1. . . の夢見鳥

1.1 フィボナッチ数列

数列とは、数が何らかの規則で並んでいるものをいう。『2, 4, 6, 8, 10, 12, . . . 』は偶数の数列だが、偶数列を作る規則は「2 から始めて前の**項**に 2 を加える」というものである。この偶数列は『

```
\begin{luacode*}
    for n = 1, 6 do tex.print(n*2 .. ', ') end
    tex.print('$\\dots$')
\end{luacode*}
```

』と書いただけだ。

さて、ここに 1, 1 から始まる数列がある。次の項を作る規則は「直前の 2 項の和」である。すなわち 1, 1 の次に来る項は 2 となり、数列は 1, 1, 2 と延びる。規則を繰り返し当てはめれば、次の項は 3 で数列は 1, 1, 2, 3 と延び、さらに次の項は 5 である。これが延々となされ『1, 1, 2, 3, 5, 8, 13, 21, 34, 55, . . .』なる数列が出来上がる。このようにしてできる数列を**フィボナッチ**[1)]**数列**と呼ぶ。

フィボナッチ数列を出力した Lua のコードは『

```
\begin{luacode*}
    a = 0  b = 1
    for i = 1, 10 do
        tex.print(b .. ', ')
        a, b = b, a + b
    end
    tex.print('$\\dots$')
\end{luacode*}
```

』である。ちょっと変な書き方かもしれない。

フィボナッチ数列は 1, 1 から始まっているのだから、初期値は `a = 1  b = 1` にして `tex.print(a .. ', ' .. b .. ', ')` から始めるのが自然なのだろうが、この `tex.print` 文が邪魔に思えたのだ。そこで、それを削るためわざわざ `a = 0  b = 1` で初期化して、`for` 文によって 1, 1 から始まるように仕向けたのだ。どうでもいいことだよね。

1) フィボナッチ（1174?–1250?）: イタリアの数学者。本名、ピサのレオナルドの通称。

フィボナッチ数列で出力したいのは

「$\ldots,\quad a,\quad b,\quad a+b,\quad \ldots,\quad \ldots$ 」の $a+b$ である。しかし次には、

「$\ldots,\quad \ldots,\quad a,\quad b,\quad a+b,\quad \ldots$ 」に変わっていくので、ここの

b を出力しても同じことである。このためスクリプトは `a = 0  b = 1` にし、`b` を出力したあと $b \to a,\ a+b \to b$ の操作を行ったのだ。この操作は Lua の場合 `a, b = b, a + b` と書ける。`a = b` と `b = a + b` の代入を同時にする記述法だ。

注意してほしいのは `a = b` と `b = a + b` を分けて書いてはいけないことである。なぜなら分けて書くと、まず `a` の値が `b` に変わるので、次の `b = a + b` が `b = b + b` になってしまうからである。分けるなら臨時変数 `t` を用いて

```
t = b  b = a + b  a = t
```

の手順を踏んで代入しよう。簡潔さでは劣るが、覚えておいて損はないと思う。

さて、このスクリプトがあればフィボナッチ数列は好きなだけ出力できるけど、そんな数の羅列を喜ぶ人はいないだろう。でも、数の羅列以上に困ったことが起きる。ここでは実際にやらないけど、たとえば繰り返し回数を `for i = 1, 100` とでもしてみるとよい。出力はだいぶ変でしょ。

最初に目につくのは数が版面からはみ出していることだろう。`for i = 1, 70` 程度までならうまく出力するので、大きすぎる桁の数が続くと流石（さすが）の TEX でも適切に組版できないようだ。

次に、後半に負の数が出ること。これについては前に少し触れたし、あとで話す機会があるから待ってもらおう。ついでに負の符号が数式モードでないが、`text.print` において `'$'` を連結すれば解消できることも前に話したね。

ところで、フィボナッチ数列の全部ではなく、ある項の値だけを知りたいときはどうしよう。それは `tex.print` 文を `for` ループの外へ出せばよい。でも『10 番目のフィボナッチ数は

```
\begin{luacode*}
    a = 0  b = 1
    for i = 1, 10 do
        a, b = b, a + b
    end
    tex.print(b)
\end{luacode*}
```

かな？』としただけでは『10 番目のフィボナッチ数は 89 かな？』となって、11 番目のフィボナッチ数が出力される。

解決方法は簡単なことで、`for i = 1, 10` を `for i = 2, 10` とするだけでよい。そもそも変数 `i` は単にカウンタとしての意味しかなかったので、最初の `b = 1` が第 1 項であることから、あと 9 回ループを回せばそのときの `b` が 10 番目となるリクツだ。実際 `for i = 2, 10` に変えて処理すると『10 番目のフィボナッチ数は 55 かな？』となる。`for i = 2, 90` に変えれば『90 番目のフィボナッチ数は 2880067194370816120 かな？』とできる。で、これは正しい。プログラミングはこのように短いものでもちょっとした工夫が入る余地がある。逆に、ちょっとしたことで期待した動作にならないこともある。勉強としては役に立ったかもしれない。

だけど、いま試した方法はいただけない。`for` 文の数値をいじってプログラミングの動作を制御するってのはどうなの？　もし 90 番目のフィボナッチ数が知りたければ、たとえば

```
tex.print(fibonacci(90))
```

のようにして出力されるべきだろう。もう少しここらをふらついたら、その場所へ行こう。