python_kobe Documentation

Release 1.0

akira

June 13, 2015

CONTENTS

Contents:

CHAPTER

ONE

GIT の使い方について

GitHub で pull したり，branch を作ってそれをローカルで再現する方法がややこしいのでまとめのメモを残しておきます.

そもそも，kenjisato さんがこの作業の流れは書いてくださっているので，そちらを　参照 すると詳しいです.

ですが，このセクションでは，自分の確認も兼ねて，もう一度流れを確認します.

わかりやすいように，Sphinx の GitHub で作業を行っている前提でチュートリアルを書きます.

master branch とは，親ファイルのことです．この，master の内容を修正したり，改善したいときは別に，branch を作ってそこに内容を書き加えます．branch を作った時点では，master と同じ内容がコピーされます.

こうすることで，親ファイル (master) の内容をいじらずに，改善を加えた親のコピー (branch) を見せることができます.

branch をつくる方法は，:

```
$git branch branc のなまえ
```

で作ることができます．ただし，branch を作った時点では，その branch に移動しているわけではないので，注意が必要です.

最終的には，branch は破棄されるか，master にマージされるかするのですが．まずは，誰かが作った，branch の内容を自分のローカル環境で再現する方法を確認してみましょう.

- master 以外のブランチに移動して，その内容を再現する方法.

まず ‘git status‘でどこのブランチにいるか，そのブランチにおける（自分が行った）更新がどのようなものかの一覧を出して確認します.

これらの更新を保存したければ ‘git stash‘を使って，保存できます.

この作業は master の brach でなくても同じ．つまり．誰かの branch に移って再現するときも同じ操作をする.

次に，:

```
$checkout branch 名
```

で他の branch にうつる.

すると，ローカルでその branch を再現できるようになる．つまり，見かけはその branch の内容が上書きされたように見える.

あれこれ確認できたら，:

```
$stash
```

をして保存（必要があれば）して，master　branch に:

```
$checkout master
```

で戻る．

もし，作った branch をリモートブランチに push したければ:

```
$git push origin branch の名前
```

で push する．Terminal(コマンド・プロンプト) で push した場合でも，GitHub のユーザー名とパスワードを入力を要求される場合があるので，従う．

CHAPTER

TWO

PYTHON を勉強する周辺環境

python の勉強会をするにあたって，周辺環境についても指導を頂きました．とりあえず，書き並べて，それぞれ別の章で詳しい説明を行う予定です．

Sphinx

Python で書かれたテキストエディタです．.rst ファイルで，テキストを書くと，latex や html でコンパイルしてくれる優れものです．

.rst ファイルの記法は reST(reStructuredText) 記法に基いています．詳しくは，**公式の reStructuredText 入門_** か，Sphinx でドキュメントを書くため reST 記法に入門した を参照するといいと思います．

（要議論）章立てについて

テキストエディタについて

- sublime text

を薦めて頂きました．使いやすそうです．今後，sphinx は sublime text で編集してみようと思います．Terminal から起動するには，path を通す必要があるようです．(僕の Mac では，subl がコマンドに割り当てられています．)

Git Hub

GitHub とはバージョン管理システムである，Git を使うサイトです．GitHub を使うことで，プログラムソースのバージョン管理が容易になります．

共同研究やプロジェクトを行うには，今後，GitHub を使えるようになることが必須だろうと予測されるので，是非習熟したいです．

この勉強会の教科書になっている，Quantitive Economic の Solution や，Library をつくるプロジェクトも GitHub で公開 されています．

そして，そもそもこの勉強会の資料なども，Github で 公開してます .

Jupyter ノートブックを RST に変換するプログラム

kenjisato さんが，Jupyter ノートブックを RST に変換するスクリプト notebookconvert.py を作りました．

Makefile のあるディレクトリで:

```
$ python notebookconvert.py
$ make html
```

として出力すれば自動的に変換されます．

CHAPTER

THREE

これまでの取り決めの流れ

普通に,Quantitive_Economic_ の教材を薦めて行く予定だったのですが，プログラミングを学ぶ上で Git といった周辺環境を構築したり，基礎知識を学ぶ必要性を認識しました.

具体的な勉強会の取り決めは以下の通りです.

- プログラムの資料は基本的に Jupyter Notebook(旧 python notebook) を使う
- 勉強会後に，新たに分かったことを書き加えた講義ノートを Sphinx で作成

  この理由として

  1. jupyter Notebook のような形式ではなく．講義資料としてまとまったものを作成したいという要請
  2. Github を使う練習としては，Sphinx のほうが Jupiter notebook よりも適切であること

  の 2 点の理由が挙げられます.

-

CHAPTER

FOUR

QUANTITATIVE ECONOMICS WITH PYTHON

QUANTITATIVE ECONOMICS with Python　に基いて行った，勉強会の講義ノートです.
勉強会当日は Jupiter Notebook で資料を作成します.
ですが，勉強会中に気付いたことや，もらった有益アドバイスなどをまとめるためにノートを作成します.
勉強会は教科書にしたがって進みますが，講義資料は随時更新されていきます.

CHAPTER

FIVE

PYTHON ESSENTIALS(P48)

Python_Essentials

5.1 Data Type(p49)

Data_Type では，Python でよく使われるデータの種類について学びます.

5.1.1 Primitive Data Types

Bool Type

最も基本的なデータタイプは Bool 型です.

これは，変数自体に True や False が入ります．:

```
x = True
```

という感じです.

また，命題についても，True or False に変換されて変数に入ります．:

```
y = 100<10
```

のようにすると，y には False が入ります.

Data type の確認方法

プログラムを書いていて，ある変数がどのようなデータタイプなのか, わからなくなることがあります.

これ以降にも，様々なデータタイプが出てきますが，先にデータタイプの確認方法を学びましょう.

ある，変数 x がどのような データタイプかを調べるためには:

```
type(x)
```

とします.

もし，x が Bool type ならば:

```
bool
```

と出力されます.

Bool Type の性質

Bool Type の変数には True もしくは False が割り当てられますが，True と False にはそれぞれ，1 と 2 という数字も割り当てられています.

もし x が True，y が False ならば:

```
x+Y

1
```

となります.

List と Bool

この，Bool Type は List の要素にもなります.

List とは

List は様々なオブジェクトを格納できる列のことです.

List には，値や文字なども入れることができます．例えば，:

```
bools = [True, True, False, True]
```

という感じです．先ほどの，True や False に 1 と 0 が割り当てられることを考えると，:

```
sum(bools)

3
```

となります．この sum() のような命令のことを関数と呼びます.

特に，この sum() は組み込み関数とよばれ，python にもともと入っています.

ほかの組み込み関数については，公式の組み込み関数 を参照してください.

そのほかのデータータイプ

Python には Bool Type 以外にのデータタイプも存在します.

例えば数字の，integers と floats の 2 つの種類のデータタイプがあります．:

```
a, b = 1, 2
type(a)

int
```

が interger Type であり，:

```
c, d = 2.5, 10.0
type(c)

float
```

となります．この int と float については後に詳しい説明をします.

注意

この，integer に関連する問題を一つ見てみましょう．

Python 2x では， ２つの integer(整数) 同士の割り算では，integer の部分だけを返します．

つまり，:

```
1/2

0
```

となります．

ただし，integer と float や，float と float 同士の割り算では，少数以下も返されます．:

```
1.0/2.0

0.5
```

ですし，:

```
1.0/2

0.5
```

となります．

また，このような問題は Python 3x では発生しません．

しかし，この教科書は Pyhton 2x を用いるので，読者はこのような問題に留意する必要があるでしょう．

complex Type

複素数も，Python における Primitive なデータタイプの一つです．

Python では，Complex Type と呼ばれます．

Python で複素数を表現するには，組み込み関数の complex() を使います．:

```
complex(実部, 虚部)
```

のように指定します．また，Python では複素数は j で表現されます．:

```
x = complex(1, 2)
y = complex(2, 1)
```

とすれば，:

```
x*y

5j
```

となります．

Containers p50

Python には様々なコンテナが存在します．コンテナは，データーを集めておくために使われます．

例えば，先に説明した，list は組み込みコンテナといって，Python にもともと備わっています．

list と同じような，組み込みコンテナとして tuple（トゥープル，タプル）があります．

この，tuple と list の大きな違いの一つに，tupule が immutable であることが挙げられます.

“ tuple が immutable ”とは，tuple の値が変更できないことを意味します.

一方で,“ list は mutable ”なので，値を変更することができます.

以下に例を示します，まず list は mutable すなわち持っている変数の数が増えたり，減ったり変わったりします.：

```
x = [1, 2]
```

という list を考え，この 1 行目の，1 を変化させてみましょう.

ところで，この行番号ですが，python では，0 から数えます．x[0] というようにすると，list である x の 0 行目を指定できます．これを変更するには,:

```
x[0]=10
```

というようにします．確認すると，:

```
x

[10, 2]
```

というように，変更されていることがわかります.

このように，list は mutabl です．しかし，一方で，tupule は immutable です．:

```
X = (1, 2)
```

にたいして，X[0] とすると，:

```
X[0] = 10

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-531149b57146> in <module>()
----> 1 X[0] = 10

TypeError: 'tuple' object does not support item assignment
```

となってしまいます

もう少し，mutable と immutable の話をしよう

mutable な list にも，immutable なリストにも，”unpacked” という操作を施すことができます.

unpacked では，それぞれの行を指定した変数に当てはめることができます．:

```
integers = (10, 20, 30)
x, y, z = integers
```

とすると:

```
x
10

y
20

z
30
```

というように割り当てられます.

また，slice notetion という操作も，mutable,immutable のどちらにも施すことができます.

例えば，:

```
a = [2, 4 , 6, 8]
```

という，list の 1 行目から，最後の行までを抜き出したいときは，:

```
a[1:]
[4, 6, 8]
```

と指定します.

また，ある行から，ある行までを抜き出したいとき，例えば，1-2 行目を抜き出したいとき，:

```
a[1:3]
[4, 6]
```

というような指定の仕方をします．:

```
list[抜き出しを開始する行番号:抜き出しを終わる行番号＋ 1]
```

という感じです.

また，:

```
a[-2:]
[6, 8]
```

というようにすれば，最後の 2 行を抜き出すことができます.

以上の一連の操作は，文字に対しても行えて，:

```
s = 'kobe univ.'

s[-5:]
'univ.'
```

と抜き出せることができます.

このような，最後の数行を抜き出すという操作は，全体を確認するには長すぎるデータの内容を確認するときに，有効な場合があります.

Sets と Dictionaries

先に，list と tupul という二種類の container を紹介しました.

次に，set と dictionary という 2 つの container について説明します.