組み込み関数 pow の知られざる進化

Unknown Evolution of the Built-in Function pow

Hayao Suzuki

PyCon JP 2021

October 15, 2021

発表に際して

GitHub に資料があります

- https://github.com/HayaoSuzuki/pyconjp2021

Twitter のハッシュタグ

- #pyconjp_4 TBA

PyCon JP Discord

- #jp-2021-track-1 TBA

Who am I ?

お前誰よ

名前	Hayao Suzuki（鈴木　駿）
Twitter	@CardinalXaro
仕事	Software Developer @ BeProud Inc.

Who am I ?

監訳・査読した技術書（抜粋）

- 入門 Python 3 第 2 版 (O'Reilly Japan)
- Effective Python 第 2 版 (O'Reilly Japan)
- 機械学習による実用アプリケーション構築 (O'Reilly Japan)
- PyTorch と fastai ではじめるディープラーニング (O'Reilly Japan)
- 実践 時系列解析 (O'Reilly Japan) New!
- 機械学習デザインパターン (O'Reilly Japan) New!

https://xaro.hatenablog.jp/ にリストがあります。

Who am I ?

発表リスト（抜粋）

- レガシー Django アプリケーションの現代化 (DjangoCongress JP 2018)
- SymPy による数式処理 (PyCon JP 2018)
- Python と楽しむ初等整数論 (PyCon mini Hiroshima 2019)
- 君は cmath を知っているか (PyCon mini Shizuoka 2020)
- インメモリーストリーム活用術 (PyCon JP 2020)

https://xaro.hatenablog.jp/ にリストがあります。

今日の目標

組み込み関数 pow

- pow 関数は数のべき乗を返す関数
- Python に限らず、大抵の言語には pow 関数が存在する

Python 3.8 で機能追加

- 整数 m を法とする剰余類における乗法逆元が計算できる
- よくわからない単語を並べるな！

今日の目標

組み込み関数 pow の知られざる進化

- 「整数 m を法とする剰余類における逆元」の意味を理解する
- 「整数 m を法とする剰余類における逆元」を計算するアルゴリズムを理解する

今までの `pow` 関数

Python 3.7 までの `pow` 関数を復習しよう

整数のべき乗

定義（整数のべき乗）

整数 b と自然数 n に対して、べき乗 b^n を

$$b^n \triangleq \overbrace{b \times b \times \cdots \times b}^{n \text{ 個}}$$

と定義する。b を底、n を指数と呼ぶ。

整数のべき乗の例

$$2^{32} = 4294967296.$$

整数のべき乗

Python におけるべき乗

組み込み関数 pow または**演算子を使う。

べき乗の実行例

```
>>> pow(2, 32)
4294967296
>>> 2 ** 32
4294967296
```

べき乗剰余

定義（べき乗剰余）

自然数の底 b と自然数 n, m に対して、

$$b^n \bmod m$$

を m を法とするべき乗剰余と定義する。

べき乗剰余の例

$$2^{32} \bmod 65535 = 1.$$

べき乗剰余

Python におけるべき乗剰余

- 組み込み関数 pow で効率的に計算できる。
- **演算子および%演算子でも計算可能だが効率が悪い。

べき乗剰余の実行例

```
>>> pow(2, 262144, 65535)
1
>>> (2 ** 262144) % 65535
1
```

べき乗剰余

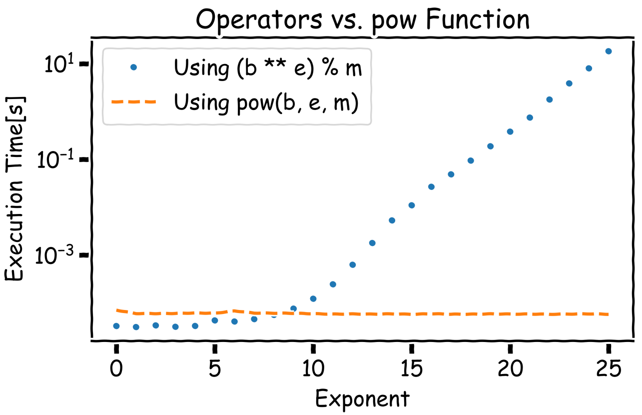
どれだけ効率的か

```
>>> import timeit
>>> timeit.timeit("pow(2, 262144, 65535)", number=1000)
0.0007324999999999693
>>> timeit.timeit("(2 ** 262144) % 65535", number=1000)
0.868453
```

結果を実行回数で割れば平均時間がわかる。

べき乗剰余

演算子と関数における計算時間の比較



これからの pow 関数

Python 3.8 からの pow 関数を理解するために

整数の合同

定義（整数の合同）

整数 a が m を法として b と合同であるとは m が $a-b$ を割り切ることをいい、

$$a \equiv b \pmod{m}$$

と表す。

整数の合同の例

$$47 \equiv 35 \pmod{6}$$

$47-35=12$ は 6 で割り切れる。

これからの pow 関数

定義（整数 m を法とする剰余類における乗法逆元）

整数 a, b と自然数 m に対して、

$$ab \equiv 1 \pmod{m}$$

となるとき、b を a の乗法逆元と呼び、a^{-1} と表す。

剰余類における乗法逆元の例

$$38 * 23 \equiv 1 \pmod{97}$$

38 の 97 を法とする乗法逆元は 23

剰余類における乗法逆元

Python における剰余類における乗法逆元

- 組み込み関数 pow の第 2 引数に -1 を渡せば計算可能
- これが Python 3.8 の新機能

剰余類における乗法逆元の実行例

```
>>> pow(38, -1, 97)
23
>>> (38 * 23) % 97 == 1
True
```

剰余類における乗法逆元

必ずしも乗法逆元が存在するとは限らない

```
>>> pow(2, -1, 6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: base is not invertible for the given modulus
```

与えられた法に対して底が乗法逆元を持たない（何故？）。

乗法逆元を求めて

乗法逆元の意味

整数 a に対して、m を法とする合同方程式

$$ax \equiv 1 \pmod{m}$$

を解くことに他ならない。

合同の定義に立ち返る

乗法逆元の意味

$ax \equiv 1 \pmod{m}$ を変形すると、不定方程式

$$ax - my = 1$$

が整数解 x, y を持つことに他ならない。

方程式を観察する

例：$3x + 9y$ なる数式

x や y に整数を代入するといずれも 3 の倍数となる。

	$x = -2$	$x = -1$	$x = 0$	$x = 1$	$x = 2$
$y = -2$	-24	-21	-18	-15	-12
$y = -1$	-15	-12	-9	-6	-3
$y = 0$	-6	-3	0	3	6
$y = 1$	3	6	9	12	15
$y = 2$	12	15	18	21	24

合同方程式の解

定理

整数 a, c に対して、m を法とする合同方程式

$$ax \equiv c \pmod{m}$$

は、c が $\gcd(a, m)$ で割り切れるときのみ、ちょうど $\gcd(a, m)$ 個の互いに合同ではない解を持つ。ただし、$\gcd(a, m)$ は a と m の最大公約数である。

証明は、適当な初等整数論の教科書を参照してください。

今回のケース

系

整数 a, に対して、m を法とする合同方程式

$$ax \equiv 1 \pmod{m}$$

は、$\gcd(a, m) = 1$ の場合のみ、1 個の互いに合同ではない解を持つ。

乗法逆元が存在するかどうかは数学的な裏付けがある。

剰余類における乗法逆元

乗法逆元が存在するケース

```
>>> import math
>>> math.gcd(38, 97)
1
>>> pow(38, -1, 97)
23
```

$\gcd(38, 97) = 1$ なので、97 を法とする 38 の乗法逆元が存在する。

剰余類における乗法逆元

乗法逆元が存在しないケース

```
>>> import math
>>> math.gcd(2, 6)
2
>>> pow(2, -1, 6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: base is not invertible for the given modulus
```

$\gcd(2,6) \neq 1$ なので、6 を法とする 2 の乗法逆元は存在しない。

Euclid の互除法

定理

a, b を $a \leq b$ である整数、r を a を b で割った余りとする。このとき、

$$\gcd(a, b) = \gcd(b, r)$$

が成り立つ。

Euclid の互除法

97 と 38 の最大公約数を計算する

$$
\begin{aligned}
97 &= 2 \times 38 + 21 \\
38 &= 1 \times 21 + 17 \\
21 &= 1 \times 17 + 4 \\
17 &= 4 \times 4 + 1 \\
4 &= 1 \times 4 + 0
\end{aligned}
$$

Euclid の互除法

97 と 38 の最大公約数を計算する

$a = 97, b = 38$ とすると

$$
\begin{aligned}
21 &= a - 2b \\
17 &= b - 21 \\
&= -a + 3b \\
4 &= 21 - 17 \\
&= 2a - 5b \\
1 &= 17 - 4 \times 4 \\
&= -9a + 23b
\end{aligned}
$$

Euclid の互除法

定理

不定方程式 a, b を 0 ではない整数とする。このとき、方程式

$$ax + by = \mathrm{gcd}(a, b)$$

は解 (x_0, y_0) を持ち、Euclid の互除法によって求めることができる。

pow の中身

longobject.c のコメントにある実装

```python
def invmod(a, m):
    x, y = 1, 0
    while m:
        q, r = divmod(a, m)
        a, m = m, r
        x, y = y, x - q * y
    if a == 1:
        return x
    raise ValueError("Not invertible")
```

Conclusion

まとめ

- pow 関数は数のべき乗を返す関数である。
- べき乗剰余を計算する場合は必ず pow 関数を使う。
- Python 3.8 で剰余類の乗法逆元が計算できるようになった。
- 乗法逆元が計算できる仕組みは Euclid の互除法にある。

pow 関数と Euclid の互除法ゎ…ズッ友だょ…！！