7...の散歩道

7.1　不思議な循環小数

$\frac{1}{7} = 0.142857\cdots$ は不思議な性質を持っている。正確には、循環節 142857 の性質が面白いのだ。それは

$$\begin{aligned} 142857 \times 2 &= 285714 \\ 142857 \times 3 &= 428571 \\ 142857 \times 4 &= 571428 \\ 142857 \times 5 &= 714285 \\ 142857 \times 6 &= 857142 \end{aligned}$$

となるからだ。なんと、循環節の数字が巡回しているではないか。

こうなってくると、他にも不思議な循環節を持つ分数を調べたくなるだろう。以前、循環節を調べたことを覚えているかい？　散歩を始めたばかりの頃は、$\frac{1}{113}$ がどの程度の循環節を持つか調べるのに苦労したね。余りをザーッと表示させて、最初の余りと同じものがあるかを調べたんだ。これは手間のかかる作業だし、第一、商がどうなっているのか分からずじまいだった。当時は、循環節を表示する力量が不足していたので、仕方なく余りを表示することで目的を達していたのだ。ところが、いまや力量はかなり上昇している。循環小数の循環節を見せてくれるスクリプトを書いてみよう。

対象とする分数は $\frac{1}{n}$ の形で十分だが、n の値によっては困ることがある。循環しないで割り切れてしまう場合があるからだ。割り切れてしまう分数は、必ず $\frac{1}{2^r 5^s}$ の形をしている。従って、n には 2 と 5 が含まれていないことが条件だ。ただ、この条件の分数だけを対象とすると、$\frac{1}{6}$ のような循環小数も除外されてしまう。しかし $\frac{1}{6}$ は、$\frac{1}{3}$ の循環小数を $\frac{1}{2}$ で分割していると見れば、本質は $\frac{1}{3}$ と同じだ。よって 2 や 5 がひとつでも含まれている分数は、対象から外して構わないだろう。そのほうがスクリプトも簡単にできる。

[py script]

```
>>> def cyc(n):
...   r = 10 % n; c = 1
...   while r != 1:
...     r = (r * 10) % n
...     c += 1
...   return c
...
>>> def cydec(n):
...   if (n % 2) and (n % 5):
...     r = 1
...     for i in range(1, cyc(n)+1):
...       print((r * 10) // n, end=' ')
...       r = (r * 10) % n
...     print('...')
...
>>> cydec(7)
1 4 2 8 5 7 ...
```

スクリプトは、$\frac{1}{7}$ なら “1 4 2 8 5 7 ...” と表示することにした。“...” 以下が循環するという意味である。もし、小数を表示していることを明確にしたければ、最初に `print('0.', end=' ')` を書いておけばよいだろう。

以前のスクリプトでは、$\frac{1}{n}$ の余りを $(n-1)$ 回表示させた。これだと $\frac{1}{9}$ のような分数では、最初の余りから 8 回目の余りまで 1 が表示されてしまう。今回は商の表示を目的としているので、$\frac{1}{9}$ は何も “1 1 1 1 1 1 1 1 ...” でなく “1 ...” となれば十分だ。それに、余分な循環節を表示するようでは、$\frac{1}{113}$ が本当に 112 の循環節を持つ小数かどうかは、綿密に調べなくてはならず効率が悪い。

そのために、循環節がいくつか調べる関数 `cyc(n)` が必要になったのだ。先に、そこから説明しよう。

`cyc` 関数は $\frac{1}{n}$ の分母を変数 `n` で受け取る。このとき最低でも 1 回の割り算が実行されるはずだから、それを `r = 10 % n` でしている。`r` は余りが代入される変数だが、この時点では分子の `1` を分母の `n` で割るわけなので、`0` を下ろす意味で `10 % n` となっているのだ。また、変数 `c` は循環の回数を記憶するために必要で、ここで 1 回目の割り算が行われたので `1` で初期化してある。

余り `r` が `1` になれば一回りしたことになるので、`while` 構文は `r` が `1` にならない間繰り返される。その際の処理は、以前のスクリプトで使った手法と同じである。違いは循環の回数をカウントしていることだ。`r = 1` となったら、`return` 文で循環の回数である値が返される。

ところで変数 `r` は、`cyc` 関数にも `cydec` 関数にも使われているけど大丈夫？　大丈夫である。それは `cyc` 関数と `cydec` 関数は独立しているからだ。そのため、同じ変数名を使っても問題はない。

これは、**スコープ**に関する大事なポイントである。

次に、cydec 関数を見よう。冒頭に述べたように、分母に 2 または 5 を**因数**に持つ数は除外したい。そこで、if 文によって入力を制限している。ここで使われている記号 and は以前も使った論理演算子である。ここで行われていることを詳しく述べたい。`n % 2` と `n % 5` は、割り切れないときは 0 以外の値になっている。前に言ったように、0 以外の値は真を意味する。すなわち、n が 2 でも 5 でも割れないときに限り真と判定される。どちらか一方の数で割れてしまえば偽だ。このことから、if 構文は n が 2 か 5 で割れる場合は実行されない。

さあ、問題ない n の値を手に入れたら割り算の開始だ。始めの `r = 1` は分子の 1 だが、計算が進むたびに分子の値が変わるので、変数 r を用いているのだ。計算の回数を示す範囲が `cyc(n)+1` であることに注意てほしい。range 関数の繰り返しは、第 2 変数の値より 1 小さい値までということを思い出そう。ただ、ここの for ブロックに使われる i は単なるカウンタだから、`in range(cyc(n))` と書くほうが分かりやすいかもしれない。

商の表示と、次回使う余りの計算も以前のスクリプトと同じだ。最後に "..." を付けて体裁を整えたら完了だ。これで安心して循環節が分かるようになった。遠慮せずに $\frac{1}{n}$ を計算させよう。