1.3 コラッツの問題

有名な数列はフィボナッチ数列だけではない。フィボナッチ数列は不思議な数列だが、他に未だ未解決の数列がある。それは次の規則で作られる。

はじめに勝手な自然数を用意する。次の項は “前の項が偶数なら 2 で割り、奇数なら 3 倍して 1 を足す” というものだ。たとえば 50 から始めると

$$50,\ 25,\ 76,\ 38,\ 19,\ 58,\ 29,\ 88,\ 44,\ 22,\ 11,\ 34,\ \ldots$$

のような数列が出来上がる。数列はどこまでも続くように思える。しかし、そうではないのだ。簡単なスクリプトを書いて、いろいろな数で試してみよう。

[py script]

```
>>> while 1:
...   n = int(input('a = '))
...   for i in range(2, 25):
...     if n % 2 == 0:
...       n = n // 2
...     else:
...       n = 3 * n + 1
...     print(n, end=' ')
...   print('\n')
...
a = 50
25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2
```

ここではコンピュータプログラムにおける、重要な分岐処理を理解してほしい。コンピュータは繰り返し同じことをするのに苦痛を感じないが、思い通りの処理を自動的にしてくれるほど気が利いているわけではない。コラッツ[1]の問題と呼ばれるこの数列の行方を調べるには、偶数と奇数の判断ができなくてはならない。

最初は、開始する数の入力を促している。`input` 関数は前回から目にしているが、試す数は整数だから `int` に変換して `n` に代入した。ここでは、とりあえず 25 項程度までの数を調べるため、`for` 構文の範囲は `in range(2, 25)` とした（実際は 24 項まで）。範囲は 1 からでもよかったのだが、次の項が 2 項目という理由でそうしたに過ぎない。枝葉の問題である。この `for` 構文のブロックは、次に同じ字下げ位置にスクリプトが書かれる直前までだから、`print(n, end=' ')` までとなる。

ここから新たな命令の登場だ。分岐処理は `if` 文で行われる。`if` 構文の仕組みは、`if` の行に記述された条件に合致すれば字下げされたスクリプトの処理が行われ、条件に合致しなければ `else`

[1] ローター・コラッツ（1910–1990）：ドイツの数学者。

後の字下げされたスクリプトの処理が行われるのである。今回は、if 後の字下げ文も else 後の字下げ文も 1 文しかないが、同じ字下げで何行でも書いてよい。

ところで、ブロックについて重要な注意がある。if 構文もブロック単位でものを考えるけれど、if 構文には elif 文や else 文がセットで含まれる場合がある。ここでは、else 文がセットになってブロックを構成する。すなわち、if ブロックは else を含んで字下げが終わるところまでで、n = 3 * n + 1 までが if ブロックのかたまりと見る。今後、よく目にするだろうから気をつけてほしい。

さて、if の行に記述された条件 n % 2 == 0 であるが、これは “n を 2 で割った余りが 0 に等しい” と読んでおく。A % B は A を B で割った余りを求める計算式である、とは前に述べた。また、等しいかどうかの判定には “==” が使われる。“=” はあくまでも代入専用の演算子なのである。

そこで n が 2 で割れれば、**Python3** は if 後の n = n // 2 を実行し、else 後のスクリプトは実行しないで、ブロックを抜けて print(n, end=’ ’) へと移る。また、n が 2 で割れなければ、**Python3** は n = n // 2 のスクリプトを実行しないで、else 後の n = 3 * n + 1 を実行して、ブロックを抜けて print(n, end=’ ’) へと移る。つまり二者択一で処理が行われるのである。“, end=’ ’” は、改行しないで空白を表示させるためのお約束だったね。ちなみに n を 2 で割る際、n // 2 とした理由は、確実に 2 で割れるのだから整数除算が適切と考えたからだ。これを n / 2 にすると 25.0 76.0 ... のように小数点が表示される。

最後の print(’\n’) が気になるのではないだろうか。これがなくてもスクリプトは正常に作動する。ただ、見栄えの問題のために入れたにすぎない。というのは、print(n, end=’ ’) で次々と数を表示すると空白は入るが改行はしない。すると、すべての数が表示されても改行をしないので、次の入力を促す “a = ” が最後の数の直後に表示されてしまう。それは少々かっこ悪い。つまり、print(’\n’) は改行を強制する命令で、決して “\n” を表示する命令ではないのだ。**Python3** には**エスケープシーケンス**と呼ばれる、この手のコードがいくつかある。

いろいろな数に対して、コラッツの問題の操作を繰り返せば、結果の予想がつくだろう。その予想は正しい。しかしコラッツの問題は予想であって証明ではない。現在でも、どうやら確からしいという程度の予測はできても、未だ証明は得られていない。ところで繰り返しの範囲を 25 に区切ってしまうと、コラッツの操作が終了する前にスクリプトが終わることもある。それでは困る。そこで範囲を決めるのではなく、n が 1 になったところでスクリプトが終了するように改良しておこう。

改良は実に簡単だ。スクリプトを見ると、for 文を while 文に変えただけだ。これは、条件 n > 1

が**真**である限り構文内の処理を繰り返すことを意味する。逆の意味に取れば、n が 1 になったところで while ブロックが終わるということだ。初期値に 27 を与えるとなかなか楽しめる。

[py script]

```
>>> while 1:
...   n = int(input('a = '))
...   while n > 1:
...     if n % 2 == 0:
...       n = n // 2
...     else:
...       n = 3 * n + 1
...     print(n, end=' ')
...   print('\n')
...
a = 27
82 41 124 62 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 1
37 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780
890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 31
9 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734
1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 32
5 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2
 1
```