2. . . の夢見鳥

2.1 指数関数

数学に**関数**は付きものである。関数は特別難しいものではない。記述の仕方に慣れさえすれば、関数はむしろ便利で使いやすいのである。

関数 $f(x) = x^2$ があるとする。これは、f という名の関数に (何がし)^2 の機能を持たせている。そのため関数 f に 5 を与えると、25 という値が返ってくるのだ。また関数 $g(x) = 2^x$ は、g という名の関数に $2^{(\text{何がし})}$ の機能を持たせている。よって関数 g に 10 を与えると、1024 という値が返ってくるのだ。

ここでちょっと変な感じを持った人がいるだろうか　私が関数 $f,\ g$ と言ったことに対してだ。だが、これでよい。$f(x)$ と書くのは、関数 f が変数 x を使うことを明らかにするためなのだ。もし皆が皆、変数には x しか使わないと取り決めていたら、$f = x^2,\ g = 2^x$ で十分である。しかし、変数には x 以外の文字を使うことはよくあるし、変数が 2 個以上ある関数だってある。さらには文字定数を利用する場合があることを思えば、関数名に続けて ( ) を付け加えることが不可欠なのである。プログラミングの世界では ( ) 内で与える変数を**引数（ひきすう）**と呼ぶ。

関数の例をあと二つ提示しよう。

一つ目は $f(x,\ y) = x^2 - 3y$ でどうだろう。関数 f は変数に $x,\ y$ を使う。その結果返ってくるのが $x^2 - 3y$ で計算された値だ。だから $f(5,\ 10)$ と書けば、$x = 5,\ y = 10$ を代入して計算した値 -5 が返ってくることになる。一般に数学で使う関数は、実数値を受け取って実数値を返すので、$f(0.8,\ 0.1)$ でも計算可能で 0.34 が返ってくる。

二つ目の関数は $g(x) = \lfloor x \rfloor$ でどうだろう。見慣れない記号 $\lfloor\ \rfloor$ はフロア記号と呼ばれる。フロア記号を簡単に説明すると、与えられた値を超えない整数のうち、最大のものを返す機能を持っている。むむ、難しい表現だ。具体的には $g(3.14)$ と書けば 3 が返され、$g(-3.14)$ と書けば -4 が返されるのだ。いわゆる**切り捨て**である。フロア（floor）の名称から想像できるように、これは建物の床を意味する。3.14 階という表現はおかしいが、3 階の部屋の中空に 3.14 階があると思えば、その床は 3 階だから $\lfloor 3.14 \rfloor = 3$ とみなすのである。

フロアと対（つい）になる関数はシーリング（ceiling）と呼ばれ、$\lceil x \rceil$ で表される。$\lceil x \rceil$ は、x を下回らない最小の整数である。具体的には $\lceil 3.14 \rceil = 4$、$\lceil -3.14 \rceil = -3$ などである。

いま二つの関数を例に出してみたが、値を求めるためにする記述は、いずれも $f(5,\ 10)$ や $g(3.14)$ のように簡便だ。にもかかわらず -5 や 3 の値が返ってくるのは、裏方で f や g がせっせと $5^2 - 3 \times 10$ や $\lfloor 3.14 \rfloor$ の計算をしているからである。また、$f(3,\ 8)$ のように別の値を与えれば、それに応じた値が返ってくる。これは、関数がする仕事がきちんと決められているからである。

何のことはない。関数とは、与えられた値を別の値に加工する手続きなのだ。ということは、これまでに翔び回ってきた夢見鳥路でも、TEX の引数をとる命令（関数）を目にしてきたはずだ。

この路では**指数関数**を話題に取り上げる。はじめに、もっとも基本的な $f(x) = 2^x$ にしておこう。x の値を与えると 2^x の値を出力するスクリプトである。まず関数を『

```
\begin{luacode*}
function pow2(x)
    return 2^x
end
\end{luacode*}
```

』で定義してみよう。Lua で関数を定義する場合は、`luacode*`環境か `\directlua{}` 内で

`function` 関数名 (引数)　　関数の処理　　`end`

のように書く。引数には数値や文字列だけでなく関数までも与えることができる。関数名は英数字を用い、先頭が数字でなければよい。その上で別の `luacode*`環境か `\directlua{}` で、たとえば『$2^8 =$ `\directlua{tex.print(pow2(8))}`』と書けば『$2^8 = 256.0$』が出力される。整数だけの計算なのに小数点がつくのはかっこ悪くないか？　そう思ったら『

```
\begin{luacode*}
function ipow2(p)
    n = 1
    while p > 0 do
        n = n * 2
        p = p - 1
    end
    return n
end
\end{luacode*}
```

』という関数を定義して『2^8 = `\directlua{tex.print(ipow2(8))}`』で処理してみう。今度は『$2^8 = 256$』と出力されるだろう。

普通 2^p というのは 2 を p 回繰り返し掛けるので、繰り返しのために `while` 構文を使った。引数が掛ける回数であるのはよいとしても、掛け始めは何だろう。2^p だから 2 が掛け始めと考えてもよいけれど、ここでは土台に 1 を選んでみた。そうすると掛け始めは 0 回目と数えることになる。で、ここからがスクリプトの肝（きも）なのだが、掛けた回数を加算するのではなく引いていることに注目してほしい。それが `p = p - 1` だ。こうすれば `p = 0` になったところで必要回数だけ 2 を掛けたことになり `while` 構文を抜けられる。フィボナッチ数列でも `while` 構文は使ったが、このような使い方が本来の使い方なのだろう。

関数のよいところは使い回しが効くことである。一度どこかで関数を定義しておけば、あとは関数名を記述すれば望みの出力が得られる。普通、関数はなんらかの処理をして結果を返すのが役目だから、`return` 文を用いて結果を返す仕組みになっている。しかし、結果は組版のときには `tex.print` 文で出力されるのだから、そのたびに `tex.print()` を書くのはわずらわしい。もし組版の出力に使うことが確実なら、関数は『

```
\begin{luacode*}
function ppow2(p)
    n = 1
    while p > 0 do
        n = n * 2
        p = p - 1
    end
    tex.print(n)
end
\end{luacode*}
```

』と定義して、『$2^8 =$ `\directlua{ppow2(8)}`』と書いて『$2^8 = 256$』を出力させてもよいだろう。そうなればもちろん値出力専用の関数になって、得た値を別のところや別の関数へ引き渡せないことになる。うまく使い分けるようにしたい。

さらに、TEX で組版しているのだから TEX のマクロにして、たとえば

```
\newcommand{\powtwo}[1]{\directlua{ppow2(#1)}}
```

のように定義しておいて、『$2^8 =$ `\powtwo8`』と書けば『$2^8 = 256$』を出力するようにもできる。

どこまで自動化するかは難しい問題だが、関数はできる限り汎用的なものにするほうが使い勝手がよい。そのために多少タイピングする文字数が増えても、役に立つ関数であるべきかもしれない。この路で見かける関数は役立たずだけどね。