0. . . の散歩道

0.1　自然数から整数への継承

われわれはものを数えるとき

$$1,\ 2,\ 3,\ 4,\ 5,\ \ldots,\ 10,\ \ldots,\ 100,\ \ldots$$

と数えるのが普通である。これらの数はどこまでも大きく数えられるし、どこまでいっても終りというものがない。数学の世界では、ここに登場した数を**自然数**と呼んでいる。

まず、約束がなされたことに気づいただろうか。いま登場した数を自然数と呼ぶ約束のことだ。ここで、何でそう呼ぶの？とか、0 は自然数じゃないの？とかの疑問が浮かぶかもしれないね。数学では “何々のことを何々と決める” ということが頻繁にでてくる。そしてその決め方が自然と納得できるものもあれば、場合によってはどうしてそう決めるのか不思議に思うことがあるだろう。約束—すなわち定義—というものは、大体の雰囲気で決める場合もあれば、深い意味があってそう決める場合がある。君たちが戸惑うのは、深い意味があって定義されることがらだろう。その場合は、定義に納得いかないこともあるはずだが、深い意味があるだけに当面は謎のままになってしまうものだ。その定義に関する内容を深く理解したとき、謎が氷解することが往々にしてあるので、楽しみはとっておくのがよいだろう。

自然数という呼び名は、大体の雰囲気から妥当と思われる。人間が自然発生的に使い出した数が 1, 2, 3, . . . だから。

ちょっと待ってほしい。われわれが現在、自然に使う数字は 0, 1, 2, 3, . . . ではないか。それなら 0 を含めて自然数と呼ぶほうが自然だろう。こう考える人はいるかな。たしかに、そのとおり。それは間違った考えではない。ただ、ここでは習慣に従い、1 から数える数を自然数であると定義しておこう。あとで分かるように、0 は特別扱いしたい数だから。

ところでわれわれが普段使う数には -1, -5 などもある。このような数は 0 より小さい数を表すために作られた数だ。自然界には 0 より小さいものはないと言ってよいだろう。何もない状態が 0 の状態だから、もうこれ以上なくなる状態はありえないのだ。ところが人々は 0 より小さい状態を

概念として作ってしまった。借金や気温や水深などはそのよい例になっている。

0 より小さい数は、自然数に − の**符号**をつけて表している。1 や 5 は、0 より 1 や 5 だけ大きい数ととらえるならば、−1 や −5 は、0 より 1 や 5 だけ小さい数ととらえることができる。数に − の符号をつけて、0 より小さい数を表すことは新たな約束となる。しかし、ここでは約束もさることながら、数の継承がなされたことに注意を払ってほしいのだ。

もし、負の数を新たに定義するだけなら a, b, c, ... という “数” を作れば済む。ところが実際は、自然数を土台に −1, −2, −3, ... と数を拡張している。つまり自然数を継承したわけだ。継承するということは、自然数の持つ性質も受け継ぐことを意味している。たとえば数の間隔は、負の数でも 1 ずつだし、大きい数字を使うほど 0 から離れた数になることなどがそうだ。

結局、数は 0 を中心にして

$$\ldots,\ -10,\ \ldots,\ -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ \ldots,\ 10,\ \ldots$$

と並んでいることになる。そしてこれらの数を**整数**と呼び、さまざまな数の基準として使うことにするわけである。

それではこの辺で、**Python** に何かさせてみよう。散歩中に書くのは小規模のスクリプトである。数学の話題にはなるべく詳しい説明をするけれど、スクリプトの説明はほどほどで済ませている。理由は、数学の考え方のほうを重視しているからだ。もしスクリプトが難しいと感じるなら、それは言語の理解が不足しているためではない。問題解決のための知識が不足しているのだ。

Terminal のプロンプト % に対して python と入力して **Python** を起動すると、画面は

[py script]

```
% python
WARNING: Python 2.7 is not recommended.

:（以下数行のコメント）
>>>
```

となりプロンプト >>> が表示される[1]。コメントは旧バージョンで **Python** を起動したことによる注意書きである。そこで改めて ctrl+z でプロンプト % へ戻り python3 と入力すると、画面には

[py script]

```
% python3
Python 3.8.2 (default, Sep 24 2020, 19:37:08)
:（以下数行のコメント）
>>>
```

[1] Mac OS Catalina 10.15.7 の Terminal の場合。

が表示されるだろう。これで version 3.x の **Python** が起動した。version 2.x と version 3.x の違いは今回の散歩では致命的な結果になることがある。よって、以後 version 3.x を前提にしていることを注意喚起するためにも **Python3** と書くことにしよう。さて、**Python3** は簡単な計算なら数式を入力するだけで答を出してくれる。

[py script]
```
>>> 10+20
30
```

いくつかの計算をまとめてすることもできる。

[py script]
```
>>> 10-20, 35*12, 2**10, 100/3, 100/4, 100//4, 100//3, 100.//3
(-10, 420, 1024, 33.333333333333336, 25.0, 25, 33, 33.0)
```

Python3 は結果を ( ) にまとめて表示するのが特徴的である。見てのとおり**べき**計算 a^b は `a**b` と入力する。割り算には "/" を使う。割り切れないときはちゃんと小数点以下を表示するが、でも末尾が変だ。これは、コンピュータが 2 進数で計算していることが原因である。この点は、しばらく歩いてから説明する予定だ。"/" での割り算は割り切れても小数点を表示する。確実に整数値が欲しいときは "/" でなく "//" を用いるのがよい。その場合は、割り切れないときでも整数値しか表示しないが、"." を含んだ計算の結果には小数点が付く[2]。"/" と "//" の使い分けは大事だ。

Python3 のすごいところは、巨大な数の計算が難なくできることだ。

[py script]
```
>>> 2**500
3273390607896141870013189696827599152216642046043064789483291368096133796
4046745548832700923259041571508866841275600710092172565458853930533285275
89376
```

普通の言語では 2^{500} など、軽く**オーバーフロー**してしまうのに、**Python3** はメモリの許す限りきちんと結果を出してくれるのだ。これは今回の散歩の相棒として最高の性質である。

文字に値を代入して、それを使って計算することも簡単だ。

[py script]
```
>>> a = 3
>>> b = 4
>>> c = a**2 + b**2
>>> c
25
```

[2] ちなみに version 2.x では、`100/3 = 33`、`100//3 = 33`、`100./3 = 33.33`$\cdots$ となる。

ここでは a に 3、b に 4 を代入して、c に a**2 + b**2 の結果を代入している（もちろん a^2+b^2 の計算だよ）。文字に代入された値を知りたければ、その文字を打ち込めばよい。ちなみに、数式のところどころに空白を入れているが、単に見やすくするための処置にすぎない。面倒ならすべて詰めて書いてもよいけれど、できる限り見やすい記述を心がけたいものだ。

文字には、新たな値を代入するまでもとの値が保持されている。c の平方根が知りたければ sqrt を使うことになるけれど、それには import math と打ち込んだ上でないといけない。そういう仕組みなのだ。ただし、import するのは一度でよい。**python3** を終了させるまで有効である。

[py script]
```
>>> import math
>>> math.sqrt(c)
5.0
```

これで $\sqrt{25}$ が計算できたことになる。

このまま c に別の値を代入すれば c の値が変わる（正しくは c へのリンクが変わるのだが、今回の散歩ではあまり気にしないでおこう）。**Python3** では代入の際、いちいち型を気にしなくてよいのも便利な点である。言語によっては、代入の型が厳しく制限されるものも多いからだ。

[py script]
```
>>> c = 'tmt\'s math'
>>> c
"tmt's math"
>>> print(c)
tmt's math
```

このように、**Python3** は自由度の高い言語であるものの、ある部分では非常に気難しい面もある。いまの入力例を見ると、ちょっと気難しい面が垣間見えるはずだ。今回の旅では気難しいことは省いておこう。まあ、それでも多少のプログラミング作法は身に付くだろうから。それに何が気難しいかは、ほんの一握りの好奇心があれば誰にでも体験できる。いろいろ試してみればよいからだ。ただ、好奇心は満たされても不満が募るかもしれない。不満の解消は各自で調べてもらうしかないが。