「Python言語」
二歩目を踏み出そう！

脱初心者になるためのポイントを伝授します

Takanori Suzuki / ヒカ☆ラボ / 2017年9月21日

Who am I?(お前誰よ)

- 鈴木たかのり / Takanori Suzuki
- Twitter: @takanory
- Pythonボルダリング部(#kabepy)部長

株式会社ビープラウド所属

- 株式会社ビープラウド

一般社団法人PyCon JP理事

- 一般社団法人PyCon JP

最初に質問

Python知ってる人?

Python書いたことある人?

他のプログラミング言語使っている人?

今日のゴール

(ほぼ)標準のPythonのみを使って、
いい感じにプログラムを書けるようになる
ヒントを得る

今日話すこと

1. 「はじめの一歩」の振り返り
2. pythonの一般的なデバッグ手法(pdb)
3. コーディングスタイルの統一(PEP8, flake8)
4. 自動テスト手法(unittest)
5. Python公式ドキュメントを読みこなそう
6. 文字列メソッドを使いこなそう
7. 例外処理の使いどころ
8. 内包表記、ジェネレーター式を使いこなそう
9. 次のステップ

1. 「はじめの一歩」の振り返り

「Python言語」はじめの一歩 / First step of Python

「Python言語」はじめの一歩 / First step of Python from **Takanori Suzuki**

1.1 Pythonの特徴

- まぁ、なんでもできる
- きれいに書きやすい
- 保守しやすい
- 20年以上開発してる
- Python 2 と 3 がある

1.2 言語の特徴

- インデントでブロック構造
- 対話モード
- たくさんの標準ライブラリ
- 豊富な外部パッケージ

1.3 インストール

- 公式からインストールしよう

1.4 言語仕様

- データ型: int, float, str, bytes
- コレクション: list, tuple, dict, set
- 分岐: if, elif, else
- 繰り返し: for, while
- ファイル操作: open(), with open
- モジュール: import

1.5 Python 2と3の違い

- printが文→関数
- 文字列がUnicodeに統一
- intの割り算の結果がfloat
- 標準ライブラリの再構成

1.6 よく使う標準ライブラリ

- re: 正規表現
- sys, os: システムパラメータとOS
- datetime: 日付と時刻
- math, random: 数学関数と乱数
- itertools: イテレータ生成関数
- shutil: 高レベルなファイル操作
- json: JSONエンコーダ、デコーダ

1.7 よく使うサードパーティ製パッケージ

- pip と venv(virtualenv)
- dateutil: 日時操作の強力な拡張
- Requests: HTTPクライアント
- BeautifulSoup4: HTML, XMLパーサ
- Pillow: 画像処理ライブラリ
- PEP8: pycodestyle, flake8

1.8 どうやって学ぶか

- 書籍紹介
- Web上のテキスト
- コミュニティに参加

2. pythonの一般的なデバッグ手法(pdb)

なぜデバッガー?

- `print()`デバッグはめんどう
- この変数の値が見たかった
  - 止めてコード書き換えて再実行
- pdbを使おう!

pdb

- Python標準のデバッガ
- インストール不要
- コマンドラインで操作

間違ったfizzbuzz

```
for num in range(1, 100):
    if num % 3 == 0:
        print('Fizz')
    elif num % 5 == 0:
        print('Buzz')
    elif num % 15 == 0:  # ここがおかしい
        print('FizzBuzz')
```

デバッグする

```
for num in range(1, 100):
    import pdb; pdb.set_trace()  # 追加
    if num % 3 == 0:
        print('Fizz')
    elif num % 5 == 0:
        print('Buzz')
    elif num % 15 == 0:
```

実行

```
$ python3 fizzbuzz.py
(env) Takanoris-MacBook-Pro:pycamp.pycon.jp takanori$ python fi
> /Users/takanori/fizzbuzz.py(3)<module>()
-> if num % 3 == 0:
(Pdb) p num
1
(Pdb) n
```

主なコマンド

- p: expression: expressionを評価した結果を表示
- n: 次の行に移動
- c: 次のブレークポイントまで実行
- h: ヘルプを表示
- l: ソースコードを表示
- q: デバッガを終了する
  他のコマンドは27.3.1. デバッガコマンドを参照

まとめ

- pdbでデバッグしよう

pdb - Python デバッガ

3. コーディングスタイルの統一(PEP8, flake8)

なぜ統一する?

- Pythonとしての統一ルールがある
- サポートするツールもある
- 統一しておけば揉めない

PEP8(1/2)

PEP 8 -- Style Guide for Python Code

Yes

```
spam(ham[1], {eggs: 2})
dct['key'] = lst[index]
i = i + 1
submitted += 1
```

No

```
spam( ham[ 1 ], { eggs: 2 } )
dct ['key'] = lst [index]
i=i+1
submitted +=1
```

PEP8(2/2)

PEP 8 -- Style Guide for Python Code

Yes

```
def complex(real, imag=0.0):
    return magic(r=real, i=imag)
```

No

```
def complex(real, imag = 0.0):
    return magic(r = real, i = imag)
```

pycodestyle

- PEP8対応のチェックツール
- インストールが必要

pycodestyle's documentation

```
$ pip install pycodestyle  # インストール
$ pycodestyle fizzbuzz.py  # pycodestyleを実行
```

だめなfizzbuzz.py

```
for num in range(1, 100) :
    if num % 15 == 0:
        print( 'FizzBuzz' )
    elif num%3 == 0:
        print('Fizz')
    elif num % 5 == 0:
        print('Buzz')
```

実行結果

```
$ pycodestyle fizzbuzz.py
fizzbuzz.py:1:25: E203 whitespace before ':'
fizzbuzz.py:3:15: E201 whitespace after '('
fizzbuzz.py:3:26: E202 whitespace before ')'
fizzbuzz.py:4:13: E228 missing whitespace around modulo operato
fizzbuzz.py:7:10: E111 indentation is not a multiple of four
```

flake8

- pycodestylesに加えてpyflakesのチェックが入る
  - importしてるけど使ってない
  - 宣言した変数を使ってない
  - `from module import *` している
- インストールが必要

```
$ pip install flake8
$ flake8 fizzbuzz.py
```

Flake8: Your Tool For Style Guide Enforcement

まとめ

- flake8 を使おう
- エディターでもチェックしてくれるよ

4. 自動テスト手法(unittest)

なぜ自動テスト?

- 関数単位で仕様が明確になる
- リファクタリングしても安心

unittest

- Pythonの標準ライブラリ
- テストコードを書いて実行できる

26.4. unittest - ユニットテストフレームワーク

```
def fizzbuzz(num):
    if num % 3 == 0:
        ret = 'Fizz'
    elif num % 5 == 0:
        ret = 'Buzz'
    elif num % 15 == 0:   # ここが間違い
        ret = 'FizzBuzz'
```

間違ったfizzbuzz関数

```python
import unittest
from fizzbuzz_ng import fizzbuzz

class TestFizzbuzz(unittest.TestCase):
    def test_num(self):  # 普通の数字を返す
        self.assertEqual(fizzbuzz(7), '7')
    def test_fizz(self):  # 3の倍数
```

テストコード

実行結果

Traceback (most recent call last): File "/Users/takanori/unittest/test_fizzbuzz.py", line 12, in test_fizzbuzz self.assertEqual(fizzbuzz(30), 'FizzBuzz') AssertionError: 'Fizz' != 'FizzBuzz'

- Fizz
- FizzBuzz

Ran 4 tests in 0.001s

FAILED (failures=1)

修正して再実行

```
Ran 4 tests in 0.000s
OK
```

自動テスト: まとめ

- 大規模ならunittestを書こう
- 26.3. doctest - 対話的な実行例をテストするもあるよ

5. Python公式ドキュメントを読みこなそう

公式ドキュメント?

- https://docs.python.jp/3/ で公開されている
- 情報がたくさん
  - チュートリアル
  - ライブラリーリファレンス
  - HOWTO
- https://docs.python.org/ja/3/ に移行予定

非公式はあくまでヒントに

- Qiita
- teratail, Stack Overflow
- 個人ブログ

print()関数

- `print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)`
- objectsをsepで区切りながらテキストストリームfileに表示し、最後にendを表示します。
- sep、end、file、flush を与える場合、キーワード引数として与える必要があります。

print()関数 - 2. 組み込み関数

print()関数は可変長引数を受け取る

```
>>> print()

>>> print(1, 2)
1 2
>>> print(1, 2, 3, 4)
1 2 3 4
```

sep引数で区切り文字が変更できる

```
>>> print(1, 2, 3, 4, sep=",")
1,2,3,4
>>> print(1, 2, 3, 4, sep="|")
1|2|3|4
>>> print(1, 2, 3, 4, sep="✌")
1✌2✌3✌4
```

file引数で出力先を変えられる

```
>>> with open('sample.txt', 'w') as f:
...     print(1, 2, 3, 4, file=f)
...     print(1, 2, 3, 4, file=f, sep=',')
...
>>> with open('sample.txt') as f:
...     f.read()
...
```

ここまで触れたドキュメント

- pdb - Python デバッガ
- unittest - ユニットテストフレームワーク
- doctest - 対話的な実行例をテストする
- PEP 8 -- Style Guide for Python Code
- pycodestyle's documentation
- Flake8: Your Tool For Style Guide Enforcement
- print()関数 - 2. 組み込み関数

まとめ

- 公式ドキュメント読もう
- 外部ライブラリもドキュメントがちゃんとしているものを使おう

6. 文字列メソッドを使いこなそう

なぜ文字列メソッド?

- Pythonの文字列はメソッドが豊富
- 正規表現使わなくてもいろいろできる

4.7. テキストシーケンス型 - str

.format()メソッド使いこなし(1/2)

```
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')  # 順番入れ替え
'c, b, a'
>>> '{lat}, {lng}'.format(lat='37.24N', lng='-115.81W')  # 名前
'37.24N, -115.81W'
>>> '{:,}'.format(1234567890)  # 数値にカンマ
```

6.1.3.1. 書式指定ミニ言語仕様

.format()メソッド使いこなし(2/2)

```
>>> c = 3-5j  # 虚数を定義
>>> '{0.real} {0.imag}'.format(c)  # 属性にアクセス
'3.0 -5.0'
>>> from datetime import datetime
>>> now = datetime.now()
>>> '{:%Y-%m-%d %H:%M:%S}'.format(now)  # 特殊な書式指定
'2017-09-20 17:14:50'
```

6.1.3.1. 書式指定ミニ言語仕様

.endswith()メソッド

- str.endswith(suffix[, start[, end]])
- 文字列が指定されたsuffixで終わるならTrueを、そうでなければFalseを返します。suffixは見つけたい複数の接尾語のタプルでも構いません。

```
>>> filename = 'hoge.jpg'
>>> filename.endswith('.gif')
False
>>> filename.endswith(('.jpg', '.gif', '.png'))
True
```

まとめ

- 文字列メソッドは多機能
- 単純な処理は文字列メソッドのみでOK

4.7. テキストシーケンス型 - str

7. 例外処理の使いどころ

なぜ例外処理?

- エラーは情報の宝庫
- 例外処理は便利
- でもなんでも例外で処理するのは危険

Tracebackを読もう

```
Traceback (most recent call last):
  File "traceback_sample.py", line 7, in <module>
    main()
  File "traceback_sample.py", line 5, in main
    return sub()
  File "traceback_sample.py", line 2, in sub
    return int('名前')
```

- どんなエラーか
- エラーの理由
- どこで発生したか

例外を握りつぶしちゃう

```
try:
    # なんか
    # 処理
except Exception:
    pass
```

どこで発生する例外かわかりにくい

```
try:
    # とっても
    # とっても
    # 長い
    # 処理
except ValueError:
    # 対応する例外処理
```

例外の発生する個所のみを対象に

```
try:
    # とっても
except ValueError:
    # 対応する例外処理
    return
try:
    # とっても
```

可能なら前もってチェックする

- 例外処理

```
try:
    num = int(num_str)
except ValueError:
    # 数値の文字列じゃないときの処理
```

- 事前にチェック

```
if num_str.isdigit():
    num = int(num_str)
else:
    # 数値の文字列じゃないときの処理
```

まとめ

- Tracebackを読もう
- 例外を握りつぶさない
- 全体を大きく囲まない
- 先にわかるものはそこでチェック

5. 組み込み例外 — Python 3.6.1 ドキュメント

8. 内包表記、ジェネレーター式を使いこなそう

なぜ内包表記?

- 簡潔に書ける場合がある
- わかりやすい
- 複雑なのものは危険

0から9の平方リスト

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

0から9の平方リスト

```
>>> squares = [x**2 for x in range(10)]
```

0から9の偶数の平方リスト

```
>>> squares = []
>>> for x in range(10):
...     if x % 2 == 0:
...         squares.append(x**2)
...
>>> squares
[0, 4, 16, 36, 64]
```

0から9の偶数の平方リスト

```
>>> squares = [x**2 for x in range(10) if x % 2 == 0]
```

ジェネレーター式

- リスト内包表記の外側の [] を () にする
- リスト内包表記→全データできちゃう
- ジェネレーター式→1つずつ値をとりだす

```
>>> l = [x*2 for x in range(10)]
>>> l
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
>>> g = (x*2 for x in range(10))
>>> g
<generator object <genexpr> at 0x1020ad0f8>
>>> next(g)
```

内包表記

- set内包表記
- 辞書内包表記

```
>>> {x**2 for x in range(10)}
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
>>> {x: x**2 for x in range(10)}
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9:
```

まとめ

- 内包表記使おう
- ジェネレータ式使おう
- やりすぎ注意

5.1.3. リストの内包表記 - Pythonチュートリアル

9. 次のステップ

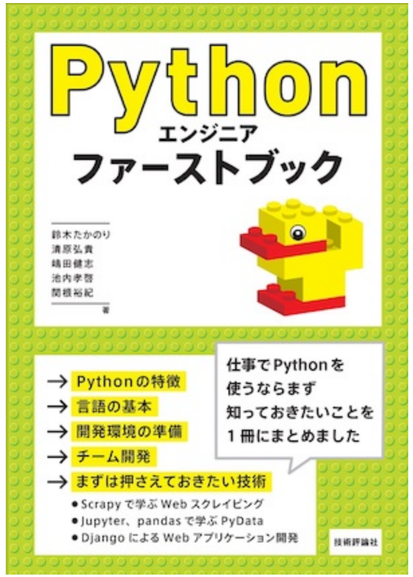
9. 次のステップ

または宣伝タイム

Python言語入門から

いちばんやさしいPythonの教本

Pythonで何か作ってみたい



Pythonエンジニア ファーストブック

ライブラリを使いこなしたい

Python ライブラリ厳選レシピ

PyQ

- オンライン学習サービス

PyQ - 本気でプログラミングを学びたい人のPythonオンライン学習サービス

コミュニティに参加しよう

- PyCon JP
- Python Boot Campで全国にPythonの環を広げよう！ #pycamp
- カテゴリ「Python」のグループ - connpass

One more thing

懇親会で書籍プレゼント

今日のゴール

(ほぼ)標準のPythonのみを使って、
いい感じにプログラムを書けるようになる
ヒントを得る
→できることろからやってみよう

ありがとうございました

Question?

Thank you