0.3 濃度の話

それでは、もう少し濃度について語ることにしよう。

偶数や奇数は自然数と一対一の対応がつく、という理由から濃度が等しいことを説明した。今度は分数、すなわち有理数の濃度についてだが、結論を先に言えば、有理数の濃度は自然数の濃度に等しい。つまり有理数は自然数と一対一の対応がつくのだ。表を見てもらいたい。

$$
\begin{array}{c|ccccccccccc}
{}_{b}\diagdown^{a} & 1 & & 2 & & 3 & & 4 & & 5 & & \cdots \\
\hline
1 & \frac{1}{1} & \rightarrow & \frac{2}{1} & & \frac{3}{1} & \rightarrow & \frac{4}{1} & & \frac{5}{1} & \rightarrow & \cdots \\
 & & \swarrow & & \nearrow & & \swarrow & & \nearrow & & & \\
2 & \frac{1}{2} & & \left(\frac{2}{2}\right) & & \frac{3}{2} & & \left(\frac{4}{2}\right) & & \frac{5}{2} & & \cdots \\
 & \downarrow & \nearrow & & \swarrow & & \nearrow & & & & & \\
3 & \frac{1}{3} & & \frac{2}{3} & & \left(\frac{3}{3}\right) & & \frac{4}{3} & & \frac{5}{3} & & \cdots \\
 & & \swarrow & & \nearrow & & & & & & & \\
4 & \frac{1}{4} & & \left(\frac{2}{4}\right) & & \frac{3}{4} & & \left(\frac{4}{4}\right) & & \frac{5}{4} & & \cdots \\
 & \downarrow & \nearrow & & & & & & & & & \\
5 & \frac{1}{5} & & \frac{2}{5} & & \frac{3}{5} & & \frac{4}{5} & & \left(\frac{5}{5}\right) & & \cdots \\
 & & & & & & & & & & & \\
\vdots & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \ddots
\end{array}
$$

( ) と矢印を無視しておけば、ここにはすべての有理数があることが分かるだろう。もっとも 0 と負の有理数を省いているが、この先の話の本質を変えることはないし、簡単のために正の有理数だけで説明を続けるとしよう。

有理数の濃度が自然数の濃度に等しいことを示すには、表にあるすべての有理数に $1, 2, 3, \ldots$ と番号が振れることを示せばよいわけである。それには、表の有理数を矢印の順に沿って番号を付けるだけで済む。( ) で囲った数は、すでに番号が振られている有理数と同じ値なので、飛ばしておく。すると有理数が自然数に一対一に対応していることが分かるはずだ。えー？　そうかな？　このような番号の振り方では、対角線の左上に番号が付いても、それと同じ量の番号の付いてない数が右下に残るんじゃないの？　そうだね。確かにそんな気がするかもしれない。でも、それはわれわれが無限をよく理解できないからである。この番号の付け方で、もれなく有理数には番号が付くのだ。いくら右下に無尽蔵の有理数があっても、自然数だって無尽蔵にあるんだから。

すると今度は小数の番である。小数は有理数と違い、自然数によって番号を振ることができない。カントールは次のように説明している。

いま、すべての小数にもれなく番号を振ることができたと仮定しよう。そしてすべての小数を番号順に

$$
\begin{aligned}
d_1 &= 0.\alpha_1\alpha_2\alpha_3\alpha_4\ldots \\
d_2 &= 0.\beta_1\beta_2\beta_3\beta_4\ldots \\
d_3 &= 0.\gamma_1\gamma_2\gamma_3\gamma_4\ldots \\
d_4 &= 0.\delta_1\delta_2\delta_3\delta_4\ldots \\
&\vdots
\end{aligned}
$$

と、並べたとする。そして、これらの数をもとに、次の操作で新たな小数 $\dot{x}$ を作る。$\dot{x}$ の小数第 1 位の数字は、d_1 の小数第 1 位の数字と “違う” 数字 $\dot{\alpha}_1$ を選ぶ。また、小数第 2 位の数字は、d_2 の小数第 2 位の数字と “違う” 数字 $\dot{\beta}_2$ を選ぶ。このように、$\dot{x}$ の小数第 n 位の数字には、n 番の番号が付いた数の小数第 n 位とは “違う” 数字を選ぶのである。すると新しい数

$$\dot{x} = 0.\dot{\alpha}_1\dot{\beta}_2\dot{\gamma}_3\dot{\delta}_4\ldots$$

が出来上がるが、$\dot{x}$ はこれまでに番号を振られた数の中にない数である。

ちょっと変だぞ。初めの仮定では、すべての小数にもれなく番号が振られたはずなのに、いま出来上がった $\dot{x}$ は番号を振られた数とは明らかに違っている。この矛盾が生じた理由は、最初にすべての小数に番号を付けることができると仮定したからだ。仮定は間違っている。すなわち、小数には自然数と同じ番号を振ることができない。つまり、小数の集合と自然数の集合では濃度は違うのだ。それも小数のほうが “濃い” 濃度を持っていることになる。

集合の濃度といっても無数に**要素**を含んでいるので、その量を数値で表すことなどできない。そこでわれわれは、自然数の集合の濃度を $\aleph_0$—$\aleph$ はヘブライ文字で “アレフ” と読む—で表すことにしよう。すると、これまでに分かったことから、偶数の集合と奇数の集合の濃度も、さらには有理数の集合の濃度も $\aleph_0$ である。しかし小数の集合の濃度は $\aleph_0$ ではない。

おおまかに言って、小数で表すことができる数は**実数**と呼ばれている。もちろん有理数は実数である。なぜなら $\dfrac{1}{3} = 0.333\cdots$ や $5 = 5.0$ のように、あらゆる有理数は小数表記にできるからだ。さて、実数の集合の濃度は $\aleph_0$ ではないので、今度はそれを $\aleph$ で表すことにしよう。これまでの話では、どうやら

$$\aleph_0 < \aleph$$

であるようだ。

すると、ちょっとした疑問が湧いてくるだろう。$\aleph$ より “濃い” 濃度を持つ集合はどんな集合だろう。また、$\aleph_0$ と $\aleph$ の間の濃度を持つ集合はあるのだろうか。残念なことに、その話題を続ける力量が私にはない。この先の大秘境に興味がある人は、足を踏み入れてみてはいかが？

濃度の話から少し離れてしまうかもしれないが、数学において一対一の対応は重要な概念である。有理数の濃度が自然数の濃度と同じというのは、有理数が 1 ずつ増える自然数の性質を継承している表れでもある。**Python3** は 1 ずつ増える処理をどうするのだろう。簡単な例で見ておこう。

[py script]

```
>>> x = 0
>>> while x < 100:
...   print(x, end=' ')
...   x = x + 1
...
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 7
6 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
```

スクリプトは、0 から 99 までを画面に表示するだけのものである。途中、数値が切れて表示されているのは、ウィンドウサイズによるもので、エラーでもなんでもない。

まず、最初は 0 を表示したいので、変数 `x` に `0` を代入している。いきなりの初登場が `while` ステートメントだ。`while` ステートメントは繰り返しをするときによく使われる。繰り返しは、字下げされたブロックの処理を、`x` が `100` になる前—すなわち 99 まで—逐次行う。`while` などのループ文は行末に “`:`” を必ずつける約束である。`while` 文を入力するとプロンプトが “`...`” に変化するが[1]、この辺の詳しい説明は他のメディアに譲っておく。気にせず入力を続けよう。

Python3 で特徴的なのは、ブロックの始めから終わりまでを “字下げ” で判断することだ—この散歩では、空白 2 個ずつの字下げにしてある。すなわち、`print(x, end=' ')` から `x = x + 1` までが `while` ステートメントのブロックで、`x < 100` である間、繰り返される部分である。

`print(x, end=' ')` は `x` の値を改行せず空白を入れて表示させる命令である。もし “`, end=' '`” がないと、数は改行されながら表示される。要するに縦にズラーッと表示されるのだ。ちょっと見苦しいね[2]。

そして、数をひとつ表示したら `x` の値を 1 増やす必要があるので、`x = x + 1` とした。この記法は等式と見れば正しくないが、多くのプログラム言語では “`=`” は代入の意味で使われる。

入力の終了は “`...`” が表示されているとき、enter キーを押すことである。それで、入力の終了

[1]Terminal Python の場合。公式サイト Python では何も表示されず、自動的にインデント（字下げ）が行われる。
[2]ちなみに verion 2.x 以前では、`print(x),` と書けば改行せず空白を入れながら表示された。

が伝わり結果が上記のように表示される。

ところでステートメントとは、言うなれば英文法の構文のようなものである。そこで散歩中は、文字数を 5 文字ケチれるという理由で、“ステートメント” を “構文” と表現しておく。ただし、ステーメンとは基本的にブロック単位でもあるので、同時にブロックと表現することもある。どちらも似たようなものだと思ってほしい。

さて、いまは 1 ずつ回数が増えていく様子を見たが、2 ずつ増やしたいときもある。その場合は、`x = x + 1` を `x = x + 2` とするとよい。これで偶数だけが表示される。

また、99 から 0 まで逆順に表示させるなら、`x = 99` から始めて `x = x + 1` を `x = x - 1` にすればよいが、`while` の行を放っておくと無限ループに近い状態に陥ってしまう。なぜなら、いつだって `x < 100` だからだ。当然ここも `while x > -1` に直さなくてはならない。