

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_1 TBA

PyCon JP Fellow Slack

- #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 @ TBA

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!**

<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)
0.00073249999999999693
>>> 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。

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
```

底と法となる数は互いに素である必要がある（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。

Conclusion

まとめ

- pow 関数は身近な存在
- pow 関数は身近な存在
- pow 関数は身近な存在
- pow 関数は身近な存在

pow 関数はズッ友！