2.2 指数関数

$f(x) = x^2$ と $f(x) = 2^x$ では、2 と x の位置が逆になっているだけなのに格段の差がある。$x = 5$ のとき、5^2 であれば 5×5 で済むが、2^5 であれば $2 \times 2 \times 2 \times 2 \times 2$ と計算する。誰だって地道に掛け算を繰り返すのだ。x^2 はいつだって 2 回の掛け算で済むのに、2^x は x の値によっては何回も何回も何回も...掛け算をしなくてはならない。計算機があれば楽なのになあ、と思った君、その考えは甘い。実はコンピュータだって何回も何回も何回も...掛け算をしているんだな。

指数関数を相手にしてみよう。

[py script]

```
>>> def powof2(x):
...   p = 1
...   while x:
...     p *= 2
...     x -= 1
...   return p
...
>>> powof2(10)
1024
```

関数名は powof2 とした。**Python3** の関数名は、基本的にアルファベットと数字の組み合わせで付けるとよい。ただし、先頭に数字は使えない。他にも名付けの規則はあるけれど、当面はこれで十分である。ところで **Python3** には、a^b を求める関数 pow(a, b) というのがある。だったら自分で作ることもないと思うだろうが、テレビで見た景色を自分の目で見に行くようなものだと考えてほしい。ついでにいうと、pow 関数は a や b に実数値を使えるが、いま書いた powof2 関数は 2 の散歩道だけに 2^x 専用で、x は自然数限定である。テレビが隅々まで紹介してくれるのに対して、徒歩では見学場所に限りがあるようなものだ。

powof2 関数の仕事は大体伝わっているだろう。はじめに p = 1 にしておけば、それ以後 2 を掛けるたびに p の値が 2 倍になる。必要な回数だけ掛けて、return 文で p の値を返せばすべてがうまくいくという寸法である。

いくつか見慣れない書き方があるので注意しておこう。まず、while 構文である。while 構文は直後に条件式を書く必要があるのに、ここでは x しか書いていない。でも、これでも十分な条件式になっているのだ。というのは、**Python3** は 0 を “偽”、それ以外の値を “真” と決めている。そのために、x の値だけで真偽の判定ができるということである。また、while ブロック中には p *= 2 の計算と x -= 1 の減算があるが、これらは p = p * 2 と x = x - 1 の意味である。いわば、略式の書き方と思えばよいだろう。散歩中よく出くわす書き方である。

結局 while ブロックは、2 倍の計算と計算回数の減算を行っていることになり、x が減り続けていても x > 0 である限り while 文の処理が続く。いずれ x は 0 になるので、そこでめでたく while ブロックの処理から抜けるのだ。以前 while 1: と書いて永久ループにできたのは、条件が常に真となってブロックから抜けることがなかったためである。。

while ブロックは字下げされているところまでなので、return p からは別のブロックとなる。つまり、while ブロックを抜けると p の値が返されるわけだ。もし、return p を while ブロックと同じ位置まで字下げしてしまうと、p を 2 倍するたびに p の値が返されるスクリプトになってしまう。それはそれで使い道もあるけれど、いまはそうなっては困る。

ちなみに、関数 powof2 関数は while 構文を用いて 2 を掛ける回数を制御したが、for 構文を使っても同じことができる。練習のために書き換えてみるのもよいだろう。

さて、**Python3** に組み込まれている pow 関数は a^b の計算ができる。いくら何でも 2^x 専用の関数では利用価値も小さいので、せめて n^x 程度の計算ができるようにしてみよう。

[py script]

```
>>> def powf(n, x):
...   p = 1
...   while x:
...     p *= n
...     x -= 1
...   return p
...
>>> powf(3, 5)
243
```

といっても掛ける数を 2 から n に変えただけだけど。しかし、関数の機能は格段にあがった。このように、関数は複数の**引数（ひきすう）**を取ることができる。使い方は powf(3, 5) のように引数を指定すればよい。pow 関数はもともと **Python3** に組み込まれているので、自分が作る関数は異なる名前にしなければいけない。

ところで関数名の名付けに規則があることは述べた。では、関数の作り方には規則はないのだろうか。実は規則はあってないようなものだ。私はこの散歩を介して、生産的なソフトウェアを作ったり効率的なプログラムを書いているわけではない。だから関数の作り方もいい加減なものだ。ただ習慣や慣例によって、関数の作り方のガイドラインは存在している。そして私には無関係なだけなのだ。

しかし関数の作り方の基本は、ひとつの関数に多くの機能を持たせないことだ。ひとつの関数にはひとつの機能が望ましい。

ひとつの関数に多くの機能を持たせると、トラブルが発生したとき、関数内のどこが問題であるかを特定しづらい。もし、ふたつの関数に分けてスクリプトが書けるなら是非そうしよう。なぜなら、トラブルの原因を特定しやすいからだ。関数には勢いよく機能を詰めがちだが、自分のためにもいくつかの小さな関数に分けるほうがよい。

そうは言っても、私はそこまで真剣に考えながら散歩していない。もともと書くスクリプトが小さいことと、これをもとに何かを作るわけではないからだ。あくまでも気まぐれな散歩をしているということである。

さてさて、ところで、もしこれまでに Terminal を使ってファイルに保存して実行してたら、関数の散歩道に入ってからはきっとうまく実行できてないはずだ。たとえばファイル名を `powf` とし `% python3 powf(3, 5)` を実行しようとしてもできないだろう。実行時に引数を伴うファイルは、スクリプトにその旨を追加記述した上で `% python3 powf 3 5` のようにする。スクリプトに何を記述するかについては Internet で調べればすぐに見つかるだろう。でも、スクリプトを実行したときに引数を入力できるように、`input` 関数を用いるほうが簡単かもしれない。ただし、それだと関数として使えないけど。