

プログラミング体験実習の手引き

第二版

2016年6月14日(火)

Copyright © 2014–2016 Daikoku Manabu

This tutorial is licensed under a Creative Commons Attribution 2.1 Japan License.

Lesson 1 プログラム

- 1.1 この体験実習では、プログラミングというものを体験してもらいます。
- 1.2 プログラミングというのは、「プログラム」と呼ばれる文書を書くということです。
- 1.3 「プログラム」というのは、コンピュータに実行してほしい動作を言葉で書きあらわしたもののことです。コンピュータが実行する動作は、すべて、人間がプログラムを書くことによって実現されています。
- 1.4 プログラムは、「ソフトウェア」とか「ソフト」とか「アプリ」などと呼ばれることもあります。この体験実習では、「プログラム」という言葉で統一することにします。
- 1.5 プログラムを書くためには、「プログラミング言語」と呼ばれる専用の言語を使う必要があります。
- 1.6 プログラミング言語には、たくさんものがあります。この体験実習では、Python というプログラミング言語を使います。

Lesson 2 端末

- 2.1 プログラミングを始める前に、まず、「端末」と呼ばれるプログラムの使い方に慣れておきましょう。
- 2.2 「端末」というのは、「シェル」と呼ばれるプログラムを実行するための環境を作り出すプログラムのことです。
- 2.3 「シェル」というのは、「コマンド」と呼ばれるものを読み込んで、それを実行するプログラムです。
- 2.4 「コマンド」というのは、コンピュータに実行してほしいことを1行で書きあらわしたもののことです。つまり、1行だけでできたプログラムのことだと考えることができます。
- 2.5 シェルは、コマンドが入力されるのを待っている状態だということを示すために、「プロンプト」と呼ばれる文字列を画面に表示します。
- 2.6 シェルに対して、プログラムの名前をコマンドとして入力すると、その名前のプログラムが起動します。たとえば、シェルに `cal` と入力すると、その名前のプログラムが起動して、今月のカレンダーが出力されます。
- 2.7 プログラムの名前のうしろに、「引数」と呼ばれる単語を書くことによって、プログラムが実行するべき動作を指定することもできます。たとえば、
`cal 月 年`
というコマンドを入力することによって、指定された年と月のカレンダーを出力させることができます。

Lesson 6 関数

- 6.1 動作をあらわしているデータは、「関数」と呼ばれます。
- 6.2 関数に、それがあらわしている動作を実行させることを、関数を「呼び出す」と言います。
- 6.3 関数は、データを受け取って、それを処理して、その結果として得られたデータを返します。関数が受け取るデータは「引数」と呼ばれ、関数が返すデータは戻り値と呼ばれます。
- 6.4 たとえば、lenという関数は、引数として文字列を受け取って、その長さ（文字数）を戻り値として返します。
- 6.5 関数は、
関数名(式, ...)
という形の式を評価することによって呼び出すことができます。たとえば、
len('umiushi')
という式を評価することによって、lenという関数を呼び出して、それに対して引数としてumiushiを渡すことができます。このような式は、「関数呼び出し」と呼ばれます。
- 6.6 関数が戻り値として返したデータは、その関数を呼び出した関数呼び出しの値になります。
- 6.7 それでは、lenという関数を呼び出す式をインタラクティブシェルに入力してみてください。

Lesson 7 組み込み関数とユーザー定義関数

- 7.1 動作を記述することによって関数を作ることを、関数を「定義する」と言います。
- 7.2 関数を定義するために書く記述は、「関数定義」と呼ばれます。
- 7.3 lenのように、言語処理系に最初から組み込まれていて、関数定義を書かなくても使うことができる関数もあります。そのような関数は、「組み込み関数」と呼ばれます。
- 7.4 関数定義を書くことによって定義された関数は、「ユーザー定義関数」と呼ばれます。
- 7.5 関数を定義するというのは、単純な動作を組み合わせることによって複雑な動作を作ることだと考えることができます。
- 7.6 「単純な動作を組み合わせることによって複雑な動作を作る」というのが、プログラミングというものの本質です。

Lesson 8 関数定義の書き方

- 8.1 Pythonでは、関数定義は、
def 関数名():
文
と書きます。「文」というのは、動作を記述したもののことです。関数定義の全体も、「関数を定義する」という動作をあらわすひとつの文です。

8.2 「return文」という文を書くことによって、「データを戻り値として返す」という動作を記述することができます。return文は、

```
return 式
```

と書きます。return文を実行すると、その中の式が評価されて、その値が戻り値として返されます。

8.3 次の関数定義は、引数を何も受け取らないで、戻り値として0を返す、zeroという関数を定義します。

```
def zero():
    return 0
```

8.4 それでは、この関数定義をインタラクティブシェルに入力してみましょう。

8.5 1行目を入力してエンターキーを押すと、... というプロンプトが表示されます。このプロンプトは、インタラクティブシェルが、2行以上にまたがる式や文の2行目以降を入力してほしいときに表示するものです。

8.6 2行目を入力してエンターキーを押したときも、... というプロンプトが表示されますが、3行目はありませんので、何も入力しないでエンターキーを押してください。

8.7 それでは、入力した関数定義によって定義された関数を呼び出してみましょう。

```
zero()
```

という関数呼び出しをインタラクティブシェルに入力すると、zeroが呼び出されて、0という戻り値が出力されるはずですが、

Lesson 9 仮引数

9.1 引数を受け取る関数を定義したいときは、「仮引数」と呼ばれるものを使います。

9.2 「仮引数」というのは、受け取った引数を参照するための名前のことです。

9.3 引数を受け取る関数を定義する関数呼び出しは、

```
def 関数名(仮引数, ...):
    文
```

と書きます。たとえば、

```
def namako(a, b, c):
    文
```

という関数定義によって定義されたnamakoという関数を、

```
namako(37, 24, 61)
```

という関数呼び出しで呼び出したとすると、aは37を、bは24を、cは61を参照する名前になります。

9.4 次の関数定義は、引数として数値または文字列を受け取って、それに10を掛け算した結果を戻り値として返す、tentimesという関数を定義します。

```
def tentimes(a):
    return a*10
```

9.5 それでは、この関数定義をインタラクティブシェルに入力して、tentimesを呼び出してみましょう。

Lesson 10 テキストエディター

10.1 次に、関数定義をファイルに保存して、その関数定義をインタラクティブシェルに実行させる、という実習をします。

10.2 文字だけでできているデータは、「テキストデータ」と呼ばれます（単に「テキスト」と呼ばれることもあります）。

10.3 テキストを入力したり修正したりするために使われるプログラムは、「テキストエディター」と呼ばれます（単に「エディター」と呼ばれることもあります）。

10.4 人間が書くプログラムは、テキストデータです。したがって、人間がプログラムと書くときには、テキストエディターが使われます。

10.5 この体験実習では、`gedit` というテキストエディターを使います。`gedit` は、

```
gedit ファイル名 &
```

というコマンドで起動することができます。ちなみに、コマンドの末尾のアンパサンド (&) は、シェルに対して、プログラムの終了を待たないで次のプロンプトを出すように依頼する文字です。

10.6 それでは、次のコマンドで `gedit` を起動してください。

```
gedit star.py &
```

10.7 Python のプログラムを保存するファイルの名前には、`.py` という拡張子を付けることになっています。

10.8 `gedit` を使って、次の関数定義を入力して、ファイルに保存してください。

```
def star(a):
    return '*' * len(a)
```

10.9 この関数定義は、引数として文字列を受け取って、その文字列と同じ長さを持つアスタリスクの列を戻り値として返す、`star` という関数を定義します。

Lesson 11 ファイルの中のプログラムの実行

11.1 ファイルに格納されているプログラムをインタラクティブシェルに実行させたいときは、

```
from ファイル名 import *
```

という文をインタラクティブシェルに入力します。ただし、この中の「ファイル名」のところには、ファイル名から拡張子を取り除いた部分を書きます。

11.2 先ほど入力した関数定義は、`star.py` という名前のファイルに格納されていますので、

```
from star import *
```

という文をインタラクティブシェルに入力することによって、それを実行することができます。

11.3 `star` という関数が定義されたはずですので、それを呼び出してみましょう。

Lesson 12 真偽値

12.1 何かが成り立っているとき、それは「真」であると言います。逆に、何かが成り

立っていないとき、それは「偽」であると言います。

12.2 真であるか偽であるかということであらわすデータのことを「真偽値」と呼びます。

12.3 Python では、真はTrue、偽はFalse と書きあらわされます。

12.4 データとデータとの間に何らかの関係が成り立っているかどうかを調べたいときに使われる演算子は、「関係演算子」と呼ばれます。

12.5 関係演算子を使って関係が成り立っているかどうかを調べた結果は、真偽値で得られます。

12.6 == という関係演算子は、二つのデータのあいだに「それらは等しい」という関係が成り立っているかどうかを調べます。

12.7 < という関係演算子は、二つのデータのあいだに「左のデータのほうが右のデータよりも小さい」という関係が成り立っているかどうかを調べます。

12.8 関係演算子としては、そのほかに、!= (等しくない)、> (より大きい)、<= (より小さいかまたは等しい)、>= (より大きいまたは等しい) というものがあります。

Lesson 13 選択

13.1 どんなプログラミング言語にも、真偽値によって動作を選択する、ということを実行する方法があります。

13.2 Python では、「if 文」と呼ばれる文を書くことによって、選択を実行することができます。

13.3 if 文は、

```
if (式):
    文1
else:
    文2
```

と書きます。if 文を実行すると、その中に書かれた式の値が真ならば文1が実行されて、偽ならば文2が実行されます。

13.4 それでは、

```
gedit evenodd.py &
```

というコマンドで gedit を起動して、次の関数定義を入力してください。

```
def evenodd(a):
    if (a%2 == 0):
        return 'even'
    else:
        return 'odd'
```

13.5 この関数定義は、引数として受け取った整数が偶数ならば even という文字列を戻り値として返して、奇数ならば odd という文字列を戻り値として返す、evenodd という関数を定義します。

13.6 入力ができたら、それを保存して、

```
from evenodd import *
```

とインタラクティブシェルに入力することによって、それを実行してください。

13.7 evenodd が正しく動作するか、確かめてみましょう。

Lesson 14 出力

14.1 インタラクティブシェルは、入力された式の値が `None` というデータだった場合、それを出力しません。インタラクティブシェルに `None` と入力して、確かめてみましょう。

14.2 `print` という関数は、引数として受け取ったデータを出力します。この関数は戻り値として `None` を返しますので、インタラクティブシェルはこの関数の戻り値を出力しません。

14.3 インタラクティブシェルを使って、`print` にさまざまなデータを出力させてみましょう。

14.4 `print` は `None` を特別扱いしませんので、`print` に引数として `None` を渡すと、それが出力されます。

Lesson 15 繰り返し

15.1 「for 文」と呼ばれる文を書くことによって、動作の繰り返しを記述することができます。

15.2 動作を実行するごとに、一定の刻み幅で整数が変化していくような繰り返しをしたいときは、

```
for 名前 in range(整数):
    文
```

という形の for 文を書きます。そうすると、この中の「名前」のところに書かれた名前によって参照される整数が、0、1、2、3、4、.....と変化しながら、「文」のところに書かれた文の実行が繰り返されます。そして、「整数」のところに書かれた整数に到達したとき、繰り返しは終了します。文が実行されるのは、指定された整数に到達する直前までですので、名前がその整数を参照しているときに文が実行されるということはありません。

15.3 たとえば、

```
for i in range(10):
    print(i)
```

という for 文をインタラクティブシェルに入力すると、0 から 9 までの整数が出力されます。

Lesson 16 開始する整数の指定

16.1 0 からではなくて、指定された整数から変化を開始させたいときは、

```
range(開始, 終了)
```

と書きます。

16.2 たとえば、

```
for i in range(10, 20):
```

```
print(i)
```

というfor文をインタラクティブシェルに入力すると、10から19までの整数が出力されます。

16.3 それでは、

```
gedit triangle.py &
```

というコマンドでgeditを起動して、次の関数定義を入力してください。

```
def triangle(a):
    for i in range(1, a+1):
        print('*' * i)
```

16.4 この関数定義は、引数として受け取った整数が底辺と高さであるような三角形になるようにアスタリスクを出力する、triangleという関数を定義します。

16.5 入力ができたら、それを保存して、

```
from triangle import *
```

とインタラクティブシェルに入力することによって、それを実行してください。

16.6 triangleが正しく動作するか、確かめてみましょう。

Lesson 17 刻み幅の指定

17.1 1という刻み幅で増加させるのではなくて、指定された整数を刻み幅として増加させたいときは、

```
range(開始, 終了, 刻み幅)
```

と書きます。

17.2 たとえば、

```
for i in range(10, 200, 20):
    print(i)
```

というfor文をインタラクティブシェルに入力すると、20という刻み幅で、10から200までの整数が出力されます。

17.3 それでは、

```
gedit reverse.py &
```

というコマンドでgeditを起動して、次の関数定義を入力してください。

```
def reverse(a):
    for i in range(a, 0, -1):
        print('*' * i)
```

17.4 この関数定義は、引数として受け取った整数が底辺と高さであるような逆三角形になるようにアスタリスクを出力する、reverseという関数を定義します。

17.5 入力ができたら、それを保存して、

```
from reverse import *
```

とインタラクティブシェルに入力することによって、それを実行してください。

17.6 reverseが正しく動作するか、確かめてみましょう。