目次

[Git 4](¥l)

[Commit messageエディタの変更 6](¥l)

[名前とメールアドレスの設定 7](¥l)

[コミットの取り消し 7](¥l)

[作業ディレクトリ/ステージング/Gitディレクトリ 8](¥l)

[git add 8](¥l)

[git commit --amend : 直前のコミットを修正する 9](¥l)

[git branch 9](¥l)

[Merge 10](¥l)

[意図せぬマージ発生 10](¥l)

[Conflict 12](¥l)

[Diff 12](¥l)

[git log 12](¥l)

[git pull : Fetch from and integrate with another repository 15](¥l)

[master と origin/master 15](¥l)

[.gitignoreファイルの作成 16](¥l)

[Sourcetree 16](¥l)

[Instrall Sourcetree 16](¥l)

[SourcetreeでのSSH認証方法 20](¥l)

[ファイルをstagingにする方法 22](¥l)

[gitのフローモデル 26](¥l)

[Github 27](¥l)

[Setting 28](¥l)

[How to use Github 28](¥l)

[Markdown記法 28](¥l)

[Wiki / Issue / README 29](¥l)

[Raw / Blame/ History 30](¥l)

[Label, Issue, Milestones 31](¥l)

[Gist 32](¥l)

[フォーク 32](¥l)

[Help 32](¥l)

[Python 33](¥l)

[事例紹介 33](¥l)

[事例１ : 東京の最高気温データ 33](¥l)

[事例2 : 東日本大震災の地震データ 37](¥l)

[事例３ Google検索 38](¥l)

[事例４：Webクローラ入門 39](¥l)

[urllib 40](¥l)

[正規表現 40](¥l)

[JSON 40](¥l)

[dir() 40](¥l)

[help() : 40](¥l)

[Python Library 41](¥l)

[Pandas 41](¥l)

[Atom 41](¥l)

[How to use Atom 41](¥l)

[Open file 41](¥l)

[Python Program 42](¥l)

[Google search 42](¥l)

[PyCon 2016 42](¥l)

[Pycon 2017 42](¥l)

[Jupyter Notebook 42](¥l)

[Jupyterとは 43](¥l)

[Jupiterの開始方法 43](¥l)

[編集方法 43](¥l)

[magic command: % 44](¥l)

[Jupiterの終了方法 44](¥l)

[Mac Operation 44](¥l)

[Screenshot 44](¥l)

[Paint 44](¥l)

[Mac標準装備の「プレビュー」は、描画ソフトでもある 45](¥l)

[**「プレビュー」の作図機能** 45](¥l)

[Finder (Explorer) 46](¥l)

[簡単！MacのFinderを使いやすくする１０のカスタマイズ 46](¥l)

[Macの仕様とか、設定 46](¥l)

[クリップボードについて 46](¥l)

[Basemapインストール方法 46](¥l)

[参考文献 47](¥l)

[GitHubの教科書 47](¥l)

# Git

|  |
| --- |
| とりあえず、このファイルをGITの作業フォルダ（masterと呼ぶんだっけ？）においてみた。  ファイル一覧を見ると、”Github + Python.docx”が新規に追加されているのは期待通り。  しかし、”.DS\_Store”って見に覚えがない。何だこれ？    ネットで調べたところ、メタデータ管理ファイルらしい。気にしなくて良いらしい。    さて、GITでは、当然、”Github + Python.docx”だけを追加したい。確か、git addコマンドを使ってステージに乗せる（ステージング？）させるんだっけ？    これで、問題なくステージング出来た。    その後、またWORDファイルを編集していたので、git status コマンドを実行すると今度は、WORDファイルがmodifiedとなってしまった。    git log で今の経緯を見てみた。    どうやら、git addしたログは出てこないらしい。  とりあえず、git commit をしてみた。    次は、これをgithubにあげてみよう。(2018/01/07) |
| Commit messageエディタの変更 コミットメッセージを編集するエディタがviだと使いにくい。これは自由に変更出来る。下記のように、core.editorを変更すれば良いらしい。    ところが、これだとエラーが発生した。どうやら、gitとエディタの連携が上手くいかないらしい。エディタが開きコミットメッセージ編集をする前に、gitがコミットメッセージがないと判断してしまい、エラーを吐いているようだ。というわけで、gitに編集待ちさせなくてはいけないようだ。  色々と調べたところ、 “ -w”　という風に、-wオプションを付けると上手くいくらしいことがわかった。    やってみると、果たして、問題なくコミット出来た。これは良い。  コミットメッセージ  <https://qiita.com/ma_me/items/f944101c97a4d0e4a842>  <http://www.devlog.alt-area.org/?p=1366> |
| 名前とメールアドレスの設定 gitでコミットする前に、名前とメールアドレスを適切に設定しておいた方が良い。  下記のようなコマンド一度実行すれば良い。   |  | | --- | | $ git config --global user.name "John Doe"  $ git config --global user.email johndoe@example.com |   <https://git-scm.com/book/ja/v1/%E4%BD%BF%E3%81%84%E5%A7%8B%E3%82%81%E3%82%8B-%E6%9C%80%E5%88%9D%E3%81%AEGit%E3%81%AE%E6%A7%8B%E6%88%90> |
| コミットの取り消し コミットの方法を間違えたり、コミットした後に編集したりして、コミットを取り消したい時には、下記のコマンドを打てば良いらしい。   |  | | --- | | git reset --hard HEAD^ |     <https://qiita.com/shuntaro_tamura/items/06281261d893acf049ed> |
|  |
|  |
|  |

## 作業ディレクトリ/ステージング/Gitディレクトリ

|  |
| --- |
| gitでは、下記３つの状態が存在する。まずはこれを理解しないといけない。  1.作業ディレクトリ  2.ステージング・エリア  3.Gitディレクトリ (レポジトリ)  作業ディレクトリで変更ファイルをステージにあげてから、レポジトリにコミットします。  ファイルを更新するには、作業ディレクトリでファイルを更新した後、更新したいファイルをステージングにあげて、Gitディレクトリの状態にする。  <https://qiita.com/hshimo/items/ab91b99cd61724127aa7> |
|  |
|  |
|  |
|  |

## git add

|  |
| --- |
| git addコマンドでファイルを作業ディレクトリからステージングに移行出来る。  git addにも色々なオプションがある。そのうち勉強しよう。  「git add . ワーキングツリーに新規作成されたり変更されたファイルをインデックスに上げる  git add -u 一つ前と最新の変更があった部分だけをインデックスに上げる。新しく作成したファイルはインデックスには上がらない。  git add -A 新規作成、修正、削除されたファイルを全てインデックスに上げる  git add filepattern ファイル名を直接指定したり、\*.rbのようにワイルドカードを使って該当するファイルを全てインデックスに上げることができる。  git add -p 多く編集したけど、コミットでは分けたいというときにおすすめ。」  <https://qiita.com/sseo/items/100ed47acfa19c2821d6> |
|  |
|  |

## git commit --amend : 直前のコミットを修正する

|  |
| --- |
| 「git commit –amend  直前のコミットメッセージを修正できる。  このコマンドを実行すると、エディタが立ち上がるのでよしなに修正してエディタを終了する。  git add → git commit --amend  もし、一つ前のコミットに含めるべきファイルが漏れていた場合。  git addで漏れたファイルを追加してからgit commit --amendを実行すると漏れたファイルも含めた状態で前のコミットのコミットメッセージを修正できる。  つまり、前のコミットを修正できるということ。ついでにコミットメッセージも修正できちゃう。」  <https://qiita.com/sseo/items/100ed47acfa19c2821d6> |
|  |
|  |
|  |

## git branch

|  |
| --- |
| ブランチを作成する時に使う。  git branch develop を実行すると、developというブランチが作成される。  　git branchを実行すると、２つのブランチが存在することが確認出来る。 \*(アスタリスク)が付いている方が、現在の作業対象になっているらしい。  　・・・ちょっと何を言っているのかわからない。ちゃんと調べる必要がある。 |
| git branch –D <ブランチ名>  不要になったブランチを削除する時に使う  (例)不要になった developブランチを削除してみた。 |
|  |
|  |

## Merge

|  |
| --- |
| 意図せぬマージ発生 remote serverに、変更したWORDファイルをpushしようとしたら、rejectされてしまった。どうやら、remote serverの方がlocalより新しい状態だからダメということらしい。  そういえば、README.md という説明をGithubにて編集していた。    そこで、git pull を実行してみた。README.mdファイルをサーバーから取得出来た。ちなみに、ここで気になるメッセージがあった。Merge。どうやら、これがMergeというやつらしい。    git pullした後で、git push origin masterを実行したら、上手くいった。そこで、git logを見ると、Marge したというコミットメッセージが生成されていた。    git merge –abort  コンフリクトが解消されない場合、マージ開始前の状態に戻してくれる。 |
|  |
|  |
|  |

## Conflict

## Diff

|  |
| --- |
| Git diffコマンドを実行すると、最後にコミットした後の変更内容がdiffされる。  ただし、試したところ、git addコマンドでステージングされるファイルは、diffされないみたい。使い所がいまいちわからない。 |
|  |
|  |

## git log

色々なオプションがあるので、知っておくと便利に使えそう。

|  |
| --- |
| git log |
| –n :表示するコミット数の指定 |
| –p : 任意のファイルの変更履歴を表示する |
| --stat : diffstatを生成する    diffstatの仕様も調べてみた(man diff)。どうやら、diffの結果を受け取って、それをグラフにするというコマンドだそうだ。 |
| git log --name-status : 変更したファイルを表示  <https://qiita.com/take4s5i/items/15d8648405f4e7ea3039> |
| git log --all --graph --pretty=oneline  git log をグラフ化してくれる。  --pretty の形式はいくつかある。Oneline, short, medium, full, fullerなど。  o oneline  <sha1> <title line>  This is designed to be as compact as possible.  o short  commit <sha1>  Author: <author>  <title line>  o medium  commit <sha1>  Author: <author>  Date: <author date>  <title line>  <full commit message>  o full  commit <sha1>  Author: <author>  Commit: <committer>  <title line>  <full commit message>    o fuller  commit <sha1>  Author: <author>  AuthorDate: <author date>  Commit: <committer>  CommitDate: <committer date>  <title line>  <full commit message> |

## git pull : Fetch from and integrate with another repository

|  |
| --- |
| git pull = git fetch + git merge origin/master  <https://qiita.com/osamu1203/items/cb94ef9da02e1ec3e921>  <https://qiita.com/takakuda/items/2123e37733445f69f0ff> |
|  |
|  |
|  |

## master と origin/master

masterはローカルリポジトリの作業ディレクトリの状態を指している。

一方、ogirin/masterはリモートリポジトリの状態を指している。Mergeする時は、まず、git fetch origin/mster で origin/masterを最新にした上で、git mergeをする。それらを一度に行うのが、git merge

## .gitignoreファイルの作成

|  |
| --- |
| .DS\_Storeファイルみたいに、OSが勝手に生成するファイルはリポジトリに含めない。  それらのファイルを記述するのが .gitignoreファイル。  gitディレクトリのトップに作っておくと除外される。エディタで生成すれば良い。 |

## Sourcetree

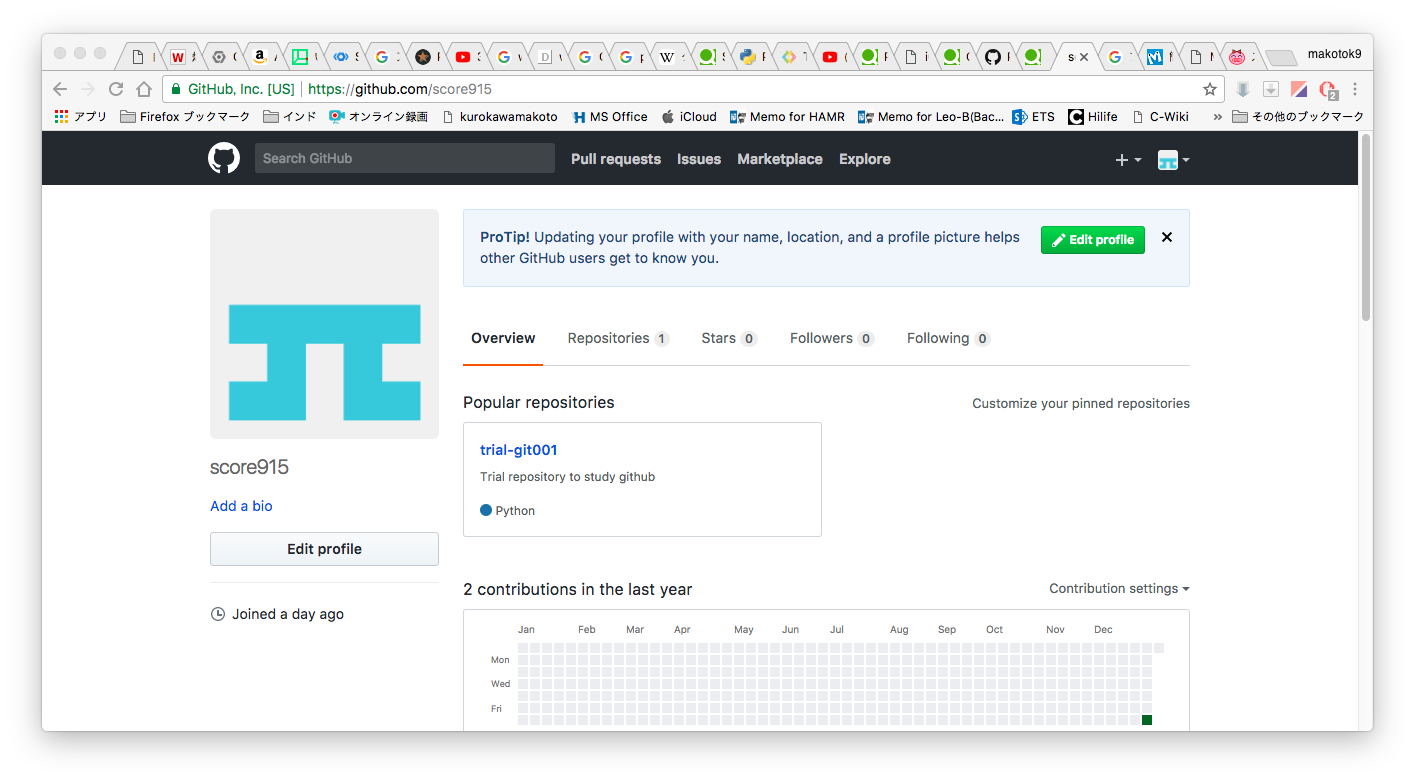
|  |
| --- |
| Sourcetreeとは、アトラシアン社が無償で提供する、GUIツール。これを使えば、マウスでGitが利用出来るらしい。  <https://ja.atlassian.com/software/sourcetree>のページにアクセスしてみると、確かに見やすそうな画面が表示されている。 |

|  |
| --- |
| Instrall Sourcetree 早速、Sourcetreeをダウンロードして、インストール開始しようとしたところ、下記のようにアカウントを要求された。アカウントは持っていなかったが、「既存のアカウント」を選択したところ、Googleアカウントが選択出来たので、それを選択したら、自動的にAtlassianアカウントが生成された。（その際、組織名は、適当につけた。）    アカウントの作成が完了すると、今度はリモートサーバーのリポジトリをクローンする画面が表示された。リポジトリはGithubに置いているので、Githubを選択した。その後、わかりにくいのだが、「接続アカウント」のボタンを押すと、Githubへのログイン画面が表示され、ユーザー名とパスワードを入力すると、SSH public keyが取り込まれた。    次に、既存のリポジトリをクローンする画面が表示された。    クローンを押したところ、なぜか失敗した。さて、どうしよう？エラーメッセージを読むと、どうやら public keyを使った SSH認証が失敗しているらしい。      ~~ネットで検索したところ、~~[~~https://qiita.com/zakki/items/d1e2839b0165e0c0750c~~](https://qiita.com/zakki/items/d1e2839b0165e0c0750c)~~が見つかった、どうやら、SSHクライアントがPuTTYという設定になっているので、それをOpenSSHに変更しないといけないらしい。~~  しかし、どうやって？環境設定を調べても、それらしい項目が見つからない。  ネットで色々検索したところ、同じ症状で悩んでいる人が多いことはわかったが、残念ながらどの記事を試してもSSH認証が出来なかった。  仕方がないので、下記のようにプロトコルをHTTPS認証に変更したところ、あっさりと認証出来た。    ほどなく、クローンが完了して、下記のような画面が表示された。上手く行ったようだ。これを使いこなせば、gitをコマンドラインを使わずに操作出来るということなので、いずれ試してみよう。    <https://ja.atlassian.com/software/sourcetree>  はじめてのSourceTree（使い方編）<https://qiita.com/naoki85/items/c7660d70347e9e70b201> |
| SourcetreeでのSSH認証方法 　その後、色々と調べたら、SSH認証が出来るようになった。  やり方は下記の通り。  １．Sourcetreeの右側にある「端末」をクリックする。    ２．ls –al ~/.ssh/ と入力して、SSHキーがすでにあるかどうか確認する。なければSSHキーを作成すること。  ３．pbcopy < ~/.ssh/id\_rsa.pub と入力して、キーをクリップボードにコピーする。（Windowsは別の方法で）  ４．ssh –T [git@github.com](mailto:git@github.com) と入力して、githubにアクセスできるか確認する。ここでパスフレーズを要求されたら、正しく入力すること。  “You've successfully authenticated, but GitHub does not provide shell access.”と表示されたら成功。メッセージを読むとエラーに見えるのだが・・・。    ５．以上が終わったら、環境設定でプロトロルをSSHに変更すれば良い。    大事なことは、Sourcetreeの「端末」を使って認証することだったみたい？？ |
| ファイルをstagingにする方法 master , uncommitted changes を選択した上で、作業ツリーのファイルからステージングしたいファイルのチェックボックスを押すと、ステージングされる。  （例）~$thub + Python002.docx をステージングに移行させる   |  |  |  | | --- | --- | --- | |  |  |  | | 作業ディレクトリ |  |  | | ステージング後 |  |  |   Conflictした場合  コミットしたファイルをプッシュしようとして、コンフリクトすると、下記のような画面が表示される。  コンフリクトしたのは、README.mdというファイルのはずだが、メッセージの中にはファイル名は記述されないようだ。      基本に返って、プルしてみる。今度は、「マージで競合」というメッセージが表示された。ファイル名は表示されていないが、README.mdに「！」マークが付いている。これがコンフリクトしたという意味らしい。    ファイルを右クリック(Mac: Ctrl + スペース)すると、メニューが表示される。基本は開く。    デフォルト登録してあるエディタでファイルが開く。コンフリクト箇所が 下記のようにマーキングされているので、正しい記述に自分で修正する。（これがとてもむずかしい作業の場合、自分で考え込まずに、知っていそうな人に頼むのが良い）  <<<<  ====  >>>>    ファイルを修正した後、コミットしようとすると、メッセージがいつもと違う。Merge何とかと表示される。    コミットメッセージを変更せず、そのままコミット＋プッシュしたら成功した。 |
|  |

## gitのフローモデル

|  |
| --- |
| gitでは、ブランチが容易に作成される。そのため、ある程度管理しないと、ブランチ間の整合が取れなくなる。  そこでいくつかのモデルがある。有名なのがgit flowと githubフロー。    <https://qiita.com/KosukeSone/items/514dd24828b485c69a05>  <https://qiita.com/mint__/items/bfc58589b5b1e0a1856a> |
|  |
|  |
|  |

# Github

<https://github.com/score915>

|  |  |
| --- | --- |
| Gitbubには、すでに Macbook のSSH公開鍵を設定済みなので、今回はSSHでpushする。  今回は、修正されたファイル（WORD）のuploadだけだし、各種設定は終わっているので、下記のコマンドだけでいけるはず。   |  | | --- | | git push origin master |     Githubを見ると、確かに WORDファイルが追加されている。    次は、本題のPythonについて記述する。(2018/01/07) |
|  |
|  |
|  |
|  |

## Setting

## How to use Github

## Markdown記法

|  |
| --- |
| 「文書を記述するための軽量マークアップ言語のひとつである。「書きやすくて読みやすいプレーンテキストとして記述した文書を、妥当なXHTML(もしくはHTML)文書へと変換できるフォーマット」として、ジョン・グル―バー(John Gruber)とアーロン・スワーツ(Aaron Swartz)によって考案された。」  # :見出し1  ## : 見出し2  --- : 水平線  \* : リスト  <https://qiita.com/Blueman81/items/72ca43681d16d44e21ad>  <https://qiita.com/Thermidor/items/b684cfabe0f2fde0c67b>  <https://qiita.com/oreo/items/82183bfbaac69971917f> |
|  |
|  |
|  |

## Wiki / Issue / README

|  |
| --- |
| Wikiも書ける。 |
|  |
|  |

## Raw / Blame/ History

|  |
| --- |
| Githubの機能。ファイル毎に履歴等を表示出来る。  Raw: 生データを表示  Blame: 誰がいつ変更したか表示  History: 変更履歴 |
|  |
|  |
|  |

## Label, Issue, Milestones

|  |
| --- |
| Githubでは、Label, Issue, Milestonesを設定出来る。Issuesのタブから設定する。  。。。多分、私は当分使わないのでスキップする。 |
|  |
|  |

Githubのファイルサイズ制限

|  |
| --- |
| リポジトリは１GB以下、ファイルサイズは１００MB以下という制限があるらしい。多分、そこまでは使わないので問題ない。  <http://huguma.hatenablog.com/entry/2014/10/03/184232> |

## Gist

|  |
| --- |
| Gist |
|  |
|  |
|  |

## フォーク

|  |
| --- |
| フォーク(fork)とは、共用リモートリポジトリを、自分のリモートリポジトリにコピーすること。これにより、フォークされたリポジトリの所有者が自分になるため、編集可能になる。修正したリポジトリは、プルリクエストすることで反映させることも出来る。  <http://cuaoar.jp/2013/03/github-fork-pull-request.html> |
|  |
|  |

## Help

|  |
| --- |
| ? キーを押すと、ヘルプを呼び出すことが出来る。 |
|  |
|  |

# Bitbucket

Bitbucketも使い方をマスターしたいので、ここで色々と練習してみる。

## Setting

|  |
| --- |
| Bitbucketは、個人でも登録出来るらしい。まずは、登録してみる。  まずはこちらにアクセスする。  <https://bitbucket.org/>  下記、Webに書いてある通りにやれば、簡単に登録出来て、ホーム画面が表示された。    <https://shootingogi.net/9168> |
| Githubからリポジトリをコピーしてみる  試しに、githubのコードをコピーしてみたら、あっさり出来てしまった。ただし、この手順が正しいのかどうかはわからない。 |
| bitbucketからsourcetreeにcloneを作ってみる  下記、概要の”clone in sourcetree”をクリックしたら、「新規にクローン」ウィンドウが開いた。    ユーザーフォルダ直下に保存されると困るので、「保存先のパス」だけ変更してみた。    すると、sourcetreeが起動した。問題なく、cloneされたようだ。 |
|  |
|  |

## How to use Bitbucket

|  |
| --- |
| <https://bitbucket.org/score915/trial-git001>  とりあえず、ここをアクセスして、色々と実験出来そう。 |
|  |
|  |
|  |
|  |

# Python

## 事例紹介

|  |
| --- |
| Pythonの文法については、色々と勉強した。  ここでは、実際のコードでどんな結果が得られるのか、それらをメモしていく。 |
| 事例１ : 東京の最高気温データ **(case001.txtを参照)**  まず、真っ先に試したいのはこれ、Pycon 2016で紹介されたツール、ulmo。これを使うと手軽に各地の気象データがアクセス出来るらしい。まずは、pip install して、利用出来るようにする。    説明によると、この状態でPYTHONシェルに下記のコマンドをコピペすると、東京の気象データが取得出来るらしい。   |  | | --- | | import ulmo  import pandas  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  st =ulmo.ncdc.ghcn\_daily.get\_stations(country='JA', as\_dataframe = True)  st[st.name.str.contains('TOKYO')] |   実行したところ、確かにそれらしいデータが取得出来た。だけど、これだけじゃ意味がわからないし、面白くもなんともない。    取得出来たデータ（この場合、JA000047662 という観測地点の名前だと思う）を使って、再度、ulmoを実行すると、観測データが取得出来るらしい。   |  | | --- | | data = ulmo.ncdc.ghcn\_daily.get\_data('JA000047662', as\_dataframe = True) |     実行すると、データを取得しにいったみたい。どのくらいのデータを取得しているのかわからないけど、３−４分はターミナルが固まったので、相当重たいコマンドだと思う。  最後の仕上げとして、データを視覚化する。   |  | | --- | | tm = data['MAX'].copy()  tm.value = tm.value/10.0  tm['value'].plot() |   ところが、実際にコマンド実行してみると、下記のようなメッセージが表示されるだけで、肝心のグラフィックが表示されない。    プレゼンだと、かきのようなグラフが表示されると説明されているので、同じ出力が出ることを期待しているのに・・・。    ネットで色々と調べたら、下記、コマンドを実行しないとグラフィックは表示されないらしい。   |  | | --- | | plt.show() |   実行したところ、無事、期待した画面が表示された。    以上、色々とせつめいしたけれども、結局、東京の気象情報データを取得して、最高気温をグラフに出力するために記述したコードは１２行だけ。Pythonってすごいパワフルな言語だと感じる。   |  | | --- | | import ulmo  import pandas  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  st =ulmo.ncdc.ghcn\_daily.get\_stations(country='JA', as\_dataframe = True)  st[st.name.str.contains('TOKYO')]  data = ulmo.ncdc.ghcn\_daily.get\_data('JA000047662', as\_dataframe = True)  tm = data['TMAX'].copy()  tm.value = tm.value/10.0  tm['value'].plot()  plt.show() |   以上まで実行した段階で、Githubに上げた。 (2018/01/07)    週末サイエンティストのススメ:  <https://pycon.jp/2016/ja/schedule/presentation/72/>  <https://github.com/bakfoo/pyconjp2016> |
| 事例2 : 東日本大震災の地震データ **(case001.txtを参照)**  実は、一番衝撃を受けたのがこの事例。まず、obspyという地震学のツールを使う必要があるそうです。  まずは pipでobspyをインストール。    また、試行中に気づいたが、Basemapか Cartopy というモジュールが必要らしい。  Basemap nor Cartopy  苦労したけど、何とかBasemapをwgetを使ってインストール出来た（手順は、下に別途記述した）。  さて、いよいよお楽しみのコード実行。下記のコマンドを実行してみた。   |  | | --- | | import obspy  from obspy.clients.fdsn import Client  client = Client("IRIS")  starttime = obspy.UTCDateTime(2011,3,10)  endtime = obspy.UTCDateTime(2011,3,12)  cat = client.get\_events(starttime=starttime,endtime=endtime,minmagnitude = 6)  cat.plot() |   結果：    2011年３月10日から12日にかけて、震度６以上の地震が日本とインドネシアで発生していることがわかる。たったの７行のコードで。  <https://www.youtube.com/watch?v=KFXf4pJ_Ko0&feature=youtu.be&list=PLMkWB0UjwFGlYvIcvKsGhJ7SJIODj_zJZ>  <https://github.com/bakfoo/pyconjp2016> |
| 事例３ Google検索(case001.txtを参照)  **googleライブラリを使えば、たった数行のコードで、Google検索を実行したり出来る。**   |  | | --- | | **from google import search**  **def google\_search(query, limit=10):**  **for url in search(query, lang="jp", num=limit):**  **print(url)**  **#def main():**  **google\_search("qiita")** |   **実行すると、”qiita”というキーワードでgoogle検索の結果が延々と表示される。MacならCtrl＋Cで止めたほうが良い。**    <https://qiita.com/code_monkey/items/0d38a92a2a2026f5ec82> |
|  |
|  |

## 事例４：Webクローラ入門

|  |
| --- |
| **「クローラ**（Crawler）とは、[ウェブ](https://ja.wikipedia.org/wiki/World_Wide_Web)上の文書や画像などを周期的に取得し、自動的に[データベース](https://ja.wikipedia.org/wiki/%E3%83%87%E3%83%BC%E3%82%BF%E3%83%99%E3%83%BC%E3%82%B9)化する[プログラム](https://ja.wikipedia.org/wiki/%E3%83%97%E3%83%AD%E3%82%B0%E3%83%A9%E3%83%A0_(%E3%82%B3%E3%83%B3%E3%83%94%E3%83%A5%E3%83%BC%E3%82%BF))である。「ボット（Bot）」、「スパイダー」、「[ロボット](https://ja.wikipedia.org/wiki/%E3%83%AD%E3%83%9C%E3%83%83%E3%83%88)」などとも呼ばれる。」 ( [ja.wikipedia.org/](https://ja.wikipedia.org/wiki/%E3%82%AF%E3%83%AD%E3%83%BC%E3%83%A9) ) |
| Scrapy  クローラとして、Scrapyというフレームワークがある。これは、下記、５つのコンポーネントから構成されている。  1.Scrapy Engine  2.Scheduler  3.Downloader  4.Spiders  5.Item Pipeline  <https://qiita.com/checkpoint/items/038b59b29df8e1e384a2> |
| 1. items.py : 抽出データを定義する  2. spiders/以下のスパイダー(クローラー)ファイル：巡回、データ抽出条件  3. pipelines.py　：　抽出データの出力先。今回はmongoDB  4. settings.py　：　データ巡回の条件 (頻度や、階層など)  <https://qiita.com/tamonoki/items/ce58ff209f8eae808162> |
|  |
|  |

## urllib

|  |
| --- |
|  |
|  |
|  |
|  |

## 正規表現

## JSON

## dir ()

## HELP ():

|  |
| --- |
| help()はPython組み込み関数。  help()  help(object) |
|  |
|  |

## デコレータについて

|  |
| --- |
| Python @  (デコレータ)について  <https://ja.wikipedia.org/wiki/Decorator_%E3%83%91%E3%82%BF%E3%83%BC%E3%83%B3>  https://qiita.com/mtb\_beta/items/d257519b018b8cd0cc2e  locals()で自身の持つローカルな名前空間の値を辞書形式で返却します.  >>> def foo(arg):  ...     x = 10  ...     print locals()  ...  >>> foo(20)  {'x': 10, 'arg': 20}  (\*)callableとは引数を取って結果を返す, 呼び出し可能なオブジェクトの総称を指します. functionはもちろんclassやmethod, generatorなどもcallableです.  ここまでの話を踏まえて先に結論から言うと, デコレータとは「関数を引数に取り, 引き換えに新たな関数を返すcallable(\*)」です.  https://qiita.com/\_rdtr/items/d3bc1a8d4b7eb375c368  https://qiita.com/makky05/items/1be6cf0f1f238b5ba01b |

# Python Library

### Pandas

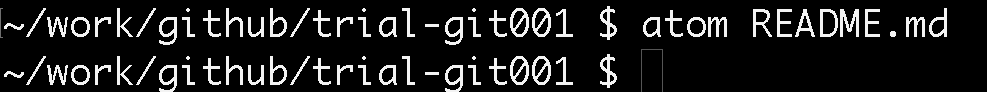
|  |
| --- |
| 「Pandasは、プログラミング言語Pythonにおいて、データ解析を支援する機能を提供するライブラリである。特に、数表および時系列データを操作するためのデータ構造と演算を提供する。PandasはBSDライセンスのもとで提供されている。」  <https://ja.wikipedia.org/wiki/Pandas>  <http://cocodrips.hateblo.jp/entry/2017/07/30/185430> |
|  |
|  |
|  |
|  |

# Atom

## How to use Atom

## Open file

atom ファイル名で特定のファイルを開いて編集出来る。



# Python Program

## Google search

## PyCon 2016

|  |
| --- |
| 週末サイエンティストのススメ:  <https://pycon.jp/2016/ja/schedule/presentation/72/>  講演の中で紹介されていた書籍「[超一流になるのは才能か努力か? 単行本 – 2016/7/29](https://www.amazon.co.jp/%E8%B6%85%E4%B8%80%E6%B5%81%E3%81%AB%E3%81%AA%E3%82%8B%E3%81%AE%E3%81%AF%E6%89%8D%E8%83%BD%E3%81%8B%E5%8A%AA%E5%8A%9B%E3%81%8B-%E3%82%A8%E3%83%AA%E3%82%AF%E3%82%BD%E3%83%B3-%E3%82%A2%E3%83%B3%E3%83%80%E3%83%BC%E3%82%B9/dp/)」、面白そうです。いずれ、読んでみたい。    <http://jupyter.org/>  <http://techlife.cookpad.com/entry/write-once-share-anywhare> |
|  |
|  |
|  |
|  |

## Pycon 2017

# Jupyter Notebook

## Jupyterとは

|  |
| --- |
| Jupyter Notebookとは、Pythonのコードを含んだWebページ(ドキュメント)を作成するツールだそうです。  「Pythonコードは単純なテキストではなくきちんとPythonコードとして実行されるため、実行可能なコードとその結果を併せて保存しておくことができます。その上Webページのリッチな装飾で解説を入れ、公開／共有できるという至れり尽くせりなツールなのです。」  <https://qiita.com/icoxfog417/items/175f69d06f4e590face9>  今後は、Jupyter Lab というツールになるらしい。  <https://www.youtube.com/watch?v=Ejh0ftSjk6g> |

## Jupiterの開始方法

|  |
| --- |
| jupyter notebookと入力すれば良い。 |

## 編集方法

|  |
| --- |
| Ctrl + Enter: セル内のプログラムを実行  Shift + Enter: 下にセルを追加  セルをダブルクリック: セルを再度編集可能にする  各種テキストを Markdown 形式でメモとして残すことが可能です。  <https://pythondatascience.plavox.info/python%E3%81%AE%E9%96%8B%E7%99%BA%E7%92%B0%E5%A2%83/jupyter-notebook%E3%82%92%E4%BD%BF%E3%81%A3%E3%81%A6%E3%81%BF%E3%82%88%E3%81%86> |

## ショートカット

|  |
| --- |
| Jupyterには便利なショートカットがいくつもある。  編集モードとコマンドモードがあるらしく、ESCを押すことでコマンドモードに移行して、使えるようになる。  一番多用しそうなのは、  　　Shift　＋　Enter：Pythonを実行させる |

## magic command: %

|  |
| --- |
| 「matplotlibのグラフをNotebook上に記述する際には%matplotlib inlineと追加する必要がある点に注意です。 このような%で始まる記法はmagic commandsというJupyterやIPythonの独自記法です。」  <https://deepage.net/machine_learning/2016/12/13/jupyter_notebook.html> |

## Jupyterの終了方法

|  |
| --- |
| Ctrl + C  <http://mashiroyuya.hatenablog.com/entry/2016/06/03/Jupyter%E3%82%92%E9%96%89%E3%81%98%E3%82%8B%E6%96%B9%E6%B3%95> |

# Mac Operation

## Screenshot

|  |
| --- |
| スクリーンショットを撮る3つの方法。  1.[command]キー　+　[Shift]キー　+　[3]キー  　　画面全体  2.[command]キー　+　[Shift]キー　+　[4]キー  　　ドラッグで選択した部分  3.[command]キー　+　[Shift]キー　+　[4]キー　+　[Space]キー  　　カメラアイコンが表示される。カメラアイコンで選択した一つのウインドウ  カメラのアイコンが現れる。 ウィンドウを選択してクリック。  <https://hamachan.info/mac/kihon/screen.html> |

## Paint

|  |
| --- |
| **Mac標準装備の「プレビュー」は、描画ソフトでもある** MacのFinderから、JPEG、PNG、PDFといったデータ形式のファイルをダブルクリックすると、「プレビュー」ソフトが起動します。 この「プレビュー」ソフトは、JPEGやPDFといった画像情報ファイルを参照できるだけでなく、描画する機能も有しているのです。何気なくMacを使っていると意外とこの機能に気がついていない人が多いのです。 **「プレビュー」の作図機能** 試しに「プレビュー」で適当なJPEGファイルを開いてみてください。そして、［メニュー］から［表示］ー［マークアップツールバーを表示］としてみてください。簡単な描画のためのツールバーが表示されます。  02_Preview  「プレビュー」の作図機能には、直線、矢印、長方形、長方形（角丸）、楕円、吹出し、文字等が使えます。また、線の太さ、色、塗りつぶしの指定もできます。  <http://minto.tech/mac-paint/> |
| スクリーンショットをWORDに貼り付けるには、  挿入　スクリーンショットと選択してから、貼り付けたい画像を選択する。  これも、Windowsとは使い勝手が違う。 |
|  |

## Finder (Explorer)

|  |
| --- |
| 簡単！MacのFinderを使いやすくする１０のカスタマイズ <http://maccle.com/my-tips/9-tips-improvement-mac-finder/> |
|  |
|  |
|  |

# Macの仕様とか、設定

|  |
| --- |
| クリップボードについて |
| Basemapインストール方法 pip install basemapだとインストール出来なかった。  ネットで調べたところ、下記の手順でインストール出来そうだとわかった。ところが、wgetもインスールされていなかった。幸い、brew install wgetでインストール出来た。   |  | | --- | | $ brew install geos  #$ pip install Pillow html5lib BeautifulSoup4  $ wget https://jaist.dl.sourceforge.net/project/matplotlib/matplotlib-toolkits/basemap-1.0.7/basemap-1.0.7.tar.gz  $ tar zxvf basemap-1.0.7.tar.gz  $ cd basemap-1.0.7  $ python setup.py install |   Unuxの世界は、こういうところが年々経っても変わらない、面倒くさい。  <https://qiita.com/msrks/items/ed18a2653bc177a24cca> |
|  |
|  |
|  |

# 参考文献

## GitHubの教科書

|  |
| --- |
| Web制作者のためのGitHubの教科書 チームの効率を最大化する共同開発ツール Web制作者のための教科書シリーズ Kindle版    特徴： gitとsourcetreeの使い方が紹介されている。  1-7章：  <https://www.amazon.co.jp/Web%E5%88%B6%E4%BD%9C%E8%80%85%E3%81%AE%E3%81%9F%E3%82%81%E3%81%AEGitHub%E3%81%AE%E6%95%99%E7%A7%91%E6%9B%B8-%E3%83%81%E3%83%BC%E3%83%A0%E3%81%AE%E5%8A%B9%E7%8E%87%E3%82%92%E6%9C%80%E5%A4%A7%E5%8C%96%E3%81%99%E3%82%8B%E5%85%B1%E5%90%8C%E9%96%8B%E7%99%BA%E3%83%84%E3%83%BC%E3%83%AB-Web%E5%88%B6%E4%BD%9C%E8%80%85%E3%81%AE%E3%81%9F%E3%82%81%E3%81%AE%E6%95%99%E7%A7%91%E6%9B%B8%E3%82%B7%E3%83%AA%E3%83%BC%E3%82%BA-%E5%A1%A9%E8%B0%B7-%E5%95%93-ebook/dp/B00QPSXY1I/ref=sr_1_1?ie=UTF8&qid=1515844896&sr=8-1&keywords=github%E3%81%AE%E6%95%99%E7%A7%91%E6%9B%B8> |
|  |
|  |
|  |
|  |
|  |