9.2 11 の倍数を判定する

九去法に関連して、倍数調査の方法を述べておこう。いずれも、九去法と同じ理屈である。

2 の倍数 一の位が偶数

3 の倍数 各桁の数字の和が 3 の倍数

4 の倍数 下 2 桁が 4 の倍数

5 の倍数 一の位が 0 か 5

6 の倍数 一の位が偶数で、各桁の数字の和が 3 の倍数

7 の倍数 （後述）

8 の倍数 下 3 桁が 8 の倍数

9 の倍数 各桁の数字の和が 9 の倍数

10 の倍数 一の位が 0

7 の倍数については後述することにする。

11 の倍数を見つける簡単な方法もある。11 の倍数とは、当たり前だが何らかの数に 11 が掛けられている。そこで、(ある数) $=[A][B][C][D]\times 11$ とおいてみよう。$[A][B][C][D]$ は各桁が A、B、C、D の 4 桁の数を表すものとする。このとき、ある数は以下の計算が示すように $[A][A+B][B+C][C+D][D]$ という 5 桁の数である。

$$\begin{array}{cccccc} & A & B & C & D \\ \times) & & & 1 & 1 \\ \hline & A & B & C & D \\ A & B & C & D & \\ \hline [A] & [A+B] & [B+C] & [C+D] & [D] \end{array}$$

このとき、各桁の数字に対して、足す引くを交互にしたらどうなるだろうか。それは

$$A-(A+B)+(B+C)-(C+D)+D=0$$

である。すなわち、11 の倍数は、各桁の数字を交互に足したり引いたりして 0 になる数なのである。

ここで、11 倍したとき桁上がりがあったら、話が違ってくるだろうと思うはずだ。そう、たしかに話が違ってくる。かりに $[C+D]$ に桁上がりが生じて、それが $[B+C]$ の桁へ繰り上がっていたとしよう。すると、ある数は $[A][A+B][B+C+1][C+D-10][D]$ と書かれるはずだ。この場合、各桁の数字を交互に足したり引いたりすれば、

$$A-(A+B)+(B+C+1)-(C+D-10)+D=11$$

である。0 にはならないが、11 になっている。それは、桁上がりが生じた桁から 10 を引き、繰り上がった桁には 1 が足されるわけだから、そういう桁が多ければ 11 がたくさん作られる。しかし、桁の場所によっては −11 となることもある。

そういうことがうまく運べば、11 と −11 が相まって、合計は 0 だ。万一、同じ符号の 11 ばかり作られても、合計の絶対値は 22、33、44、... であろう。これらは 11 の倍数だから、やはり、各桁の数字を交互に足したり引いたりすれば 0 になるのだ。

結局、各桁の数字を交互に足したり引いたりした合計が 0 にならなければ、さらに合計の各桁の数字を足す引くすればよい。最終的に一桁の数になると思うが、それが 0 なら 11 の倍数と言える。

かくして、11 の倍数を見つける関数を **Python3** で書くと次のようになる。もちろん、単に 11 で割るスクリプトでは芸がないんだからね。

[py script]

```
>>> def elmethod(n):
...   s = str(n)
...   while len(s) > 1:
...     sum = 0; sgn = 1
...     for c in s:
...       sum += int(c) * sgn
...       sgn = -sgn
...     s = str(abs(sum))
...   if s == '0':
...     print('11 の倍数')
...   else:
...     print('11 の倍数でない')
...
>>> elmethod(348011)
11 の倍数でない
>>> elmethod(11*522)
11 の倍数
```

nimethod 関数にちょっと手を加えただけだ。各桁の数字を足したり引いたりを交互にするために、前にも使った sgn 変数を用いている。変数 sum を文字列に変換する際、abs 関数で絶対値に直している点に注意してほしい。なぜなら、合計は −11 のように負の数になることがあるので、それを無造作に文字列にしてしまうと、'-' で始まる文字列ができてしまう。すると、スクリプトの各方面に問題を起こすので具合が悪いのだ。

さて、7 の倍数の調べ方である。これは、調べたい数を一の位から 3 桁ずつ区切って、3 桁のブロックごとに足したり引いたりを交互に繰り返した合計を求め、それが 7 の倍数かどうかで判断する。なぜ、3 桁ずつ区切るかといえば、$1001 = 7 \times 11 \times 13$ であることを利用しているからだ。

たとえば、$[A][B][C][D][E][F]$ という 6 桁の数は $1000 \times [A][B][C] + [D][E][F]$ だが、これは

$$1001 \times [A][B][C] + [D][E][F] - [A][B][C]$$

と書き直すことができる。1001 は 7 で割れるので、あとは $+[D][E][F] - [A][B][C]$ が 7 で割れるかどうかを調べればよいことになる。これが 3 桁のブロックを交互に足して引く理由である。桁が増えれば 1001×1000、1001×1000^2、... を利用することになるが、同じような計算をすれば 3 桁ずつの区切りを浮き出させることができる。ただ、この方法はいささか冗長だが、他によい方法がないのも事実である。

また、$1001 = 7 \times 11 \times 13$ ということは、同じ方法で 11 の倍数と 13 の倍数も調べられるということでもあるが、11 の倍数については、先の方法が簡便だ。7 の倍数を調べるスクリプトを書きたいと思ったら、君たち自身で書いてみたらどうだろう。