換して1を足すと5になります。str()関数を用いて文字列に戻して、他の文字列と連結して最終的なメッセージとします。以上の評価手順を**図1-4**に示します。

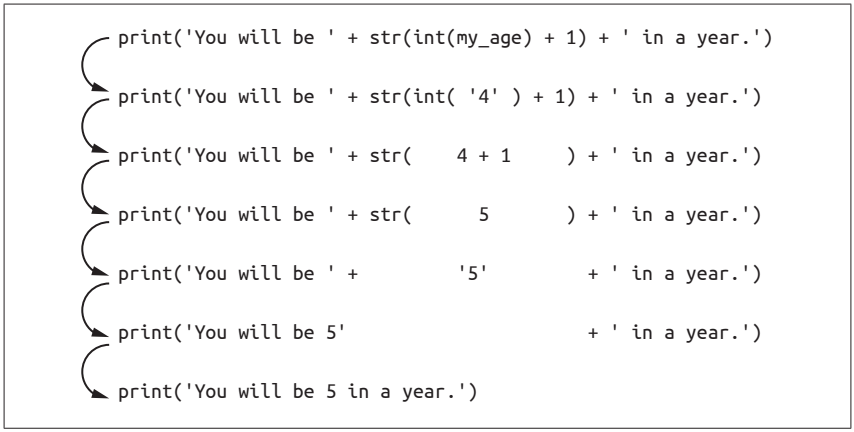

図1-4　my_ageに4が格納された場合の評価手順

テキストと数字の等価性

数字の文字列表現と整数や浮動小数点数とはまったく異なりますが、整数は浮動小数点数と等しくなることがあります。

```
>>> 42 == '42'
False
>>> 42 == 42.0
True
>>> 42.0 == 0042.000
True
```

Pythonでは、文字列はテキストであり、整数と浮動小数点数は数値である、というように両者を区別しています。

1.7　まとめ

計算機を用いれば式を計算できますし、ワープロを使って入力すれば文字列を連結することができます。文字列をコピー&ペーストすれば文字列の複製も簡単です。けれども、式とその構成要素、すなわち演算子や変数や関数呼び出しは、プログラムを作る基本的な構成要素です。これらの要素の扱い方を覚えれば、あなたに代わってPythonに大量のデータ処理をやらせることができます。

本章で紹介した、さまざまな種類の演算子（算術演算子+、-、*、/、//、%、**、文字列用の演算子 +、*）と、3つのデータ型（整数、浮動小数点数、文字列）を覚えておきましょう。

いくつかの関数も紹介しました。`print()`と`input()`は、簡単に画面に出力したり、キーボードから入力する関数です。`len()`は文字列を受け取り文字数を整数値で返す関数です。`str()`、`int()`、`float()`は、それぞれ、渡された値を、文字列、整数、浮動小数点数に変換する関数です。

次の章では、変数等の値に基づいて、どのコードを実行し、どのコードをスキップし、どのコードを繰り返すのか、Pythonに知的な判断な判断をさせる方法について学びます。これを**フロー制御**といい、プログラムを賢く条件分岐させるための手段となります。

1.8 演習問題

1-1 次のうち、どれが演算子で、どれが値ですか？

```
*
'hello'
-88.8
-
/
+
5
```

1-2 次のうち、どちらが変数で、どちらが文字列ですか？

```
spam
'spam'
```

1-3 データ型を3つ挙げなさい。

1-4 式は何から構成されていますか？ 式を評価するとどのようになりますか？

1-5 本章では、`spam = 10`のような代入文を紹介しましたが、式と文の違いは何ですか？

1-6 次のコードを実行すると、変数`bacon`には何が格納されますか？

```
bacon = 20
bacon + 1
```

1-7 次の2つの式を評価すると何になりますか？

```
'spam' + 'spamspam'
'spam' * 3
```

1-8 eggsは正しい変数名であり100は間違った変数名である理由は何ですか？

1-9 ある値を整数、浮動小数点数、文字列にそれぞれ変換するのに用いる関数は何ですか？

1-10 次の式がエラーになる理由は何ですか？ どうしたら修正できますか？

```
'I have eaten ' + 99 + ' burritos.'
```

発展問題 Pythonのオンラインドキュメントからlen()関数を探しなさい。「組み込み関数（Built-in Functions）」というタイトルのページが見つかるはずですが、そこから、他のPythonの組み込み関数を抽出し、round()関数の働きを調べ、インタラクティブシェルに入力して試しなさい。

17章
画像の操作

デジカメを持っている人や、スマホからFacebookに写真をアップロードする人は、デジタル画像ファイルと接する機会が頻繁にあると思います。Windowsのペイントのような基本的なグラフィックソフトや、Adobe Photoshopのような高度なアプリの使い方をご存じかもしれません。けれども、大量の画像を編集する必要があると、手作業で編集するのは時間がかかるうえ退屈です。

そこでPythonを使いましょう。Pillowはサードパーティーの画像処理モジュールで、切り抜きや、サイズ変更、画像内容の編集を簡単に行う関数を備えています。ペイントやPhotoshopのようなソフトと同様に画像処理できる能力を駆使して、何百枚、何千枚もの画像を簡単に自動編集できるのです。

17.1 コンピュータ画像の基礎

画像操作の説明をする前に、コンピュータがどのように画像の色や座標を扱っているのか、また、Pillowではどのように色と座標を扱うのか、基本を説明します。まず、`pillow`モジュールをインストールしましょう。サードパーティーのモジュールのインストール方法は、付録Aを参照してください。

17.1.1 色とRGBA値

コンピュータは画像の色を**RGBA値**で表現します。RGBA値とは、色の三原色（赤、緑、青）の度合いと、**アルファ値**（不透明度）をひとまとめにしたものです。それぞれの値は0から最大255までの値をとります。**ピクセル**（画素）のひとつずつにRGBA値が割り当てられます。ピクセルとは、コンピュータの画面で表示できる最小の単色の点のことであり、画面は何百万ものピクセルから構成されています。ピクセルのRGB値は、表示する色の濃淡値そのものを表しています。アルファ値は不透明度を表します。

背景画像やデスクトップの壁紙に重ねて画像を表示するとき、アルファ値はどれだけ背景が透き通らないかを表します。0なら透明で背景が完全に透き通り、255なら不透明で背景は見えません。

Pillowでは、RGBA値は4つの整数値のタプルで表されます。例えば、赤い色は`(255, 0, 0, 255)`で表されます。つまり、赤は最大値、緑と青の成分はなく、アルファ値が最大で不透明になります。同様に、緑は`(0, 255, 0, 255)`、青は`(0, 0, 255, 255)`となります。白は全成分が最大値の`(255, 255, 255, 255)`、黒は色成分のない`(0, 0, 0, 255)`となります。

アルファ値が0なら、RGBの成分に関係なく不可視になります。つまり、不可視の赤は不可視の黒と同じなのです。

PillowはHTMLが採用している標準色名を使います。**表17-1**に、標準色名とその値の抜粋を示します。

表17-1　標準色名とRGBA値

色名	RGBA値	色名	RGBA値
White	`(255, 255, 255, 255)`	Red	`(255, 0, 0, 255)`
Green	`(0, 128, 0, 255)`	Blue	`(0, 0, 255, 255)`
Gray	`(128, 128, 128, 255)`	Yellow	`(255, 255, 0, 255)`
Black	`(0, 0, 0, 255)`	Purple	`(128, 0, 128, 255)`

Pillowには`ImageColor.getcolor()`関数があり、色名に対応するRGBAを検索できるので、RGBA値を覚えておく必要はありません。この関数の第1引数に色名の文字列を、第2引数に`'RGBA'`という文字列を渡すと、RGBAのタプルを返します。

CMYKとRGB

小学校で、絵具を混ぜれば色を作れることを習ったと思います。例えば青と黄色を混ぜれば緑になります。これは減色混合といい、染料やインク、顔料に適用するものです。カラープリンタにCMYKインクを使う理由もこれです。C（シアン、水色）、M（マゼンタ、赤紫）、Y（イエロー、黄）、K（ブラック、黒）を組み合わせれば、任意の色を作れるのです。

一方、光の物理学では加色混合に従います。コンピュータの画面のように光を組み合わせる場合には、R（赤）、G（緑）、B（青）の光を組み合わせて任意の色を作ります。プログラム上で色を表すのにRGB値を使う理由がこれです。

この関数の働きを調べるために、次のようにインタラクティブシェルに入力してください。

```
>>> from PIL import ImageColor          # ❶
>>> ImageColor.getcolor('red', 'RGBA') # ❷
(255, 0, 0, 255)
>>> ImageColor.getcolor('RED', 'RGBA') # ❸
(255, 0, 0, 255)
>>> ImageColor.getcolor('Black', 'RGBA')
(0, 0, 0, 255)
>>> ImageColor.getcolor('chocolate', 'RGBA')
(210, 105, 30, 255)
>>> ImageColor.getcolor('CornflowerBlue', 'RGBA')
(100, 149, 237, 255)
```

まずPILからImageColorモジュールをインポートします（❶）。後で説明しますが、PillowでなくPILです。ImageColor.getcolor()に渡す色名は、大文字と小文字を区別しないので、'red'（❷）と'RED'（❸）は同じRGBAタプルを返します。'chocolate'や'CornflowerBlue'のようなあまり使われない色名を渡すこともできます。

Pillowは、'aliceblue'や'whitesmoke'といった大量の色名をサポートしています。Wikipedia（https://ja.wikipedia.org/wiki/ウェブカラー）に、100個以上の標準色名のリストが掲載されているので参考にしてください。

17.1.2 座標と矩形タプル

画像のピクセルの位置はx,y座標、すなわち画像上の水平位置と垂直位置で表されます。ピクセルの**原点**（origin）は画像の左上の点であり、(0, 0)で表されます。最初の0はx座標を表し、原点が0で右にいくほど増えていきます。2つ目の0はy座標を表し、原点が0で下にいくほど増えていきます。大事なことなので二度いいます。y座標は下にいくほど増えます。これは算数の授業で習ったy座標と逆です。**図17-1**に座標系を示します。

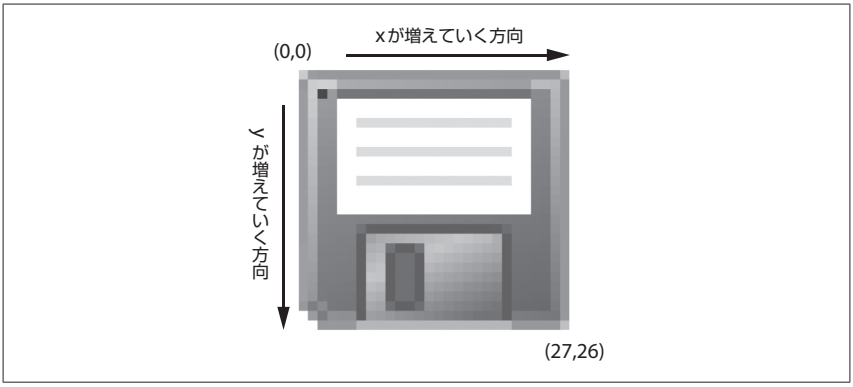

図17-1　太古のデータ記録装置を表す27×26の画像のx,y座標

多くのPillowの関数やメソッドは、**矩形タプル**の引数をとります。つまり、画像中の矩形領域を表す4つの整数値のタプルを要求します。4つの整数は順番に次のとおりです。

- **左** ── 矩形の左上点のx座標。
- **上** ── 矩形の左上点のy座標。
- **右** ── 矩形の右下点のx座標。左の値より大きくなければならない。
- **下** ── 矩形の右下点のy座標。上の値より大きくなければならない。

矩形には、左上点の座標が含まれますが、右下点の座標は含まれないことに注意してください。例えば、`(3, 1, 9, 6)`というタプルは、**図17-2**の黒いピクセルの矩形領域を表します。

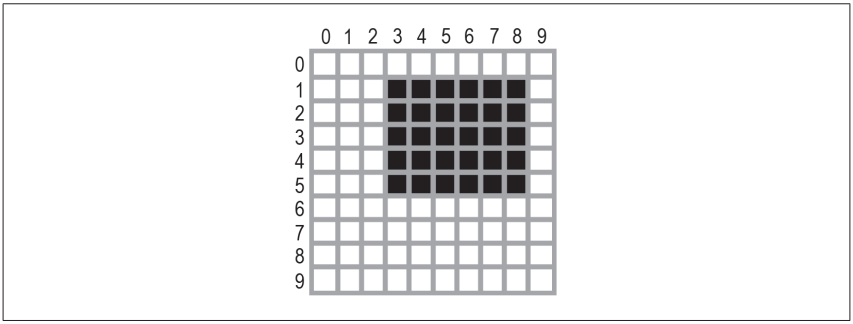

図17-2　タプル(3, 1, 9, 6)で表される領域

17.2 Pillowで画像を操作する

色と座標の扱い方を学んだので、さっそくPillowを使って画像を操作してみましょう。**図17-3**に、以後のインタラクティブシェルの例で用いる画像を示します。この画像はhttps://github.com/oreilly-japan/automatestuff-jaからダウンロード可能です。

図17-3 筆者の飼い猫Zophie。画像では太って見える

画像ファイルzophie.pngをカレントディレクトリに置いたら、次のようにPythonに読み込みます。

```
>>> from PIL import Image
>>> cat_im = Image.open('zophie.png')
```

画像を読み込むには、PillowからImageモジュールをインポートし、Image.open()にファイル名を渡して呼び出します。そして、戻り値をcat_imのような変数に格納しておきます。Pillowのモジュール名はPILです。これはPython Image Libraryという古いモジュールとの後方互換性のためです。したがって、from Pillow import Imageでなく、from PIL import Imageと書きます。Pillowの設計方針に従い、import PILではなく、form PIL import Imageの形式でimport文を書きます。

画像ファイルがカレントディレクトリにない場合には、os.chdir()関数を呼び出して、画像のあるフォルダに移動します。

```
>>> import os
>>> os.chdir('C:\\画像ファイルのあるフォルダ')
```

あるいは、Image.open()に画像ファイルのフルパスを渡します。

```
cat_im = Image.open('C:\\画像ファイルのあるフォルダ\\zophie.png')
```

Image.open()関数は、Imageオブジェクトを返します。さまざまなフォーマットの画像ファイルを、Image.open()関数に渡すだけでImageオブジェクトとして読み込むことができます。Imageオブジェクトに変更を加えたら、save()メソッドを使ってさまざまなフォーマットの画像ファイルに保存できます。回転やサイズ変更、切り取り、描画などの画像操作は、Imageオブジェクトのメソッド呼び出しを通じて行います。

以下の例を簡単にするために、zophie.pngを読み込んだImageオブジェクトが、変数cat_imに格納されているとします。

17.2.1 Imageオブジェクトを操作する

Imageオブジェクトには便利な属性があり、画像の幅や高さ、ファイル名、画像フォーマット（JPEG、GIF、PNGなど）といった、読み込んだ画像ファイルに関する基本情報を調べることができます。

例として、次のようにインタラクティブシェルに入力してください。

```
>>> # cat_im はzophie.pngを読み込んだImageオブジェクト
>>> cat_im.size
(816, 1088) # ❶
>>> width, height = cat_im.size  # ❷
>>> width   # ❸
816
>>> height  # ❹
1088
>>> cat_im.filename
'zophie.png'
>>> cat_im.format
'PNG'
>>> cat_im.format_description
'Portable network graphics'
>>> cat_im.save('zophie.jpg')  # ❺
```

Imageオブジェクトのsize属性には、画像の幅と高さをピクセル単位で表したタプルが格納されています（❶）。❷のように、変数widthとheightに代入すれば、width

(❸) やheight (❹) を使ってアクセスすることができます。filename属性は、ファイル名を表します。formatとformat_description属性は、画像フォーマットを表す文字列です。format_descriptionのほうが詳細な説明になっています。

最後に、save()メソッドに'zophie.jpg'を渡して呼び出すと、zophie.jpgというファイル名で新たに画像をハードドライブに保存します (❺)。Pillowはファイル拡張子が.jpgであるのを見て、自動的にJPEG形式で画像を保存します。これで、ハードドライブには、zophie.pngとzophie.jpgの2つの画像が存在することになります。どちらも同じ画像に基づくものですが、画像形式が異なるため、完全には一致しません (PNGは画質は劣化しませんが、JPEGは劣化します)。

PillowにはImageオブジェクトを返すImage.new()関数もあります。Image.open()に似ていますが、Image.new()は空白の画像を返します。Image.new()の引数は次のとおりです。

- 文字列'RGBA'。カラーモードをRGBAにする (本書で扱わないカラーモードが他にあります)。
- サイズ。画像の幅と高さを表す2つの整数のタプル。
- 初期値として画像を塗りつぶす背景色。RGBA値を表す4つの整数のタプル。ImageColor.getcolor()関数の戻り値を使ってもよいし、色名の文字列を直接Image.new()に渡してもよい。

例として、次のようにインタラクティブシェルに入力してください。

```
>>> from PIL import Image
>>> im = Image.new('RGBA', (100, 200), 'purple')  # ❶
>>> im.save('purpleImage.png')
>>> im2 = Image.new('RGBA', (20, 20))  # ❷
>>> im2.save('transparentImage.png')
```

ここでは、幅100×高さ200で紫の背景色の画像を表すImageオブジェクトを生成しています (❶)。それから、画像をpurpleImage.pngという名前で保存します。次に、(20, 20)の大きさで背景色のないImageオブジェクトを生成しています (❷)。背景色を省略すると、不可視の黒(0, 0, 0, 0)がデフォルトの色として使われるので、im2は透明背景になります。この20×20の透明な正方形画像をtransparentImage.pngという名前で保存します。

17.2.2　画像を切り抜く

画像の**切り抜き**は、画像中の矩形領域を選択して、矩形外を削除する操作です。Imageオブジェクトのcrop()メソッドは、矩形タプルを受け取り、切り抜いた画像を表すImageオブジェクトを返します。切り抜き操作は「インプレース」ではありません。すなわち、元のImageオブジェクトは変更されず、crop()メソッドは新しいImageオブジェクトを返します。繰り返しになりますが、切り抜き領域を表す矩形タプルには、左上の点は含まれますが、右下の点は含まれません。

次のようにインタラクティブシェルに入力してください。

```
>>> cropped_im = cat_im.crop((335, 345, 565, 560))
>>> cropped_im.save('cropped.png')
```

ここでは、切り抜かれた画像のImageオブジェクトが新たに生成されるので、cropped_imに格納し、save()を呼び出してcropped.pngに保存しています。元のファイルから作成されたcropped.pngという新しい画像ファイルを、**図17-4**に示します。

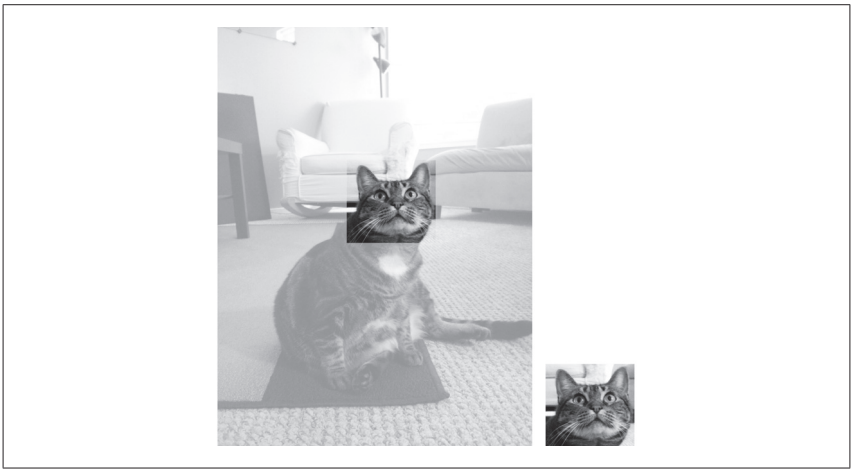

図17-4　元の画像から切り抜かれた新しい画像

17.2.3　画像のコピー&ペースト

copy()メソッドは、呼び出されたImageオブジェクトと同じ画像データを持つ新しいImageオブジェクトを生成して返します。画像に変更を加えたいが、元の変更しな

いバージョンも残しておきたい場合に便利です。例として、次のようにインタラクティブシェルに入力してください。

```
>>> cat_copy_im = cat_im.copy()
```

変数cat_imとcat_copy_imは、それぞれ別のImageオブジェクトですが、cat_copy_imにはcat_imの画像データの複製が格納されています。したがって、cat_copy_imを変更しても、cat_imには影響はありません。それでは、cat_copy_imをpaste()メソッドを使って変更してみましょう。

paste()メソッドを呼び出すと、他の画像を貼り付けることができます。インタラクティブシェルに次のように追加し、cat_copy_imに小さな画像を貼り付けます。

```
>>> face_im = cat_im.crop((335, 345, 565, 560))
>>> face_im.size
(230, 215)
>>> cat_copy_im.paste(face_im, (0, 0))
>>> cat_copy_im.paste(face_im, (400, 500))
>>> cat_copy_im.save('pasted.png')
```

まず、crop()に矩形タプルを渡して、猫の顔の部分に相当するzophie.pngの矩形領域を切り抜きます。これにより230×215の切り抜いた領域を表すImageオブジェクトが生成されるので、face_imに格納しておきます。次に、face_imをcat_copy_imに貼り付けます。paste()メソッドは、貼り付けるImageオブジェクトと、それを貼り付ける左上点のx,y座標を表すタプルの、2つの引数をとります。ここでは、paste()を2回呼び出して、face_imをcat_copy_imの(0, 0)と(400, 500)に貼り付けています。最後に、変更したcat_copy_imをpasted.pngに保存します。pasted.pngの様子を**図17-5**に示します。

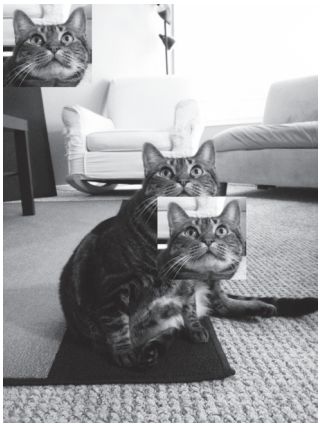

図17-5 猫の顔を2回貼り付けた様子

Pillowのcopy()とpaste()は、コピー&ペーストという名前ですが、コンピュータのクリップボードを使うものではありません。

paste()メソッドはImageオブジェクトを「インプレース」で変更する点に注意してください。つまり、貼り付けた画像を表す新しいImageオブジェクトを返すわけではありません。paste()を呼び出しつつ、元の画像を変更したくなければ、copy()で画像の複製を作り、複製に対してpaste()を呼び出します。

図17-6のように猫の顔で画像全体を敷き詰めたいなら、二重のforループで実現することができます。次のようにインタラクティブシェルに入力してください。

```
>>> cat_im_width, cat_im_height = cat_im.size
>>> face_im_width, face_im_height = face_im.size
>>> cat_copy_two = cat_im.copy()  # ❶
>>> for left in range(0, cat_im_width, face_im_width):  # ❷
        for top in range(0, cat_im_height, face_im_height):  # ❸
            print(left, top)
            cat_copy_two.paste(face_im, (left, top))

0 0
0 215
0 430
```

```
0 645
0 860
0 1075
230 0
230 215
……中略……
690 860
690 1075
>>> cat_copy_two.save('tiled.png')
```

ここではまずcat_imの幅と高さを、cat_im_widthとcat_im_heightに格納します。❶でcat_imの複製を作ってcat_copy_toに格納しています。貼り付ける先の画像の複製ができたので、face_imをcat_copy_twoに貼り付けるためにループを回します。外側のforループでは、変数leftを0からface_im_width(=230)ずつ増やします（❷）。内側のforループでは、変数topを0からface_im_height(=215)ずつ増やします（❸）。これらの入れ子になったforループによってleftとtopの値を順番に変えながら、**図17-6**に示すように、cat_copy_twoの上にface_imを敷き詰めるように貼り付けていきます。入れ子のループの様子を調べるために、leftとtopの値を表示しています。貼り付けが完了したら、変更後のcat_copy_twoをtiled.pngに保存します。

図17-6　二重のforループでpaste()を呼び出して猫の顔を複製する

透明ピクセルの貼り付け

通常、透明なピクセルは白のピクセルとして貼り付けられます。貼り付ける画像に透明ピクセルが含まれていて、白ではなく透明なピクセルとして貼り付けたいときには、`paste()`の第3引数としてその画像を渡します。第3引数はマスクImageオブジェクトです。マスクとは、アルファ値が意味を持ち、赤緑青の値が無視されるImageオブジェクトです。マスクは、`paste()`関数に、どのピクセルをコピーし、どのピクセルを透明として無視するのかを伝えます。マスクの高度な使い方は本書の範囲を超えますが、透明ピクセルを持つ画像を貼り付けたいなら、Imageオブジェクトを第1引数と第3引数の両方に渡せばいいのです。

17.2.4 画像をサイズ変更する

`resize()`メソッドを呼び出すと、指定した幅と高さにサイズを変更した新しいImageオブジェクトを返します。引数は新しい幅と高さを表す2つの整数のタプルです。次のようにインタラクティブシェルに入力してください。

```
>>> width, height = cat_im.size  # ❶
>>> quartersized_im = cat_im.resize((int(width / 2), int(height / 2)))  # ❷
>>> quartersized_im.save('quartersized.png')
>>> svelte_im = cat_im.resize((width, height + 300))  # ❸
>>> svelte_im.save('svelte.png')
```

ここではまず`cat_im.size`のタプルを変数`width`と`height`に代入します（❶）。`cat_im.size[0]`や`cat_im.size[1]`と書くより、`width`と`height`を使うほうがコードが読みやすくなります。

1つ目の`resize()`の呼び出し（❷）では、新たな幅に`int(width / 2)`、新たな高さに`int(height / 2)`を渡しているので、`resize()`の返すImageオブジェクトは、元の画像の半分の幅と高さ、つまり1/4のサイズになります。`resize()`の引数は整数のタプルしか受け付けないので、2で割ったら`int()`によって整数に変換する必要があります。

このサイズ変更では、幅と高さが同じ比率を保っていますが、元の画像の縦横比と異なる幅と高さを指定してもかまいません。変数`svelte_im`に格納される画像は、幅は元の画像と同じですが、高さは300ピクセル高いので（❸）、被写体の猫が細くなり

ます。

なお、resize()メソッドは元のImageオブジェクトを変更せず、新しいImageオブジェクトを返します。つまり、インプレースでないメソッドです。

画像の縦横比を維持したまま、あるサイズに収まるように縮小したい場合には、thumbnail()メソッドを使うのが便利です。引数は最大の幅と高さを表す2つの整数のタプルです。画像が指定サイズを上回る場合には、ちょうど収まるように縮小してくれます。

```
>>> thumb_im = cat_im.copy()
>>> thumb_im.size
(816, 1088)
>>> thumb_im.thumbnail((100, 100))
>>> thumb_im.size
(75, 100)
>>> thumb_im.save('thumbnail.jpg')
```

注意点としては、thumbnail()はインプレースなメソッドであり、元画像を変更します。元画像を保存しておきたいなら、上の例のようにcopy()で複製します。

thumbnail()の戻り値はNoneなので、間違えて次のように書くと画像を失ってしまいます。

```
>>> # 次の例は間違い
>>> thumb_im = thumb_im.thumbnail((100, 100))
>>> thumb_im == None
True
```

17.2.5 画像を回転・反転する

rotate()メソッドを用いると、画像を回転することができ、元の画像はそのままで、回転した画像をImageとして返します。rotate()の引数は、反時計回りの回転角度を表す整数か浮動小数点数です。次のようにインタラクティブシェルに入力してください。

```
>>> cat_im.rotate(90).save('rotated90.png')
>>> cat_im.rotate(180).save('rotated180.png')
>>> cat_im.rotate(270).save('rotated270.png')
```

ここでは、rotate()が返すImageオブジェクトに対して直接save()を呼び出すという、メソッドの**連鎖**呼び出しをしています。1行目の呼び出しは、反時計回りに90度回転した画像をrotate90.pngという名前で保存します。同様に、2行目は180度、3

行目は270度回転しています。結果を**図17-7**に示します。

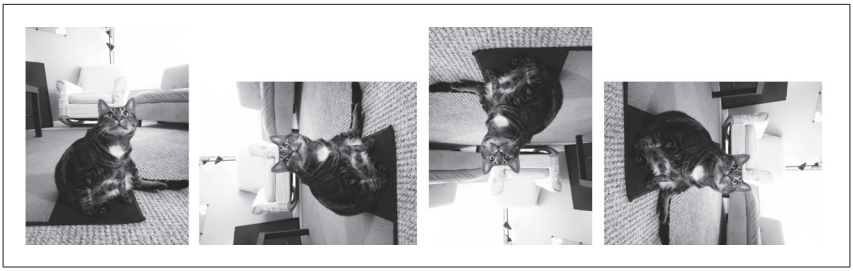

図17-7　元画像（左）と、反時計回りに90度、180度、270度回転した画像

90度と270度の回転では、画像の幅と高さが変わる点に注意してください。それ以外の角度では、元画像の幅と高さのままになります。**図17-8**に示すように、Windowsでは回転によって生じた隙間に黒いピクセルが埋められます。Macでは透明ピクセルにより埋められます。

`rotate()`メソッドにキーワード引数`expand=True`を与えると、回転した画像が収まるように、新しい画像の幅や高さは大きくなります。例として、次のようにインタラクティブシェルに入力してください。

```
>>> cat_im.rotate(6).save('rotated6.png')
>>> cat_im.rotate(6, expand=True).save('rotated6_expanded.png')
```

1行目の呼び出しでは、画像を6度回転させて`rotate6.png`に保存しています。**図17-8**の左に示すように、元のサイズで画像の一部が切り取られます。2行目の呼び出しでは、`expand=True`を追加して、`rotate6_expanded.png`に保存しています。**図17-8**の右に示すように、回転した画像が収まるようにサイズが調整されています。

`transpose()`メソッドを用いると、画像を鏡像反転することができます。引数には、`Image.FLIP_LEFT_RIGHT`か`Image.FLIP_TOP_BOTTOM`のどちらかを渡します。次のようにインタラクティブシェルに入力してください。

```
>>> cat_im.transpose(Image.FLIP_LEFT_RIGHT).save('horizontal_flip.png')
>>> cat_im.transpose(Image.FLIP_TOP_BOTTOM).save('vertical_flip.png')
```

`rotate()`と同様に、`transpose()`も新たに`Image`オブジェクトを生成し、元の画像は変更されません。1行目では、`Image.FLIP_LEFT_RIGHT`を渡して水平方向に反転した画像を`horizontal_flip.png`に保存しています。2行目では、`Image.FLIP_`

TOP_BOTTOMを渡して垂直方向に反転しvertical_flip.pngに保存しています。結果を**図17-9**に示します。

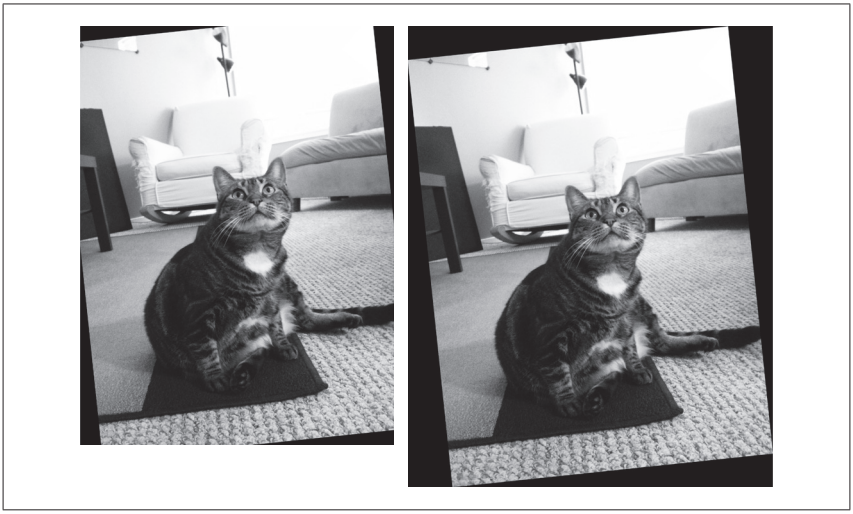

図17-8　単純に6度回転させた画像（左）とexpand=Trueを指定して回転した画像（右）

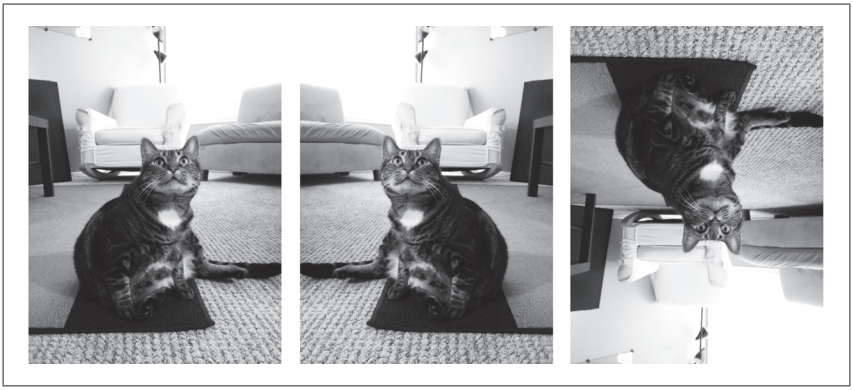

図17-9　元画像（左）、水平反転（中）、垂直反転（右）

17.2.6　ピクセルを変更する

ひとつのピクセルは、getpixel()メソッドで取得し、putpixel()メソッドで設定することができます。どちらのメソッドも、ピクセルのx,y座標を表すタプルを受け取ります。putpixel()メソッドには、さらにピクセルの色を表すタプルも渡します。色

の引数は、RGBAの4つの整数のタプルか、RGBの3つの整数のタプルです。次のようにインタラクティブシェルに入力してください。

```
>>> im = Image.new('RGBA', (100, 100))  # ❶
>>> im.getpixel((0, 0))  # ❷
(0, 0, 0, 0)
>>> for y in range(50):  # ❸
        for x in range(100):
            im.putpixel((x, y), (210, 210, 210))  # ❹

>>> from PIL import ImageColor
>>> darkgray = ImageColor.getcolor('darkgray', 'RGBA') # ❺
>>> for y in range(50, 100):  # ❻
        for x in range(100):
            im.putpixel((x, y), darkgray)

>>> im.getpixel((0, 0))
(210, 210, 210, 255)
>>> im.getpixel((0, 50))
(169, 169, 169, 255)
>>> im.save('putPixel.png')
```

❶で100×100の透明な正方形画像を生成します。この画像は透明なので、適当な座標で`getpixel()`を呼び出すと`(0, 0, 0, 0)`を返します（❷）。この画像のピクセルに色を付けるために、二重の`for`ループを使って（❸）、画像の上半分のすべてのピクセルに`putpixel()`を順番に呼び出します（❸）。`putpixel()`には、明るい灰色の`(210, 210, 210)`というRGBタプルを渡します（❹）。

画像の下半分を暗い灰色にしたいのですが、そのRGBタプルを知らないとします。`putpixel()`は`darkgray`といった標準色名を受け付けないので、`ImageColor.getcolor()`を使って`darkgray`に相当するタプルを取得します（❺）。画像の下半分をループして、`putpixel()`にそのタプルを渡します（❻）。これで**図17-10**に示すような、上半分が明るい灰色で下半分が暗い灰色の画像を作成できました。適当な座標で`getpixel()`を呼び出せば、期待どおりの色になっていることを確認できます。最後に画像を`putPixel.png`に保存します。

もちろん、画像を1ピクセルずつ描くのは不便です。後述するように、`ImageDraw`モジュールの関数群を用いれば、図形を描くことができます。

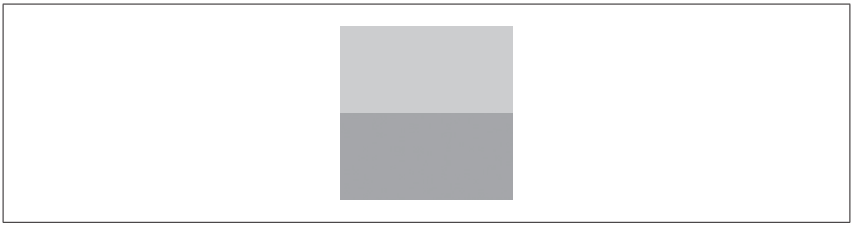

図17-10　putPixel.pngの画像

17.3　プロジェクト：ロゴを追加する

何千枚もの画像のサイズを変更して、画像の隅に小さなロゴの透かしを追加するという退屈な作業をしているとします。ペイントブラシのような単純なアプリでこの作業をすると、いつまでたっても終わらない気がします。Photoshopのような高度なアプリならバッチ処理ができますが、Photoshopは何万円もします。そこで、この作業をやってくれるスクリプトを書くことにしましょう。

画像の右下に追加するロゴは、**図17-11**に示すような、黒猫のアイコンで、図では見えませんが白い輪郭があり、残りの部分は透明だとします。

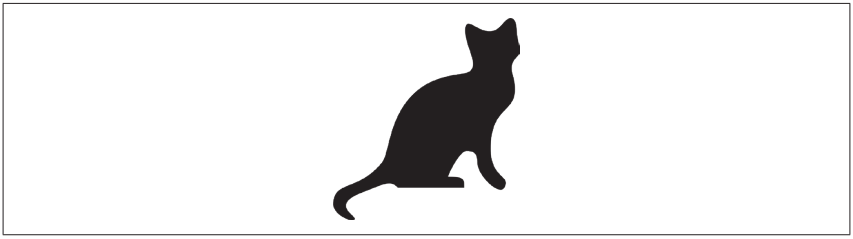

図17-11　画像に追加するロゴ

概要としては、プログラムは次のように動作します。

- ロゴ画像を読み込む。
- カレントディレクトリのすべての`.png`と`.jpg`ファイルをループする。
- 幅と高さが300ピクセル以内のサムネイル画像を作成する。
- ロゴ画像を右下に貼り付ける。
- 変更した画像を別のフォルダに保存する。

コードとしては次のように書きます。

- catlogo.pngファイルを開きImageオブジェクトを取得する。
- os.listdir('.')の戻り値の文字列をループする。
- thumbnail()メソッドを使って縮小する。
- paste()メソッドを使ってロゴを貼り付ける。
- save()メソッドを使って、元と同じファイル名で別のフォルダに保存する。

17.3.1　ステップ1：ロゴ画像を開く

新しくファイルエディタウィンドウを開いて、次のコードを入力し、resizeAndAddLogo.pyという名前で保存してください。

```
#! python3
# resizeAndAddLogo.py - カレントディレクトリのすべての画像を300x300に収まる
#                       ようにサイズ変更し、catlogo.pngを右下に追加する。

import os
from PIL import Image

SQUARE_FIT_SIZE = 300               # ❶
LOGO_FILENAME = 'catlogo.png'       # ❷

logo_im = Image.open(LOGO_FILENAME)  # ❸
logo_width, logo_height = logo_im.size # ❹

# TODO: カレントディレクトリの全画像をループする

# TODO: 画像をサイズ変更する

# TODO: ロゴを追加する

# TODO: 変更を保存する
```

プログラムの冒頭で、SQUARE_FIT_SIZE（❶）とLOGO_FILENAME（❷）という定数を設定しておけば、後で変更するのが容易になります。ここでは、画像の最大の幅と高さは300ピクセル、ロゴ画像は'catlogo.png'としていますが、これを変更したい場合には、コードを編集して、冒頭に定義されているこれらの定数の値を変えるだけで済みます。あるいは、これらの値を、コマンドライン引数から指定するようにしても

よいです。このように定数を定義しないでいると、変更したいときに、プログラム全体から300や'catlogo.png'を探し出して書き換えなければなりません。定数を使えば、プログラムの汎用性が高まるのです。

Image.open()を呼び出してImageオブジェクトをlogo_imに格納します（❸）。読みやすくするために、logo_im.sizeの値を、logo_widthとlogo_heightに代入しておきます。

残りはTODOコメントによる概要です。

17.3.2 ステップ2：全ファイルをループして画像を開く

次に、カレントディレクトリにあるすべての.pngと.jpgファイルを探します。ロゴ画像自体は加工したくないので、LOGO_FILENAMEと一致するファイル名はスキップします。次のようにコードを追加してください。

```
#! python3
# resizeAndAddLogo.py - カレントディレクトリのすべての画像を300x300に収まる
#                       ようにサイズ変更し、catlogo.pngを右下に追加する。

import os
from PIL import Image
……中略……
os.makedirs('withLogo', exist_ok=True)

# カレントディレクトリの全画像をループする
for filename in os.listdir('.'):  # ❶
    if not (filename.endswith('.png') or filename.endswith('.jpg')) \
       or filename == LOGO_FILENAME:# ❷
        continue # 画像以外とロゴ画像はスキップする  # ❸

    im = Image.open(filename)  # ❹
……後略…
```

元画像を保持したままロゴを追加した画像を保存するために、まずos.makedirs()を呼び出し、保存先のwithLogoフォルダを作成します。キーワード引数のexist_ok=Trueを指定すれば、withLogoフォルダがすでに存在するときでもos.makedirs()が例外を起こしません。次に、os.listdir('.')を使って、カレントディレクトリにあるすべてのファイルをループします（❶）。❷の長いif文により、ファイル名が.pngか.jpgで終わるかどうか、さらにロゴ画像と一致しないかどうかを

調べ、該当しないならcontinue文でスキップして次のファイルに移ります（❸）。最後に、画像ファイルを開いてimに代入します（❹）。

17.3.3　ステップ3：画像をサイズ変更する

画像の幅か高さがSQUARE_FIT_SIZE（ここでは300ピクセル）を超える場合には、それに収まるように縮小します。それにはthumbnail()を使うだけで簡単に処理することができます。次のようにコードを追加してください。

```
#! python3
# resizeAndAddLogo.py - カレントディレクトリのすべての画像を300x300に収まる
#                       ようにサイズ変更し、catlogo.pngを右下に追加する。

import os
from PIL import Image
……中略……
    # 画像をサイズ変更する
    im.thumbnail((SQUARE_FIT_SIZE, SQUARE_FIT_SIZE))
    width, height = im.size
……後略…
```

17.3.4　ステップ4：ロゴを追加して変更を保存する

画像の右下にロゴを追加します。右下ぴったりにロゴを貼り付けるために、画像とロゴのサイズを使って計算します。貼り付ける位置の計算方法を**図17-12**に示します。ロゴを貼り付ける左端のx座標は、画像の幅からロゴの幅を引いた値です。同様に、上端のy座標は、画像の高さからロゴの高さを引いた値になります。

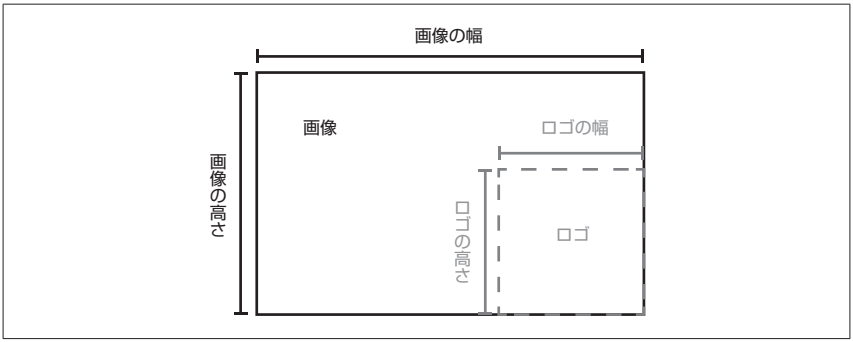

図17-12　画像の右下にロゴを配置するためのx,y座標は、画像の「幅,高さ」からロゴの「幅,高さ」を引いたもの

ロゴを画像に貼り付けたら、変更したImageオブジェクトを保存します。次のコードを追加してください。

```
#! python3
# resizeAndAddLogo.py - カレントディレクトリのすべての画像を300x300に収まる
#                       ようにサイズ変更し、catlogo.pngを右下に追加する。

import os
from PIL import Image
……中略……
    # ロゴを追加する
    print('ロゴを追加中 {}...'.format(filename))   # ❶
    im.paste(logo_im, (width-logo_width, height-logo_height), logo_im)  # ❷

    # 変更を保存する
    im.save(os.path.join('withLogo', filename))  # ❸
```

ロゴを追加中であることを示すメッセージを表示し（❶）、計算した座標にlogo_imを貼り付け（❷）、そして、withLogoフォルダの中に元と同じファイル名で保存します（❸）。このプログラムを実行したときにカレントディレクトリにzophie.pngしかなければ、出力は次のようになるでしょう。

```
ロゴを追加中 zophie.png...
```

zophie.pngは、**図17-13**のような225×300ピクセルの画像に縮小されます。paste()メソッドで透明ピクセルを反映するために、第3引数にもlogo_imを渡していることに注意してください。サイズを変更してロゴを追加する画像が何百枚もあったとしても、このプログラムを使えば数分で済むでしょう。

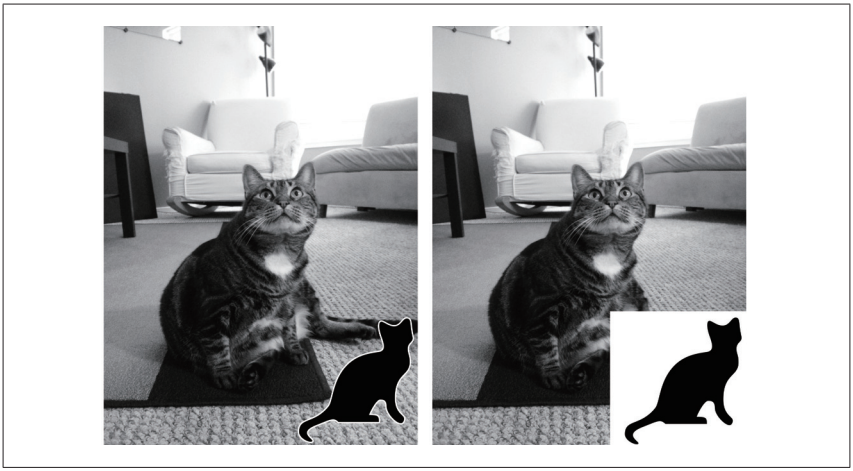

図17-13　サイズ変更したzophie.pngにロゴを追加したもの（左）。第3引数を忘れると透明ピクセルが白いピクセルになってしまう（右）

17.3.5　類似プログラムのアイデア

バッチ処理で画像を合成したりサイズを変更するのは、さまざまな応用が可能です。例えば、次のような類似のプログラムを書くことができるでしょう。

- 画像にテキストやWebサイトのURLを追加する。
- 画像にタイムスタンプを追加する。
- サイズに応じて異なるフォルダに画像をコピーしたり移動する。
- 無断複製を防ぐために、ほとんど透明の透かしを画像に追加する。

17.4　画像に描画する

画像に線や矩形や円などの簡単な図形を描きたいときには、PillowのImageDrawモジュールを使います。次のようにインタラクティブシェルに入力してください。

```
>>> from PIL import Image, ImageDraw
>>> im = Image.new('RGBA', (200, 200), 'white')
>>> draw = ImageDraw.Draw(im)
```

まずImageとImageDrawをインポートします。次に新しい画像（ここでは200×200の白地の画像）を作って、そのImageオブジェクトをimに格納します。これを

ImageDraw.Draw()関数に渡すと、ImageDrawオブジェクトを取得できます。このオブジェクトには、図形やテキストを画像に描画するためのメソッドが定義されています。後で使いやすくするために、ImageDrawオブジェクトを変数drawに格納しておきます。

17.4.1 図形を描画する

以下のImageDrawメソッドを使えば、さまざまな図形を描くことができます。これらのメソッドの塗りつぶし色fillと輪郭色outlineという引数はオプションであり、指定しなければデフォルトとして白が使われます。

17.4.1.1 点

point(xy, fill)メソッドは、1つ以上の点を描画します。引数xyは、描画したい点のx,y座標のリストで、[(x1, y1), (x2, y2), ...]のような座標値のタプルのリストか、[x1, y1, x2, y2, ...]のような座標値のリストを指定することができます。オプション引数fillには、描画する点の色として、RGBAのタプルか、'red'のような色名を表す文字列を指定します。

17.4.1.2 線分

line(xy, fill, width)メソッドは、1本以上の線分を描きます。引数xyは、[(x1, y1), (x2, y2), ...]のような座標値のタプルのリストか、[x1, y1, x2, y2, ...]のような座標値のリストです。各点は描画する線分の連結点になります。オプション引数fillには、線の色としてRGBAタプルか色名を指定します。オプション引数widthには、線幅を指定します。指定しなければデフォルトの1になります。

17.4.1.3 矩形

rectangle(xy, fill, outline)メソッドは矩形を描画します。引数xyは矩形タプル(left, top, right, bottom)です。leftとtopは矩形の左上点の座標、rightとbottomは右下点の座標を表します。オプション引数fillには、矩形の中を塗りつぶす色を指定します。オプション引数outlineには、矩形の境界線色を指定します。

17.4.1.4 楕円形

ellipse(xy, fill, outline)メソッドは楕円形を描画します。楕円の幅と高さが等しければ、円を描きます。引数xyは、楕円形に外接する長方形を表す矩形タプル

(left, top, right, bottom)です。オプション引数fillには、楕円形の中を塗りつぶす色を、オプション引数outlineには、楕円形の境界線色を指定します。

17.4.1.5　多角形

polygon(xy, fill, outline)メソッドは任意の多角形を描画します。引数xyは、[(x1, y1), (x2, y2), ...]のような座標値のタプルのリストか、[x1, y1, x2, y2, ...]のような座標値のリストであり、多角形の辺の頂点を表します。最後の頂点は先頭の頂点と自動的に連結されます。オプション引数fillには、多角形の中を塗りつぶす色、オプション引数outlineには、多角形の境界線色を指定します。

17.4.1.6　描画例

次のようにインタラクティブシェルに入力してください。

```
>>> from PIL import Image, ImageDraw
>>> im = Image.new('RGBA', (200, 200), 'white')
>>> draw = ImageDraw.Draw(im)
>>> draw.line([(0, 0), (199, 0), (199, 199), (0, 199), (0, 0)],
fill='black')  # ❶
>>> draw.rectangle((20, 30, 60, 60), fill='blue')  # ❷
>>> draw.ellipse((120, 30, 160, 60), fill='red')   # ❸
>>> draw.polygon(((57, 87), (79, 62), (94, 85), (120, 90), (103, 113)),
fill='brown')  # ❹
>>> for i in range(100, 200, 10):  # ❺
        draw.line([(i, 0), (200, i - 100)], fill='green')

>>> im.save('drawing.png')
```

200×200の白い画像のImageオブジェクトを生成して、ImageDraw.Draw()に渡し、ImageDrawオブジェクトを取得して、drawに保存して、メソッドを呼び出せるようにします。ここでは、画像の輪郭を黒い線で描き、左上点(20, 30)と右下点(60, 60)からなる青い長方形を描き、(120, 30)と(160, 60)からなる長方形に内接する赤い楕円形を描き、5点からなる茶色の多角形を描き、forループで緑の線のパターンを描きます。描画結果のdrawing.pngは**図17-14**のようになります。

ImageDrawオブジェクトには他にも描画メソッドがあります。完全なドキュメントは、http://pillow.readthedocs.org/en/latest/reference/ImageDraw.htmlから利用可能です。

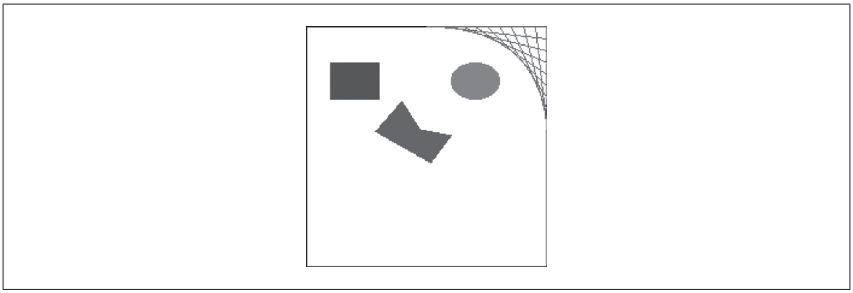

図17-14 描画結果の画像（drawing.png）

17.4.2 テキストを描画する

ImageDrawオブジェクトには画像にテキストを描画するtext()メソッドがあります。引数は、xy、text、fill、fontの4つです。

- 引数xyは、2つの整数からなるタプルで、テキストを表示する矩形の左上点の座標を表します。
- 引数textは描画するテキストの文字列です。
- オプション引数fillは、テキストの色です。
- オプション引数fontは、ImageFontオブジェクトであり、後述のようにフォントや文字の大きさを設定します。

指定したフォントでテキストが描かれる領域のサイズを事前に知るには、ImageDrawモジュールのtextsize()メソッドを用います。第1引数にはサイズを測りたいテキスト文字列を指定し、オプションの第2引数にはImageFontオブジェクトを指定します。textsize()メソッドはテキストの描画サイズの幅と高さを、2つの整数のタプルで返します。この幅と高さを用いれば、画像上にテキストを描画する位置を正確に計算することができます。

text()メソッドの先頭3つの引数は簡単です。text()でテキストを描画する前に、第4引数のImageFontオブジェクトについて説明します。

text()とtextsize()は、最後の引数としてImageFontオブジェクトをとることができます。利用するにはまず次のようにImageFontモジュールをインポートします。

```
>>> from PIL import ImageFont
```

インポートしたら、ImageFont.truetype()関数を呼び出すことができます。引数は2つとり、第1引数はフォントのTrueTypeファイルであり、ハードドライブ上の実際のフォントファイルを指定します。TrueTypeファイルの拡張子は.ttfであり、通常次のフォルダにあります。

Windows

```
C:\Windows\Fonts
```

Mac

```
/Library/Fontsや/System/Library/Fonts
```

Linux

```
/usr/share/fonts/truetype
```

Pythonはフォントのあるディレクトリを自動的に検索するので、truetype()にはTrueTypeファイルのフルパスを渡す必要はありません。指定したフォントファイルが見つからなければ、エラーを表示します。

truetype()関数の第2引数は、フォントサイズを**ポイント数**（もしくはピクセル数）で表す整数を渡します。Pillowはデフォルトで PNG 画像を 1 インチあたり 72 ピクセル（72dpi）として扱っています。そして、1ポイントは1/72インチです。

それでは次のようにインタラクティブシェルに入力してください。

```
>>> from PIL import Image, ImageDraw, ImageFont
>>> im = Image.new('RGBA', (200, 200), 'white')  # ❶
>>> draw = ImageDraw.Draw(im)  # ❷
>>> draw.text((20, 150), 'Hello', fill='purple') # ❸
>>> arial_font = ImageFont.truetype('arial.ttf', 32)  # ❹
>>> draw.text((100, 150), 'Howdy', fill='gray', font=arial_font)  # ❺
>>> im.save('text.png')
```

Image、ImageDraw、ImageFont、osモジュールをインポートしたら、200×200の白い画像のImageオブジェクト（❶）と、それに描画するImageDrawオブジェクトを生成します（❷）。❸でtext()を使って(20, 150)の位置に紫色で「Hello」と描画します。オプションの第4引数を渡さないので、この描画のフォントとサイズはデフォルトのものです。

フォントとサイズを指定するには、ImageFont.truetype()にTrueTypeフォントファイル名と、サイズを渡して呼び出します。戻り値のFontを変数arial_fontに格

納しておき（❹）、text()のキーワード引数fontに指定します（❺）。❺では、32ポイントのArialフォントを使い、(100, 150)の位置に灰色で「Howdy」と描画しています。

描画結果のtext.pngを**図17-15**に示します。

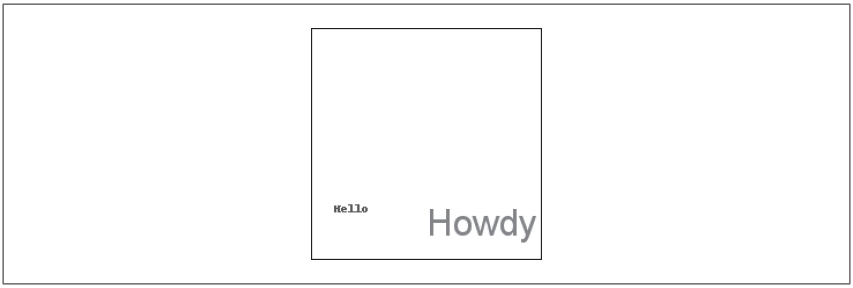

図17-15　描画結果画像（text.png）

日本語フォントを指定するには、ImageFont.truetype()に**表17-2**に示すフォントファイル名と、キーワード引数indexを渡します。

表17-2　日本語フォントとファイル名

フォント名	ファイル名	index
ＭＳ明朝	msmincho.ttc	0
ＭＳ Ｐ明朝	msmincho.ttc	1
ＭＳゴシック	msgothic.ttc	0
ＭＳ Ｐゴシック	msgothic.ttc	2
MS UI Gothic	msgothic.ttc	1
メイリオ	meiryo.ttc	0
Meiryo UI	meiryo.ttc	2

複数の書体が定義してあるフォントは、拡張子が.ttcになっていることに注意してください。コード例と実行結果を次に示します（**図17-16**）。

```
>>> from PIL import Image, ImageDraw, ImageFont
>>> im = Image.new('RGBA', (200, 200), 'white')
>>> draw = ImageDraw.Draw(im)
>>> jfont = ImageFont.truetype('msmincho.ttc', 24, index=1) # ＭＳ Ｐ明朝
>>> draw.text((20, 20), 'こんにちは', fill='black', font=jfont)
>>> jfont = ImageFont.truetype('msgothic.ttc', 24, index=2) # ＭＳ Ｐゴシック
>>> draw.text((20, 50), 'こんにちは', fill='black', font=jfont)
>>> jfont = ImageFont.truetype('meiryo.ttc', 24, index=0) # メイリオ
```

```
>>> draw.text((20, 80), 'こんにちは', fill='black', font=jfont)
>>> im.save('jtext.png')
```

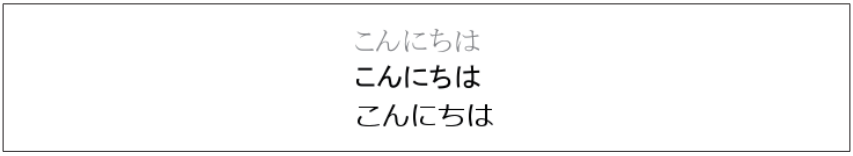

図17-16　描画結果画像（jtext.png）

17.5　まとめ

画像はピクセルの集合であり、ピクセルはRGBA値で色を、x,y座標で位置を操作します。JPEGとPNGはよく使われる画像フォーマットです。Pillowモジュールは、これら画像フォーマットを扱うことができます。

画像を`Image`オブジェクトにロードしたら、その幅と高さは、2つの整数からなるタプルとして`size`属性に格納されています。`Image`オブジェクトには、よく使われる画像処理として、`crop()`、`copy()`、`paste()`、`resize()`、`rotate()`、`transpose()`というメソッドがあります。`Image`オブジェクトを画像ファイルに保存するには、`save()`メソッドを呼び出します。

画像に図形を描きたいときには、`ImageDraw`オブジェクトを生成し、そのメソッドを呼び出して点や線、矩形、楕円形、多角形を描きます。フォントやサイズを指定してテキストを描画するメソッドもあります。

Photoshopのような高度（で高価）なアプリには、自動バッチ処理の機能がありますが、Pythonスクリプトを用いれば、無料でたくさんの画像に同じ処理をすることができます。以前の章では、プレーンテキストやスプレッドシート、PDFなどのファイル形式を扱ってきましたが、Pillowモジュールを用いれば、画像画像処理にもプログラミングの威力を展開することができるのです。

17.6　演習問題

17-1　RGBA値とは何ですか？

17-2　Pillowモジュールから、`CornflowerBlue`という色名のRGBA値を取得するには、どうすればよいですか？

17-3 矩形タプルとは何ですか？

17-4 zophie.pngという画像ファイルを読み込みImageオブジェクトを返す関数は何ですか？

17-5 Imageオブジェクトの画像の幅と高さを知るにはどうすればよいですか？

17-6 100×100の画像のImageオブジェクトから左下1/4を切り抜くには、どんなメソッドを呼び出しますか？

17-7 Imageオブジェクトを変更したら、どうしたら画像ファイルに保存できますか？

17-8 Pillowで図形を描画するモジュールは何ですか？

17-9 Imageオブジェクトには描画メソッドがありませんが、描画をするオブジェクトは何ですか？ どのようにそのオブジェクトを取得しますか？

17.7 演習プロジェクト

演習のために、次の動作をするプログラムを書きなさい。

17.7.1 章プロジェクトの改造と修正

本章で説明したresizeAndAddLogo.pyはPNGとJPEGファイルを処理しましたが、Pillowはこの2つ以外の画像フォーマットもサポートしています。そこで、resizeAndAddLogo.pyをGIFやBMP画像も処理できるように改造してください。

他にも画像のファイル拡張子は小文字のものしか扱えないという、小さな問題もあります。例えば、zophie.pngは扱えますが、zophie.PNGは扱えません。そこで、ファイル拡張子の大文字と小文字を区別しないように改造してください。

最後に、右下に追加するロゴは小さなマークであるべきですが、ロゴと同じサイズの画像に対しては、**図17-17**のようになってしまいます。そこで、画像がロゴの倍以上のサイズであることを確認し、そうでなければロゴの貼り付けをスキップするように、resizeAndAddLogo.pyを改造してください。

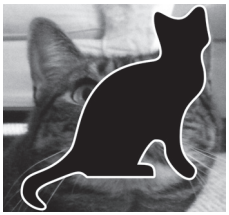

図17-17　画像がロゴより大きくなければ見た目のよい処理結果が得られない

17.7.2　ハードドライブの写真フォルダを探す

デジカメからパソコンに写真ファイルを転送するとき、適当な場所に一時的なフォルダに転送した後、忘れてしまうという悪い癖が筆者にはあります。そこで、ハードドライブ全体を調べて、そのような「忘れ去られた」写真フォルダを探し出すプログラムがあれば便利です。

ハードドライブのすべてのフォルダを調べて、埋もれた写真フォルダを探し出すプログラムを書いてください。もちろん、まず「写真フォルダ」とは何かを定義する必要があります。例えば、フォルダ内のファイルの半分以上が写真である、とします。次に、どのファイルが写真であるかを定義する必要があります。まず、写真ファイルは、.pngか.jpgの拡張子を持ちます。また、写真は大きいので、画像の幅と高さは500ピクセル以上とします。通常、デジカメ画像の幅と高さは数千ピクセルあるので、これでほとんどの写真を探せるはずです。

ヒントとして、プログラムの概観を示します。

```
#! python3
# 本プログラムのコメントを書き、モジュールをインポートする

for foldername, subfolders, filenames in os.walk('C:\\'):
    num_photo_files = 0
    num_non_photo_files = 0
    for filename in filenames:
        # ファイル拡張子が.pngでも.jpgでもなければ
        if TODO:
            num_non_photo_files += 1
            continue # 次のファイルにスキップする

        # Pillowを使って画像を開く
```

```
            # 幅と高さが500以上なら
            if TODO:
                # 画像が大きいので写真とみなす
                num_photo_files += 1
            else:
                # 画像は小さいので写真ではないとみなす
                num_non_photo_files += 1

    # 半分以上が写真なら、
    # フォルダの絶対パスを表示する
    if TODO:
        print(TODO)
```

プログラムを実行すると、写真フォルダの絶対パスを画面に表示するようにしてください。

17.7.3 カスタム座席カード

13章では演習プロジェクトとして、プレーンテキストに書かれた来場者のリストから、カスタム招待状を作るプログラムを書きました。追加プロジェクトとして、Pillowモジュールを用いてカスタム座席カードの画像を生成するプログラムを書きなさい。`guest.txt`ファイルに書かれた来場者ひとりずつにつき、来場者名と花の装飾を描画した画像を生成します。`guests.txt`とパブリックドメインの花画像は、https://github.com/oreilly-japan/automatestuff-jaから入手可能です。

座席カードは同じサイズであり、印刷後に切り抜くときのガイドとするために、黒い線でカードの輪郭を描画してください。Pillowが生成するPNGファイルは72dpiなので、4×5インチのカードは288×360ピクセルの画像になります。