

Python言語最新情報

～モダンな文法を知ってPythonを使いこなそう～

2019年9月28日 / OSC島根 / 清水川貴之

アジェンダ

- Pythonには30年近くの歴史がある
- 徐々に改良されている
- 最新の書き方を知って、使いこなそう

どんどん質問してください

Who am I?(お前誰よ?)

- 清水川貴之 / SHIMIZUKAWA Takayuki
- Twitter: [@shimizukawa](#)
- 一般社団法人PyCon JP([#pyconjp](#)) 会計理事
- 株式会社BeProud 取締役/IT Architect
- [Python mini Hack-a-thon\(#pyhack\)](#) 主催
- [Sphinx\(Pythonドキュメントツール\)](#) メンテナ



Python本書いています



1版: 2013/09/12
2版: 2017/10/20



2018/02/23



1版: 2010/5/28

2版: 2018/2/26

3版: 202x予定



1版: 2012/03/27
2版: 2015/02/27
3版: 2018/06/12



2020/02/27

島根に初めて来ました

私にとっての島根1



私にとっての島根2



一般社団法人PyCon JP

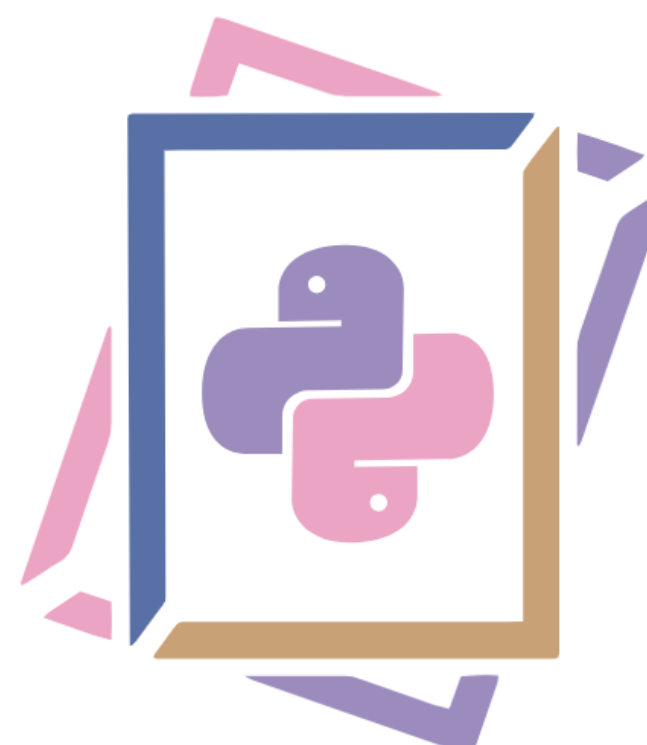
日本国内のPythonユーザのために、Pythonの普及及び開発支援を行う為に、継続的にカンファレンス(PyCon)を開くことを目的とした非営利組織

- <https://www.pycon.jp/>



PyCon JP 2019

- 国内最大のPythonイベント
- <https://pycon.jp/2019/>
- カンファレンス: 2019年9月16日(月、祝)、17(日)
- 会場: 大田区産業プラザPiO



PyCon JP 2019

Python New Era

2019.09.14 - 17

 #pyconjp

Conference (終了しました)

カンファレンス

09.16 月・祝 - 09.17 火

 大田区産業プラザ PiO

[タイムテーブル](#)[セッション一覧](#)[Youtubeでセッションを見る](#)[発表資料を見る](#)

PyCon JP 2019 の様子

- ポスターセッション(左), キーノート(右)



Python Boot Camp

- 初心者向けPythonチュートリアル
- <https://www.pycon.jp/support/bootcamp.html>



PyCon JPブースに来てね



最初に質問

プログラミングしたことある人☒☒

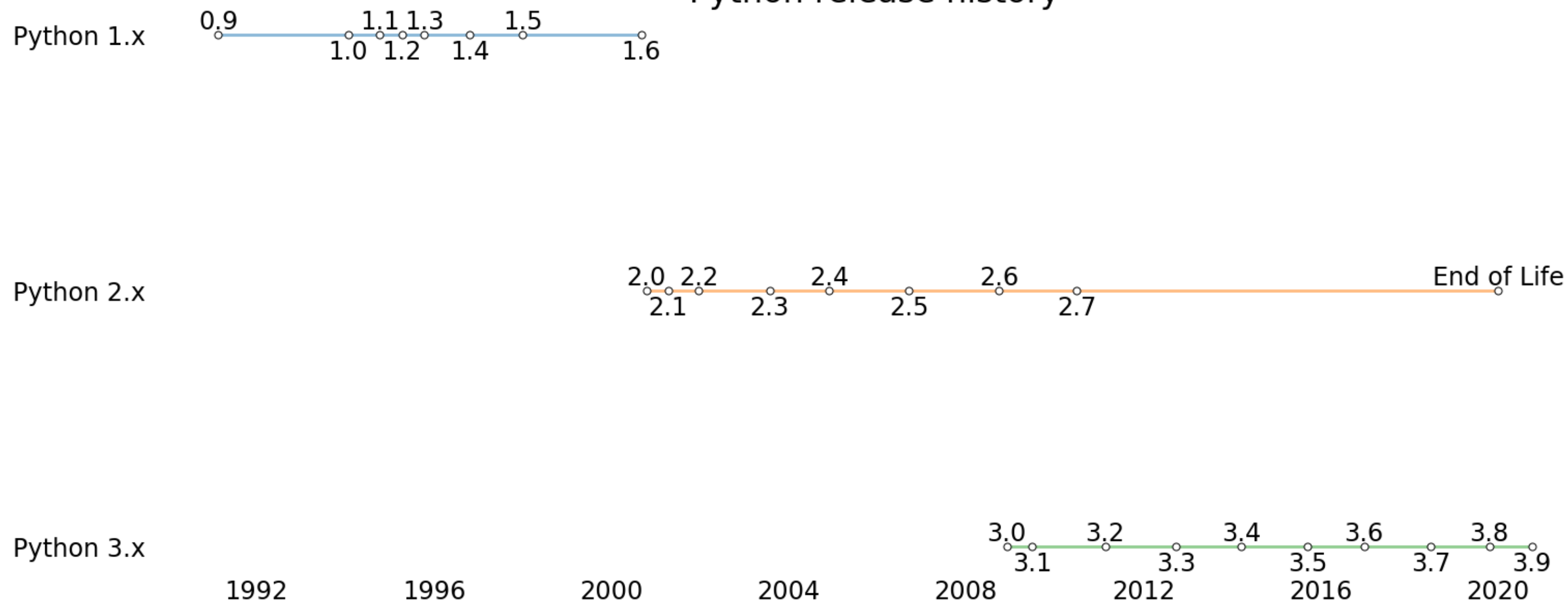
Python書いたことある人☒☒

Python 2を書いたことある人☒☒

Python 3を書いたことある人☒☒

Python開発の歴史

Python release history



PEP

- Python Enhancement Proposal
 - Pythonの拡張提案
- **PEP 1**: PEPの目的とガイドライン
 - 新機能を提案(Draft)
 - →議論して受理(Accepted)/否認(Rejected)
 - →実装→マージして終了(Final)

各バージョンでの重要な 変更

Python 2.4からPython 3.8まで

Python 2.4(2004年)

- What's New In Python 2.4
- PEP 218: ビルトインの集合(`set`)オブジェクト
- PEP 289: ジェネレータ式
- PEP 318: 関数とメソッドのためのデコレータ
- PEP 322: 逆順のイテレーション
 - `reversed()` 関数
- PEP 328: マルチラインインポート

Python 2.5(2006年)

- What's New In Python 2.5
- PEP 308: 条件式
- PEP 328: 絶対インポート、相対インポート
- PEP 341: `try/except/finally` の一体化
- 新モジュール - `ElementTree`、`sqlite3`、`ctypes` など

Python 2.6(2008年)

- What's New In Python 2.6
- PEP 343: `with` 文
- PEP 3101: 進化版文字列フォーマット
- PEP 3110: 例外処理の変更
 - `as` キーワード
- PEP 3119: 抽象基底クラス
- PEP 3129: クラスデコレータ

Python 2.7(2010年)

- What's New In Python 2.7
- PEP 372: `collections` に順序付き辞書を追加
- PEP 378: 1000区切りのための書式指定子
- PEP 389: コマンドライン解析のための `argparse` モジュール

Python 3.0-3.2(2008年)

- What's New In Python 3.0
- PEP 3105: `print()` 関数
- PEP 3106: `dict.keys().values().items()` の改良
- PEP 238: 除算演算子(/)の変更
- PEP 274: 辞書内包表記
- セットリテラルとセット内包表記

Python 3.3(2012年)

- What's New In Python 3.3
- PEP 397: Windows の Python ランチャ
 - `py` コマンド
- PEP 405: `venv`モジュール -- 仮想環境
- PEP 420: 暗黙的な名前空間パッケージ
- PEP 380: サブジェネレータへの委譲構文
 - `yield from`
- PEP 414: 明示的なユニコードリテラル -- `u'` あ'

Python 3.4(2014年)

- [What's New In Python 3.4](#)
- [PEP 453](#): Pythonインストール時のpipの明示的なブートストラッピング
- [PEP 435](#): `enum` モジュール
 - 列挙型のサポート
- [PEP 428](#): `pathlib` モジュール
 - オブジェクト指向のファイルシステムパス
- [PEP 450](#): `statistics` モジュール
 - 基礎的な統計ライブラリ

Python 3.5(2015年)

- [What's New In Python 3.5](#)
- [PEP 492](#): コルーチン、`async`構文と`await`構文
- [PEP 465](#): 新たな行列乗算演算子 `-- a @ b`
- [PEP 484](#): `typing` モジュール -- 型ヒント
- [PEP 441](#): `zipapp` モジュール
 - Python ZIPアプリケーションのサポート改善
- [PEP 471](#): `os.scandir()`関数
 - より良く、速いディレクトリイテレータ

Python 3.6(2016年)

- What's New In Python 3.6
- PEP 498: フォーマット済み文字列リテラル -- `f' '`
- PEP 515: 数値リテラル内のアンダースコア
- PEP 526: 変数アノテーションの文法
- PEP 525: 非同期(async)ジェネレータ
- PEP 530: 非同期(async)内包表記
- PEP 506: 標準ライブラリに `secrets` モジュール追加

Python 3.7(2018年)

- [What's New In Python 3.7](#)
- [PEP 553](#): `breakpoint()` 関数
- [PEP 557](#): データクラス

Python 3.8(2019年10月予定)

- [What's New In Python 3.8](#)
- [PEP 572](#): 代入式
- [PEP 570](#): 位置指定のみ引数
- fリテラルでの = によるデバッグ

新旧のスタイルを比較

PEP 328: マルチラインインポート

- 全バージョン

```
from module_name import func1, func2, \  
func3, func4, \  
func5, func6
```

- Python 2.4以上

```
from module_name import (func1,  
func2,  
func3,  
func4,  
func5,  
func6)
```

PEP 202: リスト内包表記(1/3)

- 内包表記なし

```
li = []  
for i in range(10):  
    li.append(i * i)
```

- リスト内包表記

```
li = [i * i for i in range(10)]
```


PEP 202: リスト内包表記(2/3)

- 内包表記なし

```
li = []  
for i in range(10):  
    if i % 2 == 0:  
        li.append(i * i)
```

- リスト内包表記

```
li = [i * i for i in range(10) if i % 2 == 0]
```

PEP 202: リスト内包表記(3/3)

- 内包表記なし

```
li = []  
for i in range(10):  
    for j in range(5):  
        li.append(i * j)
```

- リスト内包表記

```
li = [i * j for i in range(10) for j in range(5)]
```

リスト、辞書、セット内包表記

- リスト内包表記

```
>>> [i * i for i in range(10)]  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- Python 2.7以上
- PEP 274: 辞書内包表記

```
>>> {str(i): i * i for i in range(10)}  
{'0': 0, '1': 1, '2': 4, '3': 9, '4': 16, '5': 25, '6': 36, '7': 49, '8':
```

- セット内包表記

```
>>> {i * i % 10 for i in range(10)}  
{0, 1, 4, 5, 6, 9}
```

PEP289: ジェネレータ式

- ジェネレータ関数

```
def func():  
    for i in range(10):  
        if i % 2 == 0:  
            yield i * 1
```

- ジェネレータ式

```
(i * i for i in range(10) if i % 2 == 0)
```


sort() メソッドと sorted() 関数

- 全バージョン

```
li = [1, 3, 10, 2, 5]  
li.sort()
```

- Python 2.4以上(副作用なし)

```
li = [1, 3, 10, 2, 5]  
new_li = sorted(li)
```

PEP 308: 条件式

- 全バージョン

```
if condition:  
    x = true_value  
else:  
    x = false_value
```

- Python 2.5以上

```
x = true_value if condition else false_value
```

PEP 343: `with` 文

- 全バージョン

```
f = open('filename.txt')  
data = f.read()  
f.close()
```

- Python 2.6以上

```
with open('filename.txt') as f:  
    data = f.read()
```

%s / .format() / f リテラル

- 全バージョン

```
from datetime import date  
s = "Today: %s" % date.today()
```

- Python 2.6以上

```
from datetime import date  
s = "Today: {}".format(date.today())
```

- Python 3.6以上
- PEP 498: フォーマット済み文字列リテラル

```
from datetime import date  
s = f"Today: {date.today()}"
```

PEP 3110: 例外処理の変更

- Python 2

```
try:  
    1/0  
except Exception, e:  
    return e
```

- Python 2.6以上

```
try:  
    1/0  
except Exception as e:  
    return e
```


PEP3105: `print` 文と `print()` 関数

- Python 2
- Python 3

```
print "message"
```

```
print("message")
```

PEP238: 除算演算子 / と //

- Python 2

```
>>> 5 / 2  
2  
>>> 5 / 2.0  
2.5
```

- Python 3

```
>>> 5 / 2  
2.5  
>>> 5 // 2.0  
2
```

PEP 435: enum モジュール

- Python 3.4以上

```
import enum
```

```
class Tast(enum.IntEnum):
```

```
    todo = 1
```

```
    in_progress = 2
```

```
    done = 3
```

```
@classmethod
```

```
def get_task_types(cls):
```

```
    return tuple((x.value, x.name) for x in cls)
```

PEP 465: @ 演算子 -- a @ b

```
>>> import numpy as np
>>> a = np.array([[1, 2]])
>>> b = np.array([[3], [4]])
```

- 全バージョン

```
>>> np.dot(a, b)
array([[11]])
```

- Python3.5以上

```
>>> a @ b
array([[11]])
```

PEP 428: pathlib モジュール

- 全バージョン

```
import os
current = os.getcwd()
filepath = os.path.join(current, "dir", "filename.txt")
with open(filepath) as f:
    data = f.read()
```

- Python 3.6以上

```
from pathlib import Path
p = Path(".") / "dir" / "filename.txt"
with p.open() as f:
    data = f.read()
```


os.listdir() と pathlib

- 全バージョン

```
import os
for name in os.listdir(PATH):
    if not name.startswith('.') and os.path.isfile(os.path.join(PATH, name)):
        print(name)
```

- Python 3.6以上

```
from pathlib import Path
for entry in Path(PATH).iterdir():
    if not entry.name.startswith('.') and entry.is_file():
        print(entry.name)
```

PEP 557: データクラス

- Python 3.7以上

```
@dataclass
```

```
class InventoryItem:
```

```
    name: str
```

```
    unit_price: float
```

```
    quantity_on_hand: int = 0
```

```
    def total_cost(self) -> float:
```

```
        return self.unit_price * self.quantity_on_hand
```

PEP 553: breakpoint() 関数

- 全バージョン

```
import pdb; pdb.set_trace()
```

- Python 3.7以上

```
breakpoint()
```

PEP 572: 代入式 `-- a := b`

- "walrus operator" (セイウチ演算子) `:=`
- 全バージョン

```
m = re.match(pat, s)
if m:
    # mに対しての処理
```

- Python 3.8

```
if m := re.match(pat, s):
    # mに対しての処理
```

PEP 570: 位置指定のみ引数

- Python 3.8
- / の前の引数は位置指定のみ

```
def pow(x, y, z=None, /):  
    r = x**y  
    if z is not None:  
        r %= z  
    return r
```

```
pow(2, 10)    # OK  
pow(2, 10, 17) # OK  
pow(x=2, y=10) # NG  
pow(2, 10, z=17) # NG
```

fリテラルでの = によるデバッグ

- Python 3.8

```
x = 3  
print(f'{x*9 + 15=}')
```

- 以下のように出力される

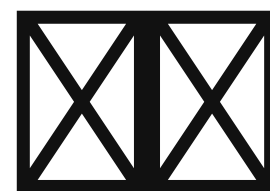
```
x*9 + 15=42
```


まとめ

まとめ

- Pythonは徐々に改良されている
- 後方互換性を維持しながら、新しい機能を追加している
- 少しずつ新しい文法を使っていこう

ありがとうございました



Question?