Pythonで
いかにして暗号を破るか

古典暗号解読プログラムを自作する本

AL SWEIGART 著　　IPUSIRON 訳

no starch press

ソシム

本書のサポートページについて

本書のサポートページは、次のURLよりアクセスできます。

https://www.socym.co.jp/book/1216

商標などについて

- 本書に記載されている社名、製品名、ブランド名、システム名などは、一般に商標または登録商標で、それぞれ帰属者の所有物です。
- 本文中では、©、®、™は表示していません。

諸注意

- 本書に記載されている情報は、2020年12月現在のものであり、URLなどの各種の情報や内容は、ご利用時には変更されている可能性があります。
- 本書の内容は参照用としてのみ使用されるべきものであり、予告なしに変更されることがあります。また、ソシム株式会社がその内容を保証するものではありません。本書の内容に誤りや不正確な記述がある場合も、ソシム株式会社は一切の責任を負いません。
- 本書に記載されている内容の運用によっていかなる損害が生じても、ソシム株式会社、著者、訳者は責任を負いかねますので、あらかじめご了承ください。
- 本書のいかなる部分についても、ソシム株式会社との書面による事前の同意なしに、電気、機械、複写、録音、その他のいかなる形式や手段によっても、複製、および検索システムへの保存や転送は禁止されています。

第 2 章

対話型シェルのプログラミング

「解析機関は何かを創作するといった気取ったことはしない。実行させるにどうやって命令したらよいか我々がわかっていることであれば、解析機関は何だってできる」

——Ada Lovelace、1842 年 10 月

暗号化プログラムを作成する前に、基本的なプログラミングの概念をいくつか学ぶ必要があります。これらの概念には、値、演算子、式、および変数が含まれます。

TOPICS

本章で扱うトピック

- 演算子
- 値
- 整数と浮動小数点数
- 式
- 式の評価
- 変数への値の代入
- 変数の上書き

まず、Pythonの対話型シェルで簡単な計算を実現する方法を見ていきます。コンピュータの隣で本書を開き、短いコードを入力して、その動作を確認してください。プログラムを手入力することにより身体で覚えられ、Pythonコードを作成する方法を習得しやすくなります。

2.1 いくつかの簡単な数式

まず、IDLE（**P.025 のIDLEを開始する**を参照）を開きます。対話型シェルが表示され、>>>プロンプトの隣でカーソルが点滅します。対話型シェルは電卓のように動作します。対話型シェルにて、キーボードから 2 + 2 と入力し、Enter キーを押します（一部のキーボードでは return キーを押します）。**図 2-1** のように、コンピュータは 4 という数を表示します。

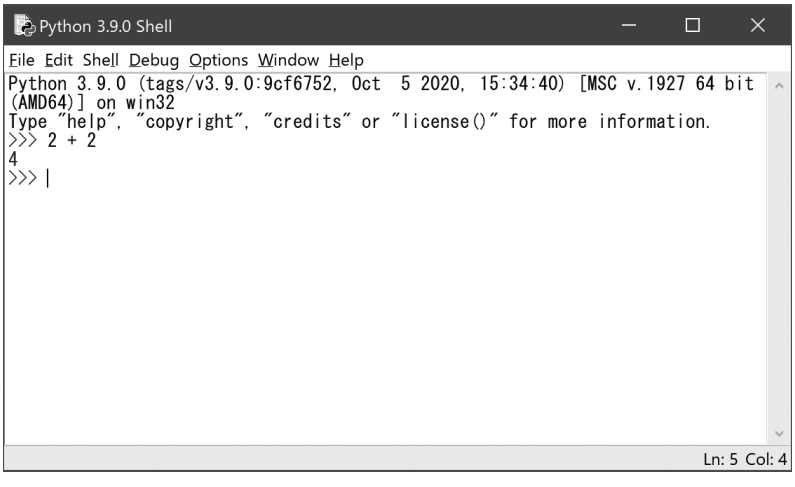


図 2-1 対話型シェルに 2 + 2 を入力する

図 2-1 の例において、+記号は数 2 と 2 を加算することをコンピュータに指示します。他にも、Pythonはマイナス記号（-）で減算、アスタリスク（*）で乗算、スラッシュ（/）で除算といった、その他の計算ができます。このように使用するとき、+、-、*、/ は**演算子**と呼ばれます。なぜなら、それらの記号は前後の数に対する操作をコンピュータに指示するからです。**表 2-1** はPythonの代数演算子をまとめたものです。2（または他の数）は、**値**と呼ばれます。

表 2-1　Python の代数演算子

演算子	操作
+	加算
-	減算
*	乗算
/	除算

2 + 2 はプログラムではなく、単なる命令です。プログラムはこれらの命令の組み合わせから成り立ちます。

2.1.1 整数と浮動小数点数

プログラミングにおいて、4、0、99 などのすべての数は**整数**と呼ばれます。小数点を持つ数（3.5、42.1、5.0）は**浮動小数点数**と呼ばれます。Pythonで数 5 は整数です。しかし、5.0 と表記した場合は浮動小数点数になります。

整数と浮動小数点数は**データの型**です。42 は整数の値であり、**int**という型になります。7.5 は浮動小数点数であり、**float**という型になります。

すべての値には型があります。他のいくつかの型（**3 章**での文字列など）について解説します。値について議論するとき、その値は特定の型を持つことを覚えておいてください。値がどのように書かれているかを見るだけで、その型を容易に識別できます。intは小数点のない数です。floatは小数点を持つ数です。よって、42 の型はintであり、42.0 の型はfloatです。

2.1.2 式

すでにPythonで 1 つの計算問題を解きましたが、Pythonはもっと多くのことを実行できます。対話型シェルに次の計算を入力し、Enter キーを押します。

```
❶ >>> 2+2+2+2+2
  10
  >>> 8*6
  48
❷ >>> 10-5+6
  11
❸ >>> 2   +        2
  4
```

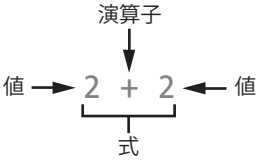


図 2-2　**式は値（2 など）と演算子（+ など）によって構成されている**

このような数学の問題を**式**と呼びます。コンピュータは数百万の問題を数秒で解いてしまいます。**図 2-2** のように、式は演算子（数学記号）によって接続された値（数値）で構成されます。式の中に必要な数だけ数値を入れられます❶。それらが演算子で接続されている限り、1 つの式で複数の種類の演算子を使用できます❷。整数とこれらの演算子の間には、任意の数のスペースを入力しても構いません❸。しかし、行頭の空白は、Pythonが指示を解釈する方法を変えてしまうので、行頭には空白を入れないようにします。行頭の空白については、**P.079** の「**ブロック**」で解説します。

2.1.3 演算の順序

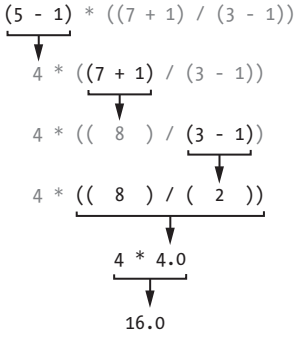
数学の授業で演算の順序について教わったことでしょう。例えば、乗算は加算より先に行います。式 `2 + 4 * 3` は 14 と評価されます。なぜならば、最初に `4 * 3` が評価され、それから 2 が加算されるからです。括弧を用いて、別の演算子を先に操作できます。式 `(2 + 4) * 3` では、始めに `(2 + 4)` が評価され、それからその合計値に 3 を掛けます。括弧により、この式は 14 ではなく、18 と評価されます。Pythonでの演算子の順序（**優先順位**とも呼ぶ）は、数学の場合と同様です。括弧内の演算は最初に評価されます。次に、演算子`*`と`/`が左から右に評価されます。その後、演算子`+`と`-`が左から右に評価されます。

2.1.4 式の評価

コンピュータが式 `10 + 5` を解き、結果として値 15 を得たとき、式が**評価された**といいます。式を評価することで、式を単一の値に換算できます。これは、さながらたった 1 つの数、つまり答えを求める数学の問題を解くようなものです。

式 `10 + 5` と `10 + 3 + 2` は、両方とも 15 に評価されるため、同じ値を持ちます。単一の値でも式と見なされます。式 15 は値 15 に評価されます。

Pythonは次のように単一の値になるまで式を評価し続けます。



Pythonはもっとも内側にある、左端の括弧で始まる式から評価します。括弧が入れ子になっている場合でも、中の式は他の式と同じ規則で評価されます。そのため、Pythonでは`((7 + 1) / (3 - 1))`を次のようにして解きます。まずもっとも内側で、左端の括弧内の式`(7 + 1)`を解きます。次に、その右の式`(3 - 1)`を解きます。内側の括弧内の各式を単一の値にすると、外側の括弧内の式が評価されます。除算は浮動小数点数の値に評価されることに注意してください。最終的に、括弧内に式がなくなると、Pythonは演算の残りの計算を実行します

式では 2 つ以上の値を演算子で接続したり、1 つの値だけを持たせたりできます。しかし、対話型シェルに 1 つの値と演算子を入力するとエラーメッセージが表示されます。

```
>>> 5 +
SyntaxError: invalid syntax
```

`5 +`は式ではないので、このエラーが発生しました。複数の値を持つ式は、それらの値を接続する演算子が必要です。Pythonでは+演算子が 2 つの値を接続するものと想定しています。**構文エラー（syntax error）**は、誤った入力のためにコンピュータが指示を理解できないことを意味します。これは重要でないように思われるかもしれませんが、プログラミングではコンピュー

タに何をすべきかを伝えるだけでなく、コンピュータの指示にしたがう正しい方法を知ることにもなります。

COLUMN

エラーでも問題なし

エラーが起きても問題ありません。エラーの原因となるコードを入力しても、コンピュータが壊れるわけではありません。Pythonは単にエラーが発生したことを伝え、再び>>>プロンプトを表示します。対話型シェルで新しいコードの入力を続行できます。

プログラミングの経験を積むまでは、あなたにとってエラーメッセージはあまり意味がないかもしれません。エラーメッセージのテキストをGoogleで検索して、その特定のエラーについて解説するWebページを見つけられます。一般的なPythonのエラーメッセージとその意味については、本書のサポートページ（原著者による英文解説となります）で参照できます。

2.2 変数への値の代入

プログラムでは後で使用する値を保存しておく必要があります。=記号（**代入演算子**と呼ぶ）を使って、**変数**の中に値を保存できます。例えば、変数spamに値15を格納するためには、対話型シェルにspam = 15を入力します。

```
>>> spam = 15
```

変数は値15を内部に持つ箱のようなものとイメージできます（**図2-3**を参照）。変数名はspamであり、これは箱のラベルといえます（別の変数と区別するためのもの）。そして、その中に格納されている値は、箱の中のメモといえます。

Enterキーを押すと、何も表示されません。エラーメッセージが表示されない限り、命令が正常に実行されたものとみなせます。そして、>>>プロンプトが表示され、次の指示を入力できます。

代入（=演算子）がある指示（**代入文**と呼ぶ）は、変数spamを作り、そこに値15を格納します。

式とは異なり、**文（statements）**はどんな値にも評価されない命令です。代わりに、指示だけを実行します。このため、対話型シェルにおいて次の行に値が表示されません。

どの命令が式であり、どれが文であるかを理解することは紛らわしいかもしれません。Python命令が単一の値に評価される場合、それは式になります。そうでなければ、文になります。

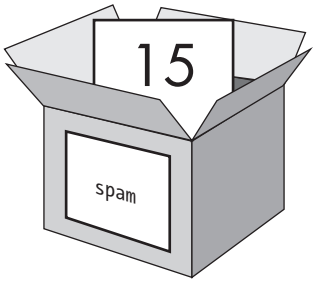


図 2-3　**変数は値を保持した、名前を持つ箱のようなものである**

代入文は**図 2-4** のように、変数を記述した後に、=演算子が続き、最後に式が来ます。式を評価した値は、変数に格納されます。

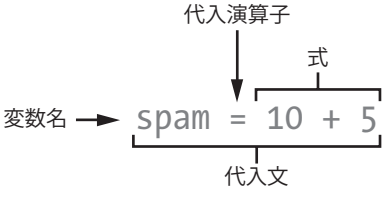


図 2-4　**代入文の各部分**

変数には割り当てられた式だけでなく、単一の値が格納されることに注意してください。例えば、文`spam = 10 + 5`を入力すると、最初に式`10 + 5`が`15`に評価され、値`15`が変数`spam`に格納されます。対話型シェルにその変数名を入力することで、変数の中身を確認できます。

```
>>> spam = 10 + 5
>>> spam
15
```

変数は格納されている値に評価される式といえます。値自体はそれ自身を評価する式でもあります。

```
>>> 15
15
```

ここで、興味深いことがあります。対話型シェルにspam + 5を入力すると、整数値20が得られます。

```
>>> spam = 15
>>> spam + 5
20
```

見てわかるように、変数は値と同様にして式の中で使用できます。spamの値は15であるため、式spam + 5は式15 + 5に評価され、最終的に20に評価されます。

2.2.1 変数の上書き

変数内の値は別の代入文により変更できます。例えば、次のように入力します。

```
>>> spam = 15
❶ >>> spam + 5
❷ 20
❸ >>> spam = 3
❹ >>> spam + 5
❺ 8
```

最初にspam + 5❶を入力します。その式は20❷に評価されます。なぜなら、変数spamに値15が格納されているからです。一方、**図2-5**のように、spam = 3❸を入力すると、値15は値**3に上書きされます**（すなわち、置換されます）。ここで、spam + 5❹を入力すると、式は8❺に評価されます。なぜなら、spam + 5は3 + 5に評価されるからです。spamに格納されていた古い値は消去されます。

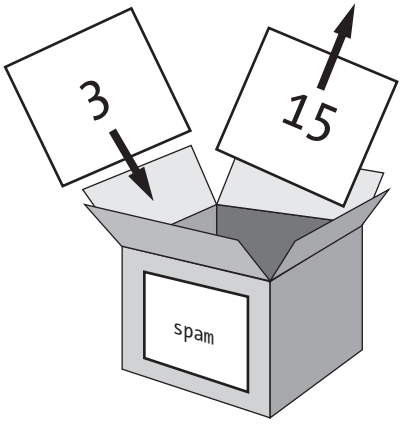


図 2-5　spam 内の 15 は、値 3 に上書きされる

spamに新しい値を代入するために、変数spamの値を使うこともできます。

```
>>> spam = 15
>>> spam = spam + 5
>>> spam
20
```

代入文spam = spam + 5 は、変数spamの現在値に 5 を足して、その結果をspamに上書きすることをコンピュータに指示します。右側の式の結果は、記号=の左側の変数に代入されます。数回繰り返すと、spamの値を 5 ずつ増やせられます。

```
>>> spam = 15
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam
30
```

spam内の値は、spam = spam + 5 が実行されるたびに変化します。spam内の値は、最終的に 30 になります。

2.2.2 変数名

コンピュータは変数の名前を気にしませんが、読者のあなたは気にすべきです。どのような

種類のデータを含んでいるのかがわかる変数名であれば、プログラムを理解しやすくなります。abrahamLincolnやmonkeyといった変数名を付けたとします。プログラムがアブラハム・リンカーンや猿と何の関係もなくても、(首尾一貫してabrahamLincolnやmonkeyを使用する限り)プログラムは実行されます。しかし、後日そのプログラムを見たとき、あなたはその変数が何をしているかを覚えていないかもしれません。

よい変数名とは、含まれるデータを判別できるものなのです。新しい家に引っ越しするとき、運ぶ箱に**Stuff (荷物)** というラベルを張ったとします。これでは、目的のものがどの箱に入っているのかわかりません。spam、eggs、baconなど(**Monty Python**の寸劇「スパム」から引用した)の変数名は、本書や多くのPython本の例でよく使われています。しかし、プログラムを書くときは、わかりやすい名前を用いて読みやすくすべきです。

Pythonだけではありませんが、変数名は大文字と小文字を区別します。**大文字と小文字を区別するということ**は、同じ変数名でも、一部の大文字・小文字が異なれば、異なる変数と見なされることを意味します。例えば、spam、SPAM、Spam、sPAMは、Pythonにて 4 つの異なる変数と見なされます。これらの変数はそれぞれ別の値を持つので、同じ意味として使用できません。

2.3 まとめ

暗号プログラムをいつ作成し始めるのでしょうか？　それはもうすぐです。しかし、暗号を解読する前に、基本的なプログラミングの概念をいくつか学ぶ必要があります。そのため、もう 1 つのプログラミングに関する章を読む必要があります。

本章では、対話型シェルでPython命令を書く基本的な方法を学びました。コンピュータはとても単純な命令だけしか理解できないので、何をすべきかをPythonに正確に伝える必要があります。Pythonが式を評価できること(つまり、式を単一の値に換算すること)、と式が演算子(+や-)と値(2 や 5 など)の組み合わせであることを学びました。変数に値を格納でき、後で使用するために変数を覚えておくこともできます。

対話型シェルは、Pythonの指示を理解するために便利なツールです。一度に 1 つずつ入力して結果を確認できるからです。**3 章**では一度に 1 つではなく、順番に実行される多くの命令を含むプログラムを作成します。いくつかの基本的な概念について説明し、最初のプログラムを作成する予定です。

練習問題

練習問題の答えは、本書のサポートページにあります。

問 1 除算の演算子は、/と\のどちらですか？

問 2 次の値は整数と浮動小数点数のどちらですか？

```
42
3.141592
```

問 3 次の行のうち、式では**ない**ものはどれですか？

```
4 x 10 + 2
3 * 7 + 1
2 +
42
2 + 2
spam = 421
```

問 4 対話型シェルに次の行のコードを入力すると、行❶と❷では何を表示しますか？

```
spam = 20
❶ spam + 20
SPAM = 30
❷ spam
```