おわりに

ようやく、プログラミングを含めたPythonの基礎の学習が終了しました。

みなさんは、学習の途中で、いろいろなことに気付いたでしょう。たとえば、

『変数、さらには、変数への代入とは、こういう意味だったのか。』

『この機能を使えば、最初の頃に作ったプログラムは、もっと簡潔に実現できる。』

といった感じです。

もちろん、このようなことは、Pythonに限ったことではなくて、すべての道の学習に共通です。どんな道であっても、最初から、その道の《全体像》を完全に知りつくした上で学習することは不可能（あるいは、極めて困難）だからです。

本書をまとめるにあたっては、みなさんが、全体像を見失うことなく、Python言語の道、Python言語を用いたプログラミングの道を少しずつ歩めるように心がけました。そのため、最初の段階では、難しいことや細かいことなどをわざと隠して解説しておき、後から“種明かし”を行うこともありました。

もっとも、本文を約400ページの分量で収めたこともあり、ジェネレータ、クラスの応用、応用的なアルゴリズム、スレッドプログラミング、GUIなどの題材は、本書では取り上げていません。そのため、すべての種明かしが完了したわけではありません（ただし、次のステップに進みやすいように配慮していますので、安心してください）。

*

これまでに、本当に数え切れないくらいの人数の、学生やプロのプログラマの方々を対象に、プログラミングやプログラミング言語を指導してきました。受講者が100人いれば、100種類のテキストが必要となるのではないか、と感じるくらい、学習の目的・学習の進度・理解の様子など、あらゆる点が個人ごとに大きく異なります。

たとえば、学習の目的もさまざまです。『趣味として勉強したい。』、『プログラミングを専門としない学部学科に所属しているけれど単位取得のために学習しなければならない。』、『情報系を専門とする学生であって、その修得が必須である。』、『プロのゲームプログラマになりたい。』といった感じです。

本書は幅広い読者層を想定して、簡単になりすぎないように、かつ、難しくなりすぎないように配慮しました。それでも、本書を簡単に感じた方もいらっしゃるでしょうし、あまりにも難しいと感じた方もいらっしゃるでしょう。

ここでは、本書を読み進める上で、参考としていただきたいポイントを示します。

▶ 本書を読んで、次のようなことを感じられた方もいらっしゃるでしょう。

『こんな知識（たとえば専門用語の英語表記）は、私には不要だ。』、『似たようなプログラムが多すぎる。』、『なぜ、こんな細かいことまで解説しているのだろうか。』、『章の構成がおかしいのではないか。』、『実際のソフトウェア開発では、こんなプログラムは作らないはずだ。』…。

本書は、幅広い層の読者を想定した上で、私なりに考え抜いた上で執筆したものです。以下の解説は、これまで、（私の他の書籍に対して）いただいた質問やご意見に対する回答ともなっています。

▪ 専門用語について

本書の専門用語は、“**キーワード**（*keyword*）”のスタイルで、日本語と英語の語句を併記しています。英語の語句は、原則として、Python ソフトウェア財団のドキュメントに準拠しています。日本語の語句（訳語）は、同財団のサイト上の日本語ドキュメントを参考にしていますが、不適切と感じられる訳語については、独自の訳語をあてています。

情報系の大学生であれば、プログラミング関連の英語の専門書を読むことになります。本書に示している英語の専門用語は、ほとんどが基本的な語句ですから、すべてを習得しておくべきです（大学院生であれば、なおさらです）。

▪ 類似したプログラムリストの掲載について

ある目的を実現するプログラムは、簡単なものでも数通りの実現法があり、複雑なものになると、無限ともいえる実現法があります。

そのため、本書では、学習の各段階にそった実現法を数多く示しています。たとえば、1 から `n` までの和を求めるプログラムは、**List 4-3**（p.92）では `while` 文によるプログラムを学習し、p.181 では、`sum` 関数と `range` 関数を組み合わせた方法を学習しました。この例に限らず、本書のプログラムは、より高度な技術を使うと、もっと簡潔に、あるいは、実用的に使えるように実現できるものが数多くあります（先ほども申し上げましたように、すべての種明かしが終わったわけではありません）。

▪ 章の構成について

理解の早い読者の方や、他のプログラミング言語の経験者の方などは、なかなか先に進まないことをもどかしく感じ、後半の章を物足りなく感じられたかもしれません。

しかし、このような構成としている理由の一つが、`if` 文（第3章）や繰返し文（第4章）の段階で挫折する学習者が決して少なくないことです。事実、『`if` 文や繰返し文の段階で多くの学生がつまずいてしまう。どのように教えればよいのでしょうか。』との相談を、教育現場の先生方から数多くいただいています。

算数の学習の最初の段階を考えてみましょう。まず数の基本から始まって、足し算や引き算などを学習していきます。たとえば「1 ＋ 3 を求める問題」を解きます。もちろん、現実の世界では、「1 ＋ 3 を求める問題」自体を解くことはありません（足し算は、たとえば技術計算やお金の計算などの一部として使われるだけです）。

また、「3 × 5」は、ほとんどの方が「15」という解を瞬時に導けるはずです。しかし、掛け算を学習する前は、“3 を 5 回加える”という足し算として行っていたでしょう。

『最初から○○（たとえば、オブジェクト指向プログラミング）を教えるべきだ。』とか『掲載されているプログラムが実用的でない。』というご意見をいただくのですが、足し算を知らない（あるいは、足し算を難しく感じる）人の存在を完全に無視して、『最初から“掛け算”や“方程式を解く方法”を教えるべきだ。』という意見に感じられます。

もちろん、たとえば Java などの他のオブジェクト指向プログラミング言語に習熟している

人だけを読者として想定するのであれば、「すべての型はクラスである。」、「すべてのクラスは、`object` クラスの派生クラスである。」といったところから開始することが可能です。とはいえ、数値計算だけのために Python を使う、という方々も数多くいらっしゃいます（プログラミング言語に限らず、どんなものであっても、その用途は、使う人によって決められます）。

しかも、入門書というものは、教育の現場によって、まったく異なる位置付けで利用されます。たとえば、本書のような入門用テキストを使う基礎的なプログラミングの他に、計算機工学、アルゴリズムとデータ構造、オブジェクト指向プログラミングといった講義科目を教えるカリキュラムもあります。このようなカリキュラムであれば、たとえば“探索”や“ソート”、“デザインパターン”などは、他のテキストでの学習が可能です。

その一方で、本書のような入門用テキスト1冊だけでプログラミングを教えるカリキュラムもあります。その場合、“探索”や“配列をある程度自在に操るための技術”などの学習が必要です。

多様な教育現場やテキストの使われ方などを広く深く考慮すると、上級者やプロの方々が想像する以上に、基礎に重点を置かざるを得ない、というのが私の持論です。

最後となりますが、本書をまとめる上で気になったことをお伝えいたします。それは、まったくの知識不足のもとで書かれた、明らかに“嘘”の内容の書籍やWebサイトが、あまりにも数多く見受けられた、ということです。

以下、気になった点の、ごく一部を示します。

✕変数には記憶寿命（記憶域期間）がある。

記憶寿命（記憶域期間）とは、たとえば、『関数の中で定義されたオブジェクトは、その関数が実行されているあいだのみ存在する』といった、C言語などで使われる概念です。オブジェクトありきのPythonでは、変数は、オブジェクトを参照する（結び付いた）名前にすぎません。そのため、関数の実行の開始と終了に伴って、オブジェクトが生成されたり、破棄されたりすることはありません（たとえば、**List 9-36**（p.277）のプログラムからも明らかです。このプログラムの整数オブジェクト`1`，`2`，`3`，`4`，`5`，`6`が、各関数の開始と終了のタイミングで作られたり破棄されたりすることはありません）。

記憶寿命という概念が、Pythonというプログラミング言語に存在し得る余地は、まったくありません（この点については、**Column 5-1**（p.119）でも簡単に補足しています）。

✕論理演算子`and`演算子と`or`演算子は、`True`あるいは`False`の論理値を生成する。

論理式“`x and y`”や“`x or y`”の評価で得られるのが、`x`もしくは`y`のいずれかであることは、第3章のp.56～p.59で詳しく学習しました。たとえば、式`5 or 3`の評価によって得られるのは、`True`ではなく`5`です。

`x`や`y`自体が論理値でない限り、`and`演算子と`or`演算子は`True`や`False`を生成しません。

そもそも、真（*true*）であることと`True`は同一ではありませんし、偽（*false*）であることと`False`は同一ではありません（p.53）。

論理式“x and y”や“x or y”が生成するのが、xあるいはyであることを利用するプログラミングテクニックが、Pythonで有効に利用されることも本書で学習しました。

×if文は判定式がTrueのときにスイートを実行する。

if文の判定式は、論理値（TrueまたはFalse）である必要はありません。“if 5:”とか、“if [1, 2, 3]:”など、整数、リストなど何でもOKです。というのも、すべての値は、真あるいは偽とみなされるからです。

ちなみに、if文を含めた「複合文」が制御するスイートを、頭部（ヘッダ）の次の行に書かなければならない、という誤った解説も数多く見受けられます（そうでないことは、p.65 〜 p.69で詳しく学習しました）。

×代入演算子は右結合の演算子である。

代入文で使われる=が演算子でないことは、第1章から繰り返し学習しました。演算子ではないため、右結合とか左結合などの結合性は存在しません。

代入演算子が右結合であるという誤った前提に基づいて、“a = b = 1”が“a = (b = 1)”とみなされるという解説を見受けますが、“a = (b = 1)”はPythonではエラーとなります。

×関数の引数の受渡しは「値渡し」あるいは「参照渡し」で行われる。

Pythonにおける関数間の引数の受渡しは、「値渡し」でも「参照渡し」でもないことを、第9章のp.250 〜 p.253で詳細に学習しました。引数の受渡しは、「実引数が仮引数に代入される」という極めてシンプルな規則に基づいて行われます（もちろん、代入されるのは、値ではなく、オブジェクトへの参照です）。

引数がイミュータブルな型であるかどうかで、見かけ上の挙動が変わるだけであって、引数の型や性質に応じて、「値渡し」と「参照渡し」が使い分けられる、といったこともありません。

*

ごく一部を示しました。上記の指摘は、私の個人的な見解ではなく、すべてPythonの公式ドキュメントの内容に基づいたものです。

なお、ここに示した指摘のほとんどが当てはまるテキストを見受けました。そのようなテキストで学習すると、Pythonの本質を誤解してしまうのではないかと危惧します。

*

本書では「変数は、値を格納する箱のようなものである。」と最初に“嘘”をついておき、その後で“種明かし”をしました。

リファレンスマニュアルではなく入門書である、という性格上、数多くの点で、文法規則やライブラリの仕様などの細かい部分は“切り捨てて”解説していますので、本書にも嘘が含まれています。

世の中にはいろいろな情報が溢れています。Pythonに限らず、いろいろな情報は、鵜呑（うの）みにすることなく選別し、ご自身で裏を取って、判断いただきたいと考えます。