6.2　ユークリッドの互除法

完全数の話題に約数が出たので、今度は最大公約数を考えてみよう。

一般に知られている**最大公約数**の求め方は次のようなものだろう。

$$\begin{array}{rcrr} 2 &) & 60 & 36 \\ \hline 2 &) & 30 & 18 \\ \hline 3 &) & 15 & 9 \\ \hline & & 5 & 3 \end{array}$$

この場合、60 と 36 の最大公約数は除数に現れた数の積 $2 \times 2 \times 3 = 12$ である。ちなみに**最小公倍数**であれば、商に現れた数までの積 $2 \times 2 \times 3 \times 5 \times 3 = 180$ となる。

ところで、最大公約数は次のようにしても求められる。同じく 60 と 36 での例である。

$$\begin{aligned} 60 &= 36 \cdot 1 + 24 \\ 36 &= 24 \cdot 1 + 12 \\ 24 &= 12 \cdot 2 + 0 \end{aligned}$$

何をしているかと言えば、まず 2 数の大きいほう（被除数）を小さいほう（除数）で割って余りを求める。普通なら $60 \div 36 = 1$, 余り 24 と書くところを、気取って $60 = 36 \cdot 1 + 24$ と書いている。次にすることは、除数を被除数に昇格（？）させ、余りを除数に昇格させて、同様に割り算をし余りを求める。この繰り返しだ。繰り返しは、余りが 0 になった時点で終了だ。このとき、最後の除数が最大公約数になるのである。

不思議な顔をしているね。それならもう一丁、11 と 26 の最大公約数を求めよう。おそらく、見ただけで共通の約数がない—こういう 2 数を**互いに素**と呼ぶ—ので最大公約数は 1 だと気付くだろう。では、いまと同じ手順を踏んでみよう。

$$\begin{aligned} 26 &= 11 \cdot 2 + 4 \\ 11 &= 4 \cdot 2 + 3 \\ 4 &= 3 \cdot 1 + 1 \\ 3 &= 1 \cdot 3 + 0 \end{aligned}$$

余りが 0 になったとき、最後の除数は 1 である。よって最大公約数は 1、すなわち 11 と 26 は互いに素であることが分かる。

2

最大公約数を求めるこのアルゴリズムは**ユークリッドの互除法**と呼ばれている。この方法で最大公約数が求められる理由は、さほど難しい理論ではないが、この散歩は厳密なところで時間を使わない。詳しく知りたければ、整数について書かれた書物を読んでもらいたい。

ユークリッドの互除法を用いて、最大公約数を求めるスクリプトを書いてみよう。

[py script]

```
>>> def findgcm(a, b): # ユークリッドの互除法
...   while a % b != 0:
...     a, b = b, a % b
...   print(b)
...
>>> findgcm(60, 36)
12
>>> findgcm(11, 26)
1
```

スクリプト自体は単純だが、新たな演算子 “!=” を使っている。これは等しくないことを意味する演算子だ。すなわち、a が b で割り切れない限り while 構文が続くのである。ユークリッドの互除法では、割り算をし余りを求めることを繰り返し、余りが 0 になれば最大公約数が求められるはずであった。そのため、余りが 0 にならない間は while 構文をが繰り返されなくてはいけないのだ。もっとも、余りが 0 でなければそれは正数だから、while a % b > 0:でも同じことである。

さて、余りがあるうちは常に除数を被除数へ、余りを除数へ引き継いでいけばよい。それが a, b = b, a % b である。この手法は関数 contfrac(a, b) と全く同じことをしていることを確認してほしい。contfrac 関数では分子・分母の入れ換えだったのに、ここでは除数と余りの順送りなのだ。同じ記述でも違う作業をしてしまうから、スクリプトは厄介なのだ。自分が書いたコードが半年後には理解不能になる理由は、こんなところにもある。重要なスクリプトを書くなら、気の利いたコメントを数多く記述しておくべきだ。上のスクリプトには、気が利いているかどうか分からないがコメントを残してみた。

さて、余りが 0 になると while ブロックを抜ける。このとき最後の除数は b に代入されているので、b を出力すれば、それが求める最大公約数である。

ところで、人がユークリッドの互除法を使うとき、自然と 2 数の大きい数を小さい数で割り始めるはずだ。しかしこのスクリプトは $a < b$ である 2 数が入力されても正常に動くので安心してよい。

最大公約数の次は最小公倍数の番だ。ところで、最大公約数が分かれば最小公倍数は直ちに計算できることを知っているかね？　それはこういうことだ。

ふたつの数を M, N としておこう。この 2 数の最大公約数を g とすると、$M = mg$, $N = ng$ と書いてよいだろう。そして m, n は互いに素であることも重要だ。よって、M, N の最小公倍数は mng であることが分かる。

では、**Python3** が M, N の値を受け取ったとき、これだけの情報から最小公倍数を計算するにはどうすればよいだろう？　簡単なことだ。$\frac{MN}{g}$、すなわち $\frac{MN}{GCM(M,\ N)}$ でよい。ちなみに、$GCM(M,\ N)$ は M, N の最大公約数を意味する。そこで最小公倍数を求めるスクリプトは次のようになる。

[py script]

```
>>> def gcm(a, b):
...   if a % b:
...     return gcm(b, a % b)
...   else:
...     return b
...
>>> def lcm(a, b):
...   return a * b // gcm(a, b)
...
>>> lcm(60, 36)
180
>>> lcm(11, 26)
286
```

スクリプトは関数 `findgcm(a, b)` に関数 `lcm(a, b)` を付け加えるだけで済む。ただ、それでは面白みに欠けるので `findgcm` 関数を再帰を使って書き直した（ついでに関数の名前を `gcm` に変えた）。`lcm` 関数を新たに書かずとも、`gcm` 関数に数行のコードを書き加えて最小公倍数を求める関数にしてもよい。方法はいろいろあるものだ。君たちはどの方法が腑に落ちるだろうか。数学的考え方に重きを置くなら、再帰で書かれたコードが一番自然であってほしい。