2015年10月27日

# プログラミング入門
# 関数

横浜国立大学

倉光君郎

# 関数

関数は、ある変数に依存して決まる値の対応を表す式である。つまり、変数$x, y$があり、$x$を入力として出力$y$を決定する規則があるとき、変数$y$は$x$の関数と呼ぶ。関数(function)の頭文字からとって、次のように書くことが多い。

$$y = f(x)$$

今回は、このような関数$f(x)$をプログラミング言語で定義し、利用する方法を学ぶ。

# 引数と返り値

まず、簡単な例として、整数 x に 1 を加算するという規則をもった関数を考えよう。これは次のように定義できる。

$$f(x) = x + 1$$

ここで、関数 $f(x)$ の $x$ のような関数に値を渡す変数を引数、もしくはパラメータと呼ぶ。また、関数 $f(x)$ を評価したときの結果の値を返り値 (return value) と呼ぶ。

# $f(x)$の型はどう型付けられる？

まず最初に、プログラミング言語では「型付け」が重要だと思い出して欲しい。当然、関数にも型付けがある。

$$f(x) = x + 1$$

今、引数 $x$ は整数なので、int 型としよう。
$x + 1$ はint型となるので、$f(x)$ の返り値は int 型となる。ただし、これは $f(x)$ を評価したときの型であり、$f(x)$ 自体の型とはいえない。

# 関数の型付け

$f(x)$ **の型は「**int **型の入力をうけて** int **型を出力する」という型になる。一般的に、次のような写像で書かれることが多い。**

$$f : \texttt{int} \mapsto \texttt{int}$$

# 関数定義の記法

関数定義の表記は、プログラミング言語ごとには様々な表記法がある。大きな流派は、変数とおなじく、静的言語か動的言語かで大きく分類される。

Konoha は、どちらの言語も体験できるように作られている。

# 型付けありの場合

静的言語では、関数定義には型付けが必要になる。次は、C言語スタイルの関数定義である。

```
int f(int x) {
  return x + 1;
}
```

型付けは、引数 x と関数 f(x) の返り値の型をそれぞれ前で宣言する。それに続く中括弧{ }は関数の本体であり、定義された関数が評価すべき式

を書く。

# 型付けなしの場合

**動的言語では、型付けを書く必要がない。次は、**JavaScript **スタイルの関数定義である。**

```
function f(x) {
  return x + 1;
}
```

**関数定義が始まることを示す** `function` **に続いて、中括弧** `{ }`**の中に評価すべき式を書いている。**

# return文

**関数定義に共通する特徴は、`return` 文と呼ばれるステートメントが登場する点だろう。**

```
return x + 1;
```

**ステートメントに関しては「制御構造」で詳しく述べるが、`return` 文には関数の返り値を評価する式を書く用意された構造である。行末の`;`はステートメントの終端を表す記号である。英文の終端を表すピリオッドに相当する。**

# 関数適用

定義した関数は，式として利用可能になる。関数のパラメータに値を渡して評価することを、関数を呼び出す（コールする）、もしくは適用するという。例えば、関数f(x) のパラメータに0を渡して評価するには：

```
f(0)
```

# 評価順序

**関数のパラメータには、値だけでなく、式を与えることができる。**

```
f(1+1)
```

**このとき、評価する順番が重要となる。**

$$f(1+1) \mapsto f(2) \mapsto 3$$

$$f(1+1) \mapsto (1+1)+1 \mapsto 3$$

数学的では、評価する順序は関係ない。どちらから評価しても同じ結果がえられる。プログラムは、状態をもつため、評価する順序によって結果が異なる場合がある。これを副作用と呼ぶこともある。

# 先行評価と遅延評価

**プログラミング言語では、パラメータの式を先に評価したのち、関数を適用する。**

$$f(1+1) \mapsto f(2) \mapsto 3$$

**逆に、必要になるまで評価しない方式を遅延評価と呼ぶ。**

$$f(1+1) \mapsto (1+1)+1 \mapsto 3$$

# 演習問題

2次関数 $f(x)$ を定義し，f(0.5) の値を求めてみよう。

$$f(x) = x^2 + 2x + 1$$

# 関数と名前

まず、簡単な関数定義は理解できるようになったと思う。あとは、いくつか関数を書いてみて少し慣れることが重要だ。
その前に、これから関数を定義する上で注意すべきことをまとめておきたい。

- 関数には適切な名前をつける
- 複数のパラメータを利用できる
- 関数を使う前に定義する

- **車輪を再発明しない**

# 適切な名前

数学では、関数の名前はそれほど意味をもたない。しかし、プログラミング言語では、非常に多くの関数を扱うことになるので、関数の名前はその関数の機能や役割を判断する重要な役割を担う。前節の例では、便宜上、関数 $f(x)$ としたが、適切な名前をつけるべきである。
だから、もし奇数(odd)かどうか判定する関数を定義するなら、単に関数名 $f$ とする：

```
boolean f(int x) {
  return x % 2 == 1;
}
```

**よりも、`isodd`のような名前のほうがよい。**

```
boolean isodd(int x) {
  return x % 2 == 1;
}
```

**関数名には、英字もしくは数字、_(アンダースコア)が使える。言語によっては、漢字、ひらがな・**

カタカナが使える言語もあるが、和英辞典でちゃんとしらべて、英語で関数名をつけるのがよい。

# 複数のパラメータ

関数定義では、複数のパラメータを用いることができる。パラメータには、それぞれ異なる変数名をつけ、それぞれ型付けする。
次は、条件演算子を用いて x と y の大きな値を得る関数である。

```
int max(int x, int y) {
  return x > y ? x : y;
}
```

もちろん、2パラメータ以上の関数も定義できる。次のmax(x,y,z) は、$x, y, z$のうち、最も大きな値をえる関数となる。

```
int max(int x, int y, int z) {
  return max(x, max(y, z));
}
```

max(x,y,z)では、関数max(x,y)をうまく利用して定義している。このように、既に定義された関数は別の関数から利用することができる。逆に、定義されていない関数を呼び出そうとすると、エ

ラーになる。

# 多重定義

関数名のルールでもうひとつ大切なルールを覚えておこう。原則、関数定義では同じ関数名を使うことはできない。max(x,y) と max(x,y,z) は同じ関数名である。
唯一例外が認められる場合がある。それは、関数名は同じでも、パラメータの数やその型付けが異なる場合である。このような定義を多重定義と呼ぶ。
残念ながら、多重定義をサポートしていない言語

**では、例えば** max3(x,y,z) **のように名前を変えて定義しなければならない。**

# 車輪の再発明

車輪を再発明するとは、「広く受け入れられ確立されている技術や解決法を知らずに（または意図的に無視して）、同様のものを再び一から作る」ことである。
あなたが定義しようとする関数は、必ず誰かが過去に定義している。だから、定義しようと考える前に、定義していないか調べることが重要である。

# ライブラリ

ライブラリとは、定義済の関数などを再利用しやすいようにまとめたものである。インポートして利用する。

Konoha は、Java 言語のライブラリを利用できる。例えば、java.lang.Math を利用したいときは、次のように import 文を用いる。

```
import java.lang.Math;
```

# ライブラリ: Math

Math **ライブラリは、基本的な数値計算の関数を集めたライブラリである。三角関数など便利な関数が含まれている。**

```
import java.lang.Math;
>>>
sin(0.5)
```

# 演習問題2

Math ライブラリの中から、整数 n の絶対値を求める関数名をしらべてみよう。

ヒント: java.lang.Math で検索する

# 演習問題2-2

1から6 の値をランダムに出力するサイコロ関数 (dice()) を定義してみよう。

ヒント： Math ライブラリの関数をよく調べる

# 再帰関数

**整数の $n$ の階乗、つまり $n!$ を求める関数を考えてみよう。**

$$1! = 1$$

$$2! = 2 \times 1$$

$$3! = 3 \times 2 \times 1$$

$$4! = 4 \times 3 \times 2 \times 1$$

# 再帰関数

$$n! = \begin{cases} 1 : (n = 1) \\ n \times (n-1)! : (otherwise) \end{cases} \tag{1}$$

**プログラミング言語で関数定義してみると：**

```
int fact(int n) {
  return n == 1
      ? 1
      : n * fact(n - 1);
}
```

**定義された関数において、自分自身を呼び出すことを再帰呼び出しと呼ぶ。再帰呼び出しを含んだ関数を再帰(recursion)関数と呼ぶ。**
**再帰は、繰り返し処理を記述する重要な構造である。**

# 演習問題3

**フィボナッチ数を求める関数を定義せよ。**

$$
\begin{aligned}
F_1 &= 1 \\
F_2 &= 1 \\
F_n &= F_{n-1} + F_{n-2}
\end{aligned}
\tag{2}
$$