7.3 書式を整えよう

ああ、散歩を長くすると景勝地をたくさん見ることができるけれど、徒歩で行動しているので疲れただろう。相棒の **Python3** は本当に手軽ないいやつである。わずかのコードを書くだけで、思い通りの結果を出してくれる。それでも、完璧な出力を目指すには知らなくてはいけない機能が多い。ちょっと一休みして、書式について考えてみたい。

途中、円周率を 100 桁ほど計算させたけど、計算は長整数に変えて行ったので、小数点の位置は自分で把握する必要があった。できれば

3. 14159 26535 89793 23846 26433 83279 50288 41971 69399 37510
   58209 74944 59230 78164 06286 20899 86280 34825 34211 70679
   82148 08651 32823 06647 09384 46095 50582 23172 53594 ...

みたいに、かっこ良く表示したいものだ。

[py script]

```
>>> def fxl(f, n, u, v):
...   return (4 * 10**f // (n * u) - 10**f // (n * v))
...
>>> def disp(s):
...   print(s[:1] + '.')
...   i = 1
...   while s[i:i+5]:
...     print(s[i:i+5], end=' ')
...     i += 5
...
>>> def mpil(f):
...   p = 0; sgn = 1
...   u = 5; v = 239
...   n = int(f * math.log(10) / math.log(5))
...   for n in range(1, n, 2):
...     p += sgn * fxl(f, n, u, v)
...     sgn = -sgn
...     u *= (5 * 5)
...     v *= (239 * 239)
...   disp(str(4 * p))
...
>>> mpil(100)  # 実行前に import math
3.
14159 26535 89793 23846 26433 83279 50288 41971 69399 37510 58209 74944
59230 78164 06286 20899 86280 34825 34211 70684
```

スクリプトは以前の `mpil` 関数をほぼそのまま使ったので、決して洗練されたスクリプトになっていない。スクリプトに機能を追加するときは、部分的に変えるほうがよいのか、いっそ全部書き

換えるほうがよいのか迷うものだ。たいてい、部分的に変えるほうが複雑化しがちである。ここでは部分的な変更で済ませたので、スクリプトが煩雑になるという見本のようなものだ。

`fxl` 関数が妙にたくさんの引数をとっていること、新たに `disp` 関数を追加したこと、それと `mpil` 関数に与える引数が違うことが変更点だ。ついに関数がみっつになったぞ。

最初に `mpil` 関数を見よう。以前 `mpil(n)` であったものを `mpil(f)` に変えている。引数の文字を変えても機能に差が出るわけではない。引数名を変えたのは数値の意味が違うからである。以前の引数は計算回数を与えるものであった。それは、関数が 100 桁分の計算に限られていて、計算回数を探る意味もあったからである。しかし、π の計算をするのに、そういう引数の与え方をするのは不自然である。`mpil(300)` と入力したら、計算を 300 回するのではなく、300 桁の表示をしてもらいたい。今回の `mpil` 関数の引数 `f` は、純粋に求める π の桁数 f を意味している。

桁数を引数に与えるということは、計算回数は自ら求めなくてはならないということでもある。$\frac{1}{5^{2n-1}}$ のほうが $\frac{1}{239^{2n-1}}$ より収束が遅いので、たとえば f 桁の精度が欲しければ $\frac{1}{5^{2n-1}} < \frac{1}{10^f}$ を満たす $2n-1$ を計算すればよい。それは、$\frac{f\log(10)}{\log(5)}$ で求められる。

それを `fxl` 関数に渡して $\frac{4}{(2n-1)5^{2n-1}} - \frac{1}{(2n-1)239^{2n-1}}$ を計算するのだが、以前はそのたびに 5^{2n-1} などを繰り返しの掛け算で求めていたから効率が悪かった。それよりも順次 `5 * 5` の積を追加するほうが計算量が格段に減る。そこで、それらの計算は `fxl` 関数が受け持つのではなく、あらかじめ計算した値を渡すようにした。以上の理由から、`fxl` 関数の受け取る引数が多くなってしまったのだ。

ついでだから、変数の初期化について一言触れておこう。ここでも、関連の強い変数を 1 行にまとめて記述している。もし “;” で区切るのがうしろめたければ、`p, sgn = 0, 1` や `u, v = 5, 239` のようにタプル代入するとよいだろう。作法と行数を天秤にかければ、おそらく `p = 0` と `sgn = 1` は別行に、`u` と `v` はタプル代入、というのが適切かもしれない。

さて、計算された π の表示を受け持つのは `disp` 関数である。これは受け取った文字列を 5 文字ずつ区切って出力するものである。数値を区切って表示するのは至難の技であるから、`4 * p` の値は文字に変換して `disp` 関数に渡している。`disp` 関数が何をするかといえば、最初に小数点より上の値を出力することである。ここで、スライシングについて詳しく述べておこう。

`s[:1]` という書式が最初の `3` を表示し、`+`演算子で小数点を連結している。基本的に `s[x:y]` と書けば、文字列 `s` の `x` 番目の文字から `y` 番目のひとつ前の文字まで切り出すことになる。n 文字ある文字列の先頭が 0 番目、末尾が $n-1$ 番目と数えることに注意してもらいたい。ちなみに、`s[:y]` は `s[0:y]` を、`s[x:]` は `s[x:n]` を意味する。

スライシングの命令は `while` ブロックの中にあるので、`while` の条件、`s[i:i+5]` に文字列がなくなるまで切り出しが続く。`for` 構文を使ってもよいのだが、それだと繰り返しの回数も `disp` 関数に知らせなくてはならない。`while` 構文ならその必要がないのでそうしたのである。`while` 構文と `for` 構文はまったく同じ処理ができるけれど、適宜使い方を選択するとよいだろう。

さて、これで円周率を 1000 桁でも 10000 桁でも、見栄えよく表示できるようになった。ただし、アルゴリズムにはまだ問題が多い。たとえば、計算回数は $\frac{1}{5^{2n-1}}$ の精度で決めているが、ある時点からは $\frac{1}{239^{2n-1}}$ は 0 になっているのに計算してしまう。これは無駄が多い。こういったことをひとつひとつ吟味すれば、スクリプトは洗練されてくる。まあ、それは君たちの宿題としておこう。楽しんで改良してもらいたい。

余談になるが、5^2 と 239^2 を `5 * 5`、`239 * 239` と書いたことに注意してほしい。**Python3** で n^2 を計算する場合は、この他に `n**2` と `pow(n, 2)` がある。いずれも同じ結果を返すので、どれを用いてもかまわない。ただし計算時間の点では、おそらく `n * n` がいちばん有利で、`pow(n, 2)` がいちばん不利なはずじゃないかな～（自信のない言い方で申し訳ない）。計算回数が増えれば、実行時間が短くなるように工夫することは大事だ。塵も積もれば山となる、という格言はまさにこのためにある。