第2章

単純な計算プログラム

インタープリタの基本的な仕組みは、ある意味ではとても単純です。この章では、きわめて単純なプログラムから出発して、作成するのも理解するのもとても容易な計算プログラムを作成します。

2.1 最初のインタープリタ

インタープリタ（interpreter）とは、解釈（interpret）して何かをするプログラムです。言い換えると、インタープリタは、ユーザーからの入力を受け取って、その意味を解釈して適切な動作を行う、ということを繰り返すプログラムです。

◆ 単純なループ

最初に、このきわめて単純な作業を行うものを、具体的なプログラムとして作成してみましょう。このプログラムは、ユーザーの入力をキーボードから受け取って、受け取った文字列が「exit」または「quit」であればプログラムを終了し、そうでなければ受け取った文字列をそのまま出力するということを繰り返すプログラムです。

プロンプトを表示して 1 行のテキストを入力できるようにするには input() を使います。

```
line = input('->')  # プロンプトを出力して行を受け取る
```

表示される「->」はユーザー（インタープリタの場合はプログラマーがインタープリタのユーザー）に提示されるプロンプトです。

これを繰り返すためには、たとえば while ループを使います。

```
while True:
  line = input('->')  # プロンプトを出力して行を受け取る

  print(line)  # そのまま出力する
```

ただし、このままだと永遠にループを繰り返してプログラムが終了しないので、入力された文字列が「quit」か「exit」であれば break でループを抜けます。

```
  # 文字列が「quit」か「exit」であればループを抜ける。
  if line == "quit" or line == "exit" :
    break
```

ユーザーが「　exit」や「quit　」のように空白を入れた場合に対処するために、strip() で前後の空白を削除します。

```
  # 文字列前後の空白文字を削除する。
  line = line.strip()
```

このプログラムの名前は simpleloop にします。
プログラム全体は次のようになります。

リスト 2.1 ● simpleloop プログラム

```
# simpleloop.py

# ユーザーの入力を受け取って出力するループ
while True:
  line = input('->')  # プロンプトを出力して行を受け取る

  # 文字列前後の空白文字を削除する。
  line = line.strip()
  # 文字列が「quit」か「exit」であればループを抜ける。
  if line == "quit" or line == "exit" :
    break

  print(line)  # そのまま出力する
```

このプログラムはきわめて自明なので、コメントを見れば、何が行われるのか理解できるでしょう。「->」はユーザーに入力を促すプロンプトであり、プログラムは受け取った文字列が「exit」あるいは「quit」でなければ、文字列をそのまま出力します。

プログラムの実行例は、たとえば次のようになります。

```
InterpretPy\ch02>python simpleloop.py
->123
123
->abc 987
abc 987
->Hello, ++
Hello, ++
->quit
```

これはとても単純なプログラムですが、「exit」や「quit」という文字列を解釈して、そのほかの文字列の場合とは異なった動作をするという点で、立派なインタープリタです。また、「exit」と「quit」という文字列は、このインタープリタの「コマンド」であるという点も認識しておく必要があります。

インタープリタには、直接入力して実行するコマンドを定義することがあります。このインタープリタの「exit」と「quit」はインタープリタを終了するコマンドであり、本書の以降のプログラム全体を通してインタープリタを終了するためのコマンドとして「exit」と「quit」を使います。

このプログラムコードの意味をまったく理解できないか、ほかのプログラミング言語の知識を総動員してもこのプログラムコードの意味をまるで想像できないような場合には、Python の入門書で Python の基礎を学習してから先に進むことをお勧めします。

◆メッセージ

Python のインタラクティブシェルでは、ユーザーが「exit()」または「quit()」を入力するとインタラクティブシェルの実行が終了します。

「exit()」または「quit()」の最後の () を忘れて入力すると、次のようにメッセージが表示されます。

```
>>> exit
Use exit() or Ctrl-Z plus Return to exit
```

これは「終了するには exit() か、キーボードで [Ctrl]+[Z] を入力してください」という意味です。

また、「exit()」や「quit()」ではなくてもそれらに近い文字列を入力すると、次のようにメッセージが表示されます。

```
>>> exiy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'exiy' is not defined. Did you mean: 'exit'?
>>> exsit
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'exsit' is not defined. Did you mean: 'exit'?
```

最後の「NameError: name ' ○○ ' is not defined. Did you mean: 'exit'?」は、○○という名前は定義されていないが、「exit」と入力したいのか？というメッセージです。

このプログラムでは、「exit」と間違えてたとえば「Exit」や「EXIT」のような大文字を含むコマンドが入力されたときには、次のようなメッセージを表示することにしましょう。

```
->Exit
もしかして'exit'ではないんかい？
Exit
->
```

必要なことは次のコードを追加することだけです。

```
#「quit」か「exit」で大文字が含まれていればメッセージを出力する。
if line.lower() == "exit" :
  print("もしかして'exit'ではないんかい？")
if line.lower() == "quit":
  print("もしかして'quit'ではないんかい？")
```

プログラム全体は次のようになります。

リスト 2.2 ● simpleloop_a プログラム

```
# simpleloop_a.py

# ユーザーの入力を受け取って出力するループ
while True:
  line = input('->')  # プロンプトを出力して行を受け取る

  # 文字列前後の空白文字を削除する。
  line = line.strip()
  # 文字列が「quit」か「exit」であればループを抜ける。
  if line == "quit" or line == "exit" :
    break
  # 終了しないのに文字列が大文字が含まれている
  #「quit」か「exit」で大文字が含まれていればメッセージを出力する。
  if line.lower() == "exit" :
    print("もしかして'exit'ではないんかい？")
```

```
  if line.lower() == "quit":
    print("もしかして'quit'ではないんかい？")

  print(line)  # そのまま出力する
```

◆ バージョン情報

OS のコマンドプロンプト（> や %、$ などを含む文字列）に対して入力するコマンド文字列全体をコマンドラインといいます。

Python のインタラクティブシェルを起動するときに、コマンドラインオプションとして「-V」または「--version」を付けて実行すると、次のように Python のバージョンが表示されます。

```
>python -V
Python 3.11.3

>python --version
Python 3.11.3
```

simpleloop もコマンドラインオプションとして「-V」または「--version」を付けて実行すると、バージョンが表示されるようにしましょう。

コマンドラインオプションをプログラムが受け取ることができるようにするには、次のように Python を起動するときに、Python のスクリプトファイル名（この例では pyarg.py）に加えて、たとえば次のように「abc 123」という引数をスクリプトファイル名の後に指定して python を起動します。

```
> python pyarg.py abc 123
```

OS から見ると「python pyarg.py abc 123」全体がコマンドラインです。

Python のプログラム（スクリプト）から見ると、「pyarg.py abc 123」が引数を伴うコマンドラインです。

Python のプログラムは、sys モジュールの argv を使って受け取ることができます。

次の例は、スクリプトが実行されたときのコマンドラインの引数をすべて表示するためのスクリプトの例です。

リスト 2.3 ● pyarg.py

```
# pyarg.py
import sys

args = sys.argv

for v in args:
    print(v)
```

コマンドラインを「python ex12-1.py abc 123」として上のスクリプトを実行すると次のように出力されます。

```
InterpretPy\ch02>python ex12-1.py abc 123
ex12-1.py
abc
123
```

最初の引数（sys.arg[0]）はスクリプトファイル名（ex12-1.py）です。

2 番目の引数（sys.arg[1]）はこの場合「abc」で、3 番目の引数（sys.arg[2]）はこの場合「123」です。

ここでは、Python のスクリプトに対する引数が「-V」か「--version」であるときにバージョン情報を出力するというのが目標です。

そこで、次のようにします。

```
# 引数に「-V」か「--version」が含まれていれば
# バージョン情報を出力して終了する
args = sys.argv
for s in args[1:]:
  if s == '-V' or s == '--version':
```

```
    print('simpleloop_b バージョン 0.001')
    sys.exit()   # このプログラムを終了する
```

args[0] にはスクリプト名が入っているので、2 番目の要素から最後の要素まで加算するために、for 文の範囲を「args[1:]」としている点に注意してください。

プログラム全体は次のようになります。

リスト 2.4 ● simpleloop_b プログラム

```
# simpleloop_b.py
import sys

# 引数に「-V」か「--version」が含まれていれば
# バージョン情報を出力して終了する
args = sys.argv
for s in args[1:]:
  if s == '-V' or s == '--version':
    print('simpleloop_b バージョン 0.001')
    sys.exit()   # このプログラムを終了する

# ユーザーの入力を受け取って出力するループ
while True:
  line = input('->')  # プロンプトを出力して行を受け取る

  # 文字列前後の空白文字を削除する。
  line = line.strip()
  # 文字列が「quit」か「exit」であればループを抜ける。
  if line == "quit" or line == "exit" :
    break
  # 終了しないのに文字列が大文字が含まれている
  #「quit」か「exit」で大文字が含まれていればメッセージを出力する。
  if line.lower() == "exit" :
    print("もしかして'exit'ではないんかい？")
  if line.lower() == "quit":
    print("もしかして'quit'ではないんかい？")

  print(line)  # そのまま出力する
```

このプログラムの実行例を次に示します。

```
InterpretPy\ch02>python simpleloop_b.py -V
simpleloop_b バージョン 0.001

InterpretPy\ch02>python simpleloop_b.py --version
simpleloop_b バージョン 0.001
```

これで、「exit」と「quit」というコマンドを解釈し、バージョン情報も出力する単純なインタープリタができました。

2.2 eval マシン

Python には文字列の式をプログラムコードの式として評価することができる関数 eval() があります。

これを使うと計算式を実行するプログラムを容易に作成できます。

◆ Python の eval()

Python の eval() は、引数に指定した文字列形式の式をプログラムコードの式として評価します。

たとえば、line という名前の変数に式を文字列として保存し、eval() で式を評価します（実行します）。

```
line = '2 + 4 * 3'
eval(line)
```

上のコードを Python のインタラクティブシェルで実行すると、次のように実行することができます。