巻頭言

何かをゼロから始める人にとって、入門書選びはとても難しい問題です。その分野のことを全く知らない状態のため、使われている言葉が1つ分からないというだけでつまずくことや、誤解して何時間もハマることがよくあります。だから「分かりやすく、丁寧に説明されている本で基礎から学びたい」と考えるかもしれません。

私も最近、電子工作の世界に足を踏み入れましたが、その分野の常識を知らずにハマって試行錯誤しています。それでも、基礎をしっかり身に付けてから作り始めようという気にはなりません。だって、何時間学んでも、そこから何が作れるようになるのか全然見えてこないんです。基礎を学ぶことはとても重要ですが、基礎を全部身に付けてから実践に進む方法では途中で疲れて挫折してしまいます。プログラミングでも電子工作でも、「簡単でも実践的なもの」を作って完成までの流れを把握してから、そこで必要になった基礎知識をその都度寄り道しながら学ぶほうが楽しめます。

本書『最短距離でゼロからしっかり学ぶ Python 入門』は、「必修編」で“実践に必要な基礎知識”がひと通り網羅されています。それはつまり、楽しいゲームを作るのも、仕事に役立つ道具を作るのも、必修編を修了すればあとはアイディア次第で始められるということです。そして「実践編」は、何かを作る全体像をなぞりながら必要な知識を学ぶのにピッタリです。原書名は『Python Crash Course』で、「Python短期集中講座」という意味合いです。クラッシュコースのハードな解釈には「水に投げ込んで泳げるか確認する（やってみてできないところを練習する）方式」というものもあるので、この解釈に沿って「実践編」から読みはじめるのもおすすめです。私のように実践から学びたい人は、「実践編」から始めて分からない部分にマーカーをひきつつ、「必修編」の当該箇所で基礎を押さえながら読み進める方式がおすすめです。日本語版で2冊に分かれたことで並行して読み進めやすくなったと言えるでしょう（笑）。翻訳した2人は日常的にPythonを書いている現役プログラマーとベテランWeb制作者というコンビです。異なる2つの視点で最新情報や日本特有の事情などが補完されていることにより、本書をより価値の高いものにしています。

本書の構成についてまとめましょう。**「必修編」**ではPythonの文法や基本的なデータ型が分かりやすく丁寧に説明されているため、基礎から足場をしっかり固めてから一歩ずつ学びたい人はこちらから読みはじめるとよいでしょう。読者自身で解決が難しい問題が起こりそうなところには、すぐに復帰できるように解決方法が配置してあります。これによって、「本のとおりにコードを書いたつもりなのにうまく動作しない」という書籍にありがちな問題にハマりづらい構成になっています。また、本書では例外とユニットテストについても触れていて、実践に必要な基礎知識がひと通り網羅されています。ユニットテストは作ったプログラムの入出力を明確にする設計の一部であり、その後の修正で意図しない変更（バグ）を起こさないためにも必要な、現在のプログラミングではほぼ必須となる大事な要素です。各所に配置された演習問題「やってみよう」は理解度を確認するのによいでしょう。

「実践編」では「エイリアン侵略ゲーム」の他、データ可視化ツールやWebアプリなどをプログラミングしていきます。プログラミングは、シンプルに書いて必要になったらより良い書き方に修正していく「リファクタリング」を行うことがよいとされていますが、実践編でもはじめから完璧なコードを目指すのではなく、愚直なコードをリファクタリングして整理されたコードを発掘していく流れを体験できる構成になっています。実践編を終えたら、いずれかのプロジェクトを足がかりに自分のアイディアを盛り込んでプログラムを進化させるのも良い腕試しになると思います。

2020年8月 清水川貴之

清水川貴之

株式会社ビープラウド所属。一般社団法人PyCon JP Association会計理事。2003年からPythonを使いはじめた。Python関連イベントを運営しつつ、カンファレンスや書籍、OSS開発を通じてPython技術情報を発信している。最近電子工作を始めました。

著書／訳書『自走プログラマー（2020年 技術評論社）』『Pythonプロフェッショナルプログラミング第3版（2018年 秀和システム）』『エキスパートPythonプログラミング 改訂2版（2018年 アスキードワンゴ）』『独学プログラマー（2018年 日経BP社）』『Sphinxをはじめよう第2版（2017年 オライリー・ジャパン）』

日本語版に寄せて

『Python Crash Course』（訳注：本書の原書）を執筆したことで得られている楽しみの1つは、この本が多くの異なる言語に翻訳され、プログラミングを学ぶ世界中の人々の目標達成を手助けできることです。ここのところ何年も多くの日本の読者からメールをいただいていたので、日本の読者がこの本を母国語で読めるようになることを知って、興奮を覚えています。

Pythonがはじめて世に出たとき、クリーンな言語であるとすぐに評判になりました。それは、他の言語と同等にパワフルでありながら扱いが簡単なことによります。そして、Pythonは長年にわたってシンプルさを維持しながら、着実に新機能を追加してきました。Pythonは開発されてから30年経ちましたが、完璧に現代的なプログラミング言語でもあります。Pythonは最初に学習するのに最適なプログラミング言語であり、多くのアプリケーションにおいてまさに最高のプログラミング言語です。多岐にわたる科学分野でも重用されており、データサイエンスのあらゆる局面で頼りになるプログラミング言語の1つでもあります。また、世界で最も人気のあるWebサイトのいくつかは、PythonベースのWebフレームワークであるDjangoで作られています。著名なテック系企業のほぼすべてが、何らかの形でPythonを使用しています。

Pythonは、他のプログラミング言語よりも一歩踏み込んだ形で人々に愛されています。これを物語る言葉があります。

I came for the language, and I stayed for the community.

——言語目当てでやって来て、コミュニティのおかげで居着くことになった。

Pythonコミュニティは、あらゆるバックグラウンドを持つ人々が世界中から参加することに対してとても積極的です。PyConイベントは50か国以上で開催されており、そのほとんどの国にPythonのユーザーグループがあります。オンラインとオフライン（対面）のPythonコミュニティには一貫したコード・オブ・コンダクト（行動規範）があり、職業や地位を問わず、すべての人々の学びと貢献が歓迎され、サポートされます。

クリーブランドで開催されたPyCon 2019で、翻訳者の1人である鈴木たかのりさんにお会いできたのは嬉しい出来事でした。この本の日本語版への翻訳にあたって、翻訳者のお二人とレビュアーの方々が注意深く丁寧な仕事をしてくださったことに深く感謝しています。鈴木さんとの再会を楽しみにしていますし、近いうちに家族で日本を訪れてみたいです。

皆さんのPythonの旅の成功を祈っています。もしこれがあなたにとってはじめてのプログラミング言語なら、あなたの前にはまったく新しい世界への扉が開かれようとしています。もしあなたが別のプログラミング言語の知識をお持ちなら、Pythonの持つシンプルさとパワーをお楽しみいただけるでしょう。

Happy coding!
Eric Matthes

はじめに

すべてのプログラマーには、最初にプログラムを書くことを学んだときの物語があります。私は子どもの頃にプログラミングを始めました。そのとき、父親はDECという近代コンピューティング時代の先駆的な企業で働いていました。私の最初のプログラムは、家の地下で父親が組み立てたコンピューターキット上で作成されました。そのコンピューターは、ケースがなくむき出しのマザーボードにキーボードを接続したもので、モニターはむき出しのブラウン管でした。私のはじめてのプログラムは単純な数字当てゲームで、次のようなものです。

```
数字を考えたよ！ぼくが考えた数字を当ててね: 25
小さすぎる！もう一度: 50
大きすぎる！もう一度: 42
正解！もう一度遊びますか？(yes/no) no
遊んでくれてありがとう！
```

私が作成したゲームが想定どおりに動き、それを家族が遊ぶところを見て、いつも満足していたことを覚えています。

この幼い頃の経験はいつまでも影響を及ぼしました。目的や課題を解決するために何かを作成することは満足感をもたらします。現在、私が作成しているソフトウェアは子どもの頃よりも重要なものですが、正しく動作するプログラムを作成することで得られる満足感は、同じくらい大きなままです。

この本の対象読者は?

本書の目的は、Pythonをできるだけ早く使いこなせるようになることです。そのために、「実践編」では動作するプログラム（ゲーム、データの可視化、Webアプリケーション）を構築しながら、今後の人生で役立つプログラミングの基礎を習得します。『最短距離でゼロからしっかり学ぶ Python 入門』は、Pythonのプログラムを書いたことがない、または全くプログラムを書いたことがないあらゆる世代の人に向けて書かれています。興味のあるプロジェクトに集中するためにプログラミングの基礎を学びたい人や、新しい概念を理解するために意味のある課題を解きたい人におすすめです。また、『最短距離でゼロからしっかり学ぶ Python 入門』は生徒に対してプロジェクトベースでプログラミングを導入したい中学校や高校の先生にとっても最適です。大学で受講しているPython講座のテキストよりもわかりやすい入門書が必要な場合は、本書が助けとなります。

なにを学ぶことができるのか?

本書の目的は、読者に一般的な良いプログラマー、もしくは特に優れたPythonプログラマーになってもらうことです。一般的なプログラミングの概念の基礎を説明することによって、プログラムを効率的に学び、よい習慣を身につけることができます。『最短距離でゼロからしっかり学ぶ Python 入門』の全体を通して学んだあとは、より高度なPythonのテクニックを学ぶ準備ができているでしょう。また、別のプログラミング言語を理解することがより簡単になっているはずです。

『最短距離でゼロからしっかり学ぶ Python 入門』の本書「必修編」では、Python でプログラムを書くために必要な基本的なプログラミングの概念を学びます。この概念は、ほとんどのプログラミング言語を学びはじめるときに共通のものです。次のことを学びます。

- データのさまざまな種類について
- データをリストや辞書に格納する方法
- データの集まりを作成し、そのデータの集まり全体に対して効率的に処理を行う方法
- while ループと if 文で特定の条件をチェックし、成功した場合はコードの特定の箇所を実行し、失敗した場合は他の箇所を実行する方法（処理を自動化する場合に非常に役に立ちます）

また、対話的なプログラムを作成するためにユーザーからの入力を受け取る方法と、有効な状態の間のみプログラムを実行しつづける方法を学びます。プログラムの一部を再利用するための関数の書き方を知ることにより、特定の処理を行うコードのブロックを一度書くだけで何度も繰り返し使用できるようになります。この考え方を拡張し、より複雑な振る舞いをするクラスを使用することで、さまざまな状況にシンプルなプログラムで対応できるようになります。加えて、一般的なエラーを適切に処理するプログラムの書き方を学びます。これらの基本的な概念を実践したあとに、いくつかのわかりやすい問題を解くための短いプログラムを作成します。最後に、コードのテストの書き方を学ぶことで中級プログラミングの第一歩を踏み出します。テストを利用することでバグの混入を心配せずにプログラムを開発できます。必修編の情報はすべて、大規模で複雑なプロジェクトに取り組むための準備に必要なものです。

『最短距離でゼロからしっかり学ぶ Python 入門』の「実践編」では、必修編で学んだことを 3 つのプロジェクトに適用します。これらのプロジェクトのいずれか、またはすべてを好きな順番で進めてください。最初のプロジェクト（第 1 章から第 3 章）では、「エイリアン侵略ゲーム」というスペースインベーダーのようなシューティングゲームを作成します。このゲームはレベルが上がるとだんだん難しくなります。このプロジェクトを完了すると、2D ゲームを開発できるようになります。

2 番目のプロジェクト（第 4 章から第 6 章）はデータの可視化を紹介します。データサイエンスは、大量の情報にさまざまな可視化の技術を適用することでデータを理解することを目標としています。コードから生成したデータセット、インターネット上からダウンロードしたデータセット、またはプログラムで自動的にダウンロードしたデータセットを使用します。このプロジェクトを完了すると、大量のデータを詳しく調査し、格納された情報を可視化して表現するプログラムを書けるようになります。

3 番目のプロジェクト（第 7 章から第 9 章）は「学習ノート」という名前の小さな Web アプリケーションを構築します。このプロジェクトでは特定のトピック（話題）に関するアイデアやコンセプトを日記にして保管します。異なるトピックに対して別々のログで保存し、他のユーザーがアカウントを作成して自分の日記を書きはじめられるようにします。誰もがインターネット上のどこからでもアクセスできるように、プロジェクトをデプロイする方法も学びます。

この章では**関数**の書き方について学びます。関数は特定の処理を行うコードブロックに名前をつけたものです。関数で定義した特定のタスクを実行する際には、その関数を**呼び出し**ます。あるタスクをプログラム中で複数回実行する場合、そのタスクを実行する同じコードを何度も書く必要はありません。そのタスクの処理を行う専用の関数を呼び出すだけで、Pythonは関数の中にあるコードを実行します。関数を使用することで、プログラムの作成、内容の把握、テスト、修正が容易になります。
この章では、他に次のことについて学びます。

- 関数に情報を渡す方法
- 受け取った情報を出力する関数を書く方法
- データを処理して1つまたは複数の値の集まりを返す関数を書く方法
- **モジュール**と呼ばれる別のファイルに関数を格納し、メインプログラムから呼び出す方法

関数を定義する

次に示すのは、あいさつメッセージを表示するgreet_user()というシンプルな関数です。

greeter.py

```
❶ def greet_user():
❷     """シンプルなあいさつメッセージを出力する"""
❸     print("こんにちは！")

❹ greet_user()
```

この例は、もっともシンプルな関数の構造を表しています。defキーワードを使用することで、関数を定義することをPythonに伝えます❶。これは**関数の定義**と呼ばれ、Pythonに関数の名前を提示し、必要であれば関数が動作するために必要な情報の種類を記述するものです。情報は丸カッコの中に入ります。この例で関数の名前はgreet_user()であり、この関数の動作に追加の情報は不要なのでカッコの中は空になっています（その場合でも丸カッコは必要です）。関数定義はコロン（:）で終わります。

def greet_user():のあとのインデントされた行に関数の**実体**を記述します。次の行のコメントは**docstring**（ドックストリング）と呼ばれるもので、この関数の動作に関する説明を記述します❷。docstringは3重クォート（"""）で囲み、Pythonはプログラム中における関数のドキュメント生成にこの説明を使用します。

print("こんにちは!")の行はこの関数の実体である唯一のコードであり、greet_user()関数で実行されるのはprint("こんにちは!")という1つの動作だけです❸。

この関数を使用するには、関数を呼び出します。**関数呼び出し**は、Pythonに関数の中のコードを実行するように指示します。関数を**呼び出す**には、関数名を書いてその後ろの丸カッコの中に必要な情報を記述します❹。ここでは情報が不要なため、単純にgreet_user()と入力するだけで関数を呼び出せます。関数を実行すると「こんにちは!」と出力されます。

```
こんにちは！
```


関数に情報を渡す

greet_user()関数を少し変更し、「こんにちは！」だけではなく、ユーザーの名前も含んだあいさつメッセージを出力するようにします。関数がユーザーの名前を受け取れるように、関数定義のdef greet_user()の丸カッコ内にusernameと記述します。usernameを追加することで、関数はusername変数で任意の値を受け取れるようになります。この関数は、呼び出し時にusernameの値を要求するようになりました。greet_user()関数を呼び出す際に、丸カッコの中に'jesse'のような名前を渡すことができます。

```
def greet_user(username):
    """シンプルなあいさつメッセージを出力する"""
    print(f"こんにちは{username.title()}！")

greet_user('jesse')
```

greet_user('jesse')と入力すると、greet_user()関数が呼び出され、print()の呼び出しに必要な情報が渡されます。関数は渡された名前を受け取り、名前を含めたあいさつメッセージを出力します。

```
こんにちはJesse！
```

同様にgreet_user('sarah')と入力すると、greet_user()関数は'sarah'を受け取って呼び出され、「こんにちはSarah！」と出力します。greet_user()関数は何度でも呼び出すことができ、指定した名前によって想定された出力が生成されます。

実引数と仮引数

前の例のgreet_user()関数では、username変数の値を要求するようにgreet_user()を定義しました。そのため、この関数を呼び出すときに情報（人の名前）を渡すと、正しいあいさつメッセージが出力されました。

greet_user()関数の定義に出てくるusername変数は、**仮引数**（parameter）の一例です。仮引数とは、関

やすいコードを書くために1行の文字数を79文字以下にすることが推奨されています。複数の仮引数によって関数定義が79文字を超える場合は、関数定義の開きカッコ（(）のあとに改行を入力します。次の行でTabキーを2回押し、関数のボディ部より1レベル深いインデントで仮引数のリストを記述します。

多くのテキストエディターでは、追加される仮引数のリストのインデントが自動的に揃います。

```
def function_name(
        parameter_0, parameter_1, parameter_2,
        parameter_3, parameter_4, parameter_5):
    関数のボディ部...
```

プログラムやモジュールに2つ以上の関数がある場合は、関数の間に2行の空行を入れることで関数の終わりと次の関数の始まりがわかりやすくなります。

すべてのimport文はファイルの先頭に記述します。唯一の例外は、ファイルの先頭にコメントでプログラム全体の説明文を記述する場合です。

やってみよう

8-15. モデルを印刷する

printing_models.pyの中にある関数をprinting_functions.pyという別のファイルに移動します。printing_models.pyの先頭にimport文を追加し、インポートした関数を使用するように書き換えます。

8-16. インポート

1つの関数を使用するプログラムを書き、その関数を別のファイルに格納します。メインのプログラムファイルで次の各手法により関数をインポートし、その関数を呼び出します。

```
import module_name
from module_name import function_name
from module_name import function_name as fn
import module_name as mn
from module_name import *
```

8-17. 関数のスタイル

この章で作成したプログラムを3つ選び、この節で学んだスタイルのガイドラインにしたがっていることを確認します。

まとめ

この章では、次のことについて学びました。

- 関数の書き方と、渡された実引数の情報に関数からアクセスする方法
- 位置引数、キーワード引数、可変長引数、可変長キーワード引数の使い方
- 関数で結果を出力する場合と戻り値を使用する場合の違い
- リスト、辞書、if文、whileループとあわせて関数を使用する方法
- 関数をモジュールと呼ばれる別ファイルに格納し、プログラムを読みやすくわかりやすくする方法
- 関数にスタイルを適用し、構造化された読みやすいコードを書く方法

プログラマーとしての目標の1つは、実現したいことに対してシンプルなコードを書くことです。関数はそういった目標を達成するのに役立ちます。関数にはコードの固まりを書くことができ、正しく動作すれば書き換えは不要です。関数が正しく動作していることがわかっていれば、その関数の動作を信頼して次のコーディング作業に移ることができます。

関数は、コードを一度書くだけで必要なときに何度も再利用できます。関数のコードを実行したい場合は、関数を呼び出すコードを1行書くだけです。関数の動作を変更したい場合は、関数内のコードを一部書き換えるだけで関数を呼び出している箇所すべてを同じように変更できます。

関数を使用するとプログラムが読みやすくなります。そして、よい関数名はプログラムの各部分の実行する内容を要約します。一連の関数呼び出しを読むことで、長いコードブロックを読むよりも素早くプログラム全体を理解できます。

また、関数によってコードのテストやデバッグも容易になります。プログラムの大部分が一連の関数によって実現されている場合、コードのテストと保守がとてもやりやすくなります。各関数を呼び出す別々のプログラムを作成することで、それぞれの関数が遭遇するであろうすべての状況で正しく動作するかをテストできます。このようなテスト用のプログラムを実行することにより、関数が厳密に正しく動作することを確認できます。

第9章では、クラスの書き方について学びます。**クラス**とは、関数とデータの組み合わせを1つのパッケージに整理し、柔軟に効率的に使用できるようにしたものです。