## 7.2 循環節の計算

循環小数の循環節を無駄なく見ることができるようになった。しかし、不十分だ。われわれが知りたいのは、$142857 \times 2 = 285714$ のように循環節が巡回をしているかどうか、である。cydec 関数は、循環節を眺めることはできても計算をすることはできない。なぜなら、cydec 関数が吐き出した数はまとまった数値でなく、桁ごとの数だからだ。

桁ごとの数をまとまった数値にするには、ひとつの変数にまとめてやればよい。

[py script]

```
>>> def cyc(n):
...   r = 10 % n; c = 1
...   while r != 1:
...     r = (r * 10) % n
...     c += 1
...   return c
...
>>> def cyval(n):
...   r = 1; val = ''
...   for i in range(cyc(n)):
...     val += str(r * 10 // n)
...     r = (r * 10) % n
...   return val
...
>>> cyval(7)
'142857'
>>> cyval(91)
'010989'
```

まず、cyc 関数は以前の流用である。もし、前回のスクリプトを実行していたなら、今回は cyval 関数の入力だけでよい。なぜなら **Python3** がちゃんと記憶してくれているから。

さて、cyval 関数が cydec 関数と違うのは、数字をひとつずつ print する代わりに、val 変数へまとめている点である。そのために val を''（**ナルストリング**）で初期化している。0 でなく'' で初期化したのには訳がある。0 で初期化すると変数 val は数値扱いとなる。'' なら文字列扱いとなる。**Python3** では変数の型は自在にできるので、どっちでも同じようなものだが、ここはぜひとも文字列で扱いたかったのだ。というのは、割り算の最初が 0 になっても 0 を表示したいからだ。数値だとそれは難しい。しかし、'' で初期化した場合、逆に計算には不向きとなる。どちらの手法をとるかは、スクリプト作成上のコストの問題なのである。ちなみに r と val の初期化を 1 行にまとめたのも、行数をケチった以上の意味はない。

もし、先頭の 0 を表示しなくてよいなら、val = 0 で初期化した上で、val += str(r * 10 // n)

と書いた部分を

[py script]
```
...       val = 10 * val + (r * 10) // n
```

に直せばよいだろう。これで一気に val が数値になる。

違いはもうひとつある。if 文が消えていることに気づいたかな。実は、cyval 関数は別の関数—循環節にいくつかの数を掛け、循環の様子を見る関数—の部品である。そこで、入力されて困る値をはねるための if 文は、本命の関数に移してある。

ところで val += str(r * 10 // n) は足し算をしているが、文字列に “+” を用いて演算すると、**Python3** は自動的に文字列の連結をしてくれる。ただし、文字列の後ろにある数 “値” まで自動的に数 “字” と見なしてくれないので、str 関数で文字列に変換する必要はあるけれど。

さあ、いよいよ循環節に数を掛けてみよう。処理は簡単だ。文字列を数値に変換するため int 関数を使い、循環節の回数分の掛け算をすればよい。よって、次のようなスクリプトを追加しよう。

[py script]
```
>>> def cycalc(n):
...   if (n % 2) and (n % 5):
...     print(cyval(n))
...     for i in range(2, cyc(n)+1):
...       print(i * int(cyval(n)))
...
>>> cycalc(7)
142857
285714
428571
571428
714285
857142
```

スクリプトはちょっと変わったことをしている。それは、最初に cyval(n) を出力して、2 から循環節分の数まで掛けていることだ。1 から循環節分の数まで掛けるコードではまずいのだろうか。

その理由は、ほんのわずかのこだわりからである。$\frac{1}{7}$ の循環節では問題ないけれど、分母が 2 桁になると循環節の先頭は 0 になってしまう。つまり、n が 2 桁の場合でも 0 を表示するようしたかったので、最初の値だけは文字列で出力したのである。しかし、これで十分でないことはすぐ分かるだろう。

分母が 3 桁になると、先頭から 2 桁が 0 になるからだ。一般に、分母が $n$ 桁なら、先頭から $(n-1)$ 桁が 0 だ。これらの場合すべてに対応するには、状況によって書式を変えなくてはならない。その場合は、n の桁数 $f$ はすぐ分かるので、最初の $f$ 個の数値だけを文字列で表示すればよい。ただ

し、そのまま表示したのでは 0 がついていないので、あらかじめ $(f-1)$ 個の 0 を連結しておく必要がある。**Python3** では、文字の連結には +演算子を使う。スクリプトはちょっと面倒になるけど、見栄えに凝るなら、ぜひスクリプトを書き換えよう。

ただ **Python3** には、たとえば `print('%06d' % (i * int(cyval(n))))` と書いて先頭に 0 を埋めることは可能だが問題がある。`'%06d'` というのは出力が 6 桁に満たないときに限り 0 が埋められるのであって、6 桁を超えたら 0 が埋められずに出力される。じゃあ`'%099d'` のように値を大きく取ればよいかというと、それはうまくない。実際に試してみれば、この方法が今回の出力には不向きと分かるだろう。スクリプトを書き換えるならば知恵を絞ってほしい。