

引き継ぎ資料 Vol.5

Pythonのライブラリ等

2016/09/??

コンセプト

“私はこんなライブラリ・ツールを使っていますよ”

要は布教

今さら説明しないやつ

numpy 数値計算

scipy (numpyにはない)数値計算

matplotlib グラフ作成

目次

1. 導入
2. multiprocessing
3. docopt
4. sphinx
5. まとめ

目次

1. 導入

2. multiprocessing

3. docopt

4. sphinx

5. まとめ

Pythonには大体なんでもあるよ!

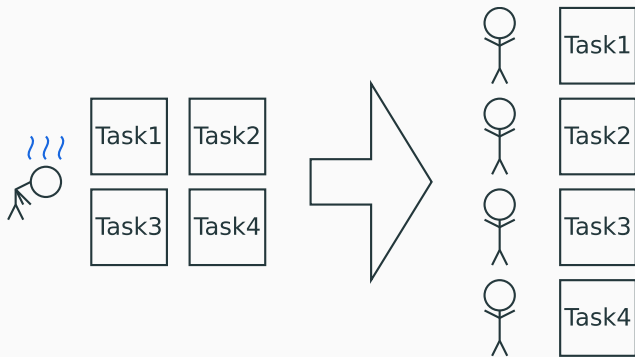
すでにあるライブラリを上手に使おう!

目次

1. 導入
2. multiprocessing
3. docopt
4. sphinx
5. まとめ

multiprocessing

作業者を増やして高速化





Q. Macのなかには作業者が何人？



Q. Macのなかには作業者が何人？ **A.** 8人(環境によるが目安として)



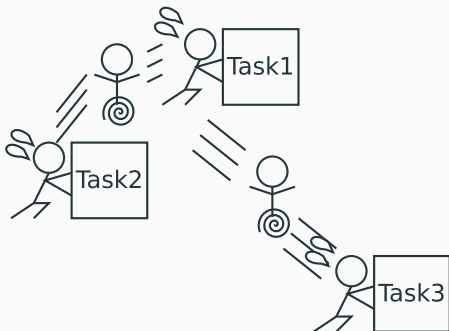
8倍速で実行可能

いわゆるプログラミング用語

作業者を作る = スレッド(Thread)を立てる

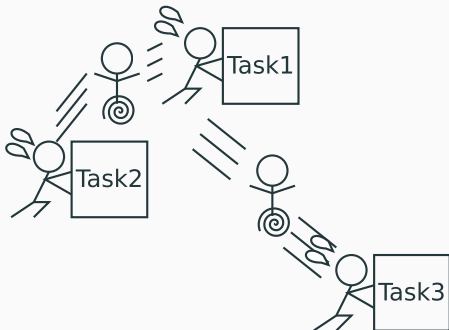
Pythonのスレッド

Pythonのスレッドは一人で幾つかの作業



Pythonのスレッド

Pythonのスレッドは一人で幾つかの作業



いくつ立てても高速化はされない

そこで登場!

multiprocessingはProcess操作を提供

Process = 作業者


```
from multiprocessing import Process

def single_process():
    for _ in range(5):
        hard_task()

def multi_process():
    ps = []
    for _ in range(5):
        ps.append(Process(target=hard_task))
        ps[-1].start()
    for p in ps:
        p.join()
```

実行時間

約3倍の実行速度

関数名	実行時間[秒]
single_process	6.62
multi_process	2.02

注意点

- ・ 5プロセスで実行しているが5倍にはならない
 - ・ プロセスの生成はそれなりに大変
- ・ なんでもかんでも別プロセスにしない!
 - ・ デバッグがし難い
 - ・ ここぞという時の切り札

目次

1. 導入
2. multiprocessing
3. docopt
4. sphinx
5. まとめ

コマンドラインオプションを扱うライブラリ
Python標準のargparseより簡単かつカッコいい

インストール方法

普通にpipで導入

```
$ pip install docopt
```

コメントからコマンドラインオプションを指定

```
""" docopt sample
```

Usage:

```
example_docopt.py [-ab] <arg1> <arg2>  
example_docopt.py -h
```

Options:

```
-a, --arg1  Argument1  
-b, --ball  Ball
```

```
"""
```

```
from docopt import docopt  
if __name__ == "__main__":  
    print(docopt(__doc__))
```


Usage:

```
example_docopt.py [-ab] <arg1> <arg2>  
example_docopt.py -h
```

実行

```
$ python example_docopt.py hoge fuga  
{'--arg1 ': False ,  
 '--ball ': False ,  
 '-h': False ,  
 '<arg1 > ': 'hoge' ,  
 '<arg2 > ': 'fuga' }
```

Usage:

```
example_docopt.py [-ab] <arg1> <arg2>  
example_docopt.py -h
```

実行

```
$ python example_docopt.py -a hoge fuga  
{'--arg1 ': True,  
'--ball ': False,  
'-h ': False,  
'<arg1 > ': 'hoge',  
'<arg2 > ': 'fuga'}
```

Pros.

- ・ 書くのが楽
- ・ コメントとプログラムが必ず一致

Cons.

- ・ 重い
- ・ <http://docopt.org/>をよく見ないとたまに変な動作

ソースコードの中にファイル名とか入れるのやめよう

ファイルの位置が変わったらソースも書き換え

コマンドラインオプションにしておけば呼び出し時に自由自在

目次

1. 導入
2. multiprocessing
3. docopt
4. sphinx
5. まとめ

Pythonに関するツール¹

Anacondaにはデフォルトで搭載

¹sphinxはPython限定ではない

1. 一定の方式に則ってコメントを記述
2. Sphinxで処理
3. プログラムのドキュメント完成

例: コメント記述

Sphinxでは決まったルールでコメントを書くことが必要
ルール

- ・ Sphinx形式(デフォルト)
- ・ Google形式
http://www.sphinx-doc.org/en/stable/ext/example_google.html
- ・ NumPy形式
http://www.sphinx-doc.org/en/stable/ext/example_numpy.html

例: コメント記述(Google形式)

```
"""Example Program"""
```

```
def sample_function(arg):
```

```
    """Sample function
```

```
    long description of this function
```

```
    Args:
```

```
        arg (str): text
```

```
    Returns:
```

```
        str: printed text
```

```
    """
```

```
    print(arg)
```

```
    return arg
```

例: Sphinxで処理

ソースコードからドキュメントの元を生成

```
$ sphinx-apidoc -F -o doc src
```

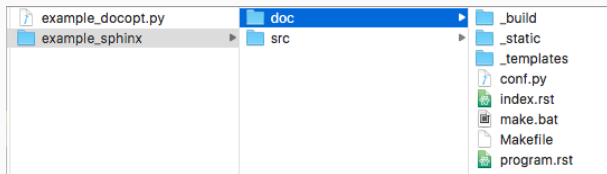
srcの中身を元にdocにドキュメントの元を生成

例: docフォルダの中身

***.rst** ドキュメントの元

Makefile ドキュメント生成のプログラム

conf.py 設定が書かれたpythonファイル



例: conf.py

設定が書かれたpython形式のファイル

記述が必要な事項

- ・ extensionsに"sphinxcontrib.napoleon"追加
Google形式コメントの有効化
- ・ languageを"ja"に設定
自動生成される部分を日本語化
- ・ テーマの設定

詳細はサンプルコードを参照

例: ドキュメント完成

ドキュメントディレクトリへ移動

```
$ cd doc
```

htmlファイルの生成

```
$ make html
```

pdfファイルの生成

```
$ make latexpdfja
```

例:結果(html)



[Docs](#) » [program module](#)

[View page source](#)

program module

Example Program

```
program.sample_function(arg)
```

[\[ソース\]](#)

Sample function

long description of this function

パラメータ: *arg (str)* – text

戻り値: printed text

戻り値の型: str

[⏪ Previous](#)

© Copyright 2016, Author.

Built with [Sphinx](#) using a [theme](#) provided by [Read the Docs](#).

なぜ使うのか？

- ・ ドキュメントはプログラムの理解に不可欠
- ・ きちんとコメントを書こう
- ・ きちんと綺麗なコメントを書こう
- ・ ルールに則ったコメントを書こう

目次

1. 導入
2. multiprocessing
3. docopt
4. sphinx
5. まとめ

Pythonにはライブラリがたくさん
きちんと使って楽をしよう

Questions?