MacOSへのPythonインストール

2019-07-08　　山内 長承　　yamanouc@is.sci.toho-u.ac.jp

１）CUI （Character-based User Interface）を始める

CUI ⇔ GUI （Graphical User Interface）

ターミナルを開く

２）手元のPC (Mac) にPythonをインストールする

https://qiita.com/ms-rock/items/72b8f1abc661c539bb09 に従う。　いくつか前提インストールが必要

３）Xcode、Xcodeコマンドラインツールのインストール

Wikipediaより

> **Xcode**（エックスコード）はソフトウェアを開発するためのアップル)の統合開発環境 (IDE) であり、かつてはMac OS Xに付属する形で配布されていた。　..... 中略 .....　Macintosh (macOS) にてmacOSあるいはiOS用のアプリケーションを開発する場合、またソースコードで配布されているUNIX用ソフトウェアをインストールする場合に、Xcodeが必要になる。初期状態ではXcodeはインストールされておらず、Mac App Storeからの無料ダウンロードでインストールを行う。

⇒　Apple Storeからダウンロード・インストール

更に、Xcode-コマンドラインツールのインストール

```
sudo xcode-select --install
```

４）Homebrewのインストール

Wikipediaより

> **Homebrew**（ホームブルー）は、macOSオペレーティングシステム上でソフトウェア)の導入を単純化するパッケージ管理システムひとつである。MacPortsやFinkと同様の目的と機能を備えている。LinuxのDebianのAPTに似た使用感で急速に利用が広がっている。
>
> 基本命令はbrew。 ソフトウェアの導入はbrew install [ソフト名]。 brewの設定の更新はbrew update brewの設定・導入ソフトウェアの整合性の確認はbrew doctor

https://brew.sh/index_ja　から、スクリプトをコピペして実行

５）Homebrew + pyenvで Pythonをインストールする

上記のページ　https://qiita.com/ms-rock/items/72b8f1abc661c539bb09　の2.3.1～2に従う

まず、pyenv をインストールする　　pyenvを使うかどうかについていろいろ議論あり。

pyenv単独、pyenv + virtualenv 、venv　など。　venvがPython公式。

<脱線>　Python version2 と version3の問題

Pythonにはversion2と3があり、非互換な部分がある。歴史的経緯から、言語使用の一部を変更した。

Python2で書いたプログラムは、Python3では動かないことがある。

v3対応の書き直しをすれば、v2のプログラムでも動かせる（いくつか直す、変換プログラムも提供）。

変更後、しばらく経つので、新しいソフトはv3で書かれるし、古いソフトもv3対応に書き直している。

まだ、v3非対応のプログラムが残っているので、v2が残してあるが、近々v2は凍結の方向。

MacOS X でも、初期導入時はPython 2.7のみが入っているが、v3を使いたいのでインストールする。

その結果、v2と3が共存するので、切り替えのメカニズムが必要。その１つがpyenv。

<脱線終り>

```
brew install pyenv
```

更に、~/.bash_profileに下記を書き込む。具体的には

> vi ~/.bash_profile として、コマンド i を打つ　⇒　挿入モード

ページからコピペ

```
export PYENV_ROOT=${HOME}/.pyenv
if [ -d "${PYENV_ROOT}" ]; then
    export PATH=${PYENV_ROOT}/bin:$PATH
    eval "$(pyenv init -)"
fi
```

Escキーを押す（⇒ 挿入モード終了、`vi`のノーマルモードになる）⇒ `zz` (大文字) = `vi`の保存&終了

<脱線>　ファイルのディレクトリ指定と、`~`について

ファイルはシステム内で、トリーの構造になっている。　ファイルを指定する場合、大別して２つの方法がある。

１）絶対パス（フルパス）：　トリー内の位置を、頂上（ルート）から指定する

例）/Users/yamanouc/project1/data/mytest.dat　　戦闘が`/`から始まる

２）相対パス：　現在自分が居るディレクトリ（カレントディレクトリ）から見た、相対位置を指定

例）現在のカレントディレクトリが /Users/yamanouc/project1/ だとすると、data/mytest.dat

先頭に`/`が無い

すぐ上（親）のディレクトリは、`..` でも指定できる。　例：`../../tanaka/project2/schedule.txt`

カレントディレクトリの設定は `cwd <ディレクトリ>`　(change working directory)

カレントディレクトリの確認は `pwd`　(print working directory)

`~` はbashの中で、自分のホームディレクトリを表す記号

`~/.bash_profile`はホームディレクトリ直下にある`.bash_profile`という名前のファイル

なお、ドット `.` で始まるファイル名は、`ls`コマンドで（`-a`を指定しない限り）表示されない

<脱線終り>

<脱線>　`vi`(`vim`)について

（ウィキペディアから抜粋）

> **vi**（ヴィーアイ）は、Emacsと共にUNIX環境で人気があるテキストエディタ。`vim`はその拡張版
>
> マウスを使わない、　カーソルキーを使わない、　命令を覚える必要がある（画面上に命令表示領域がない）
>
> コンパクトで負荷が小さいという利点から、最低限のUNIX環境でも含まれている事が多く … …

操作法については、たとえば、http://turbo.mech.iwate-u.ac.jp/Fel/unix/vi.html　参照　（その他にも多数あり）

以下、最低限の操作法：

０）起動：　`vi ファイル名`　　ファイルがなければ新規作成

１）モード：　ノーマルモード（カーソル移動・テキスト削除など）と、挿入モード（キー入力をそのまま挿入、Escで終了）

ノーマルモードでの操作：

２）カーソル移動：　矢印キーで移動できる。元のバージョンでは`k`が↓、`j`が↑、スペースが→、`h`が←

３）テキスト削除：　`x`キーでカーソル上の１文字削除、BackSpaceキーで１つ前の文字を削除、`dd`で１行削除

４）Exコマンド：　コロン「`:`」で始まるコマンド、`:q`は終了、`:w`はセーブ、`:wq`は保存終了（＝`zz`と同じ）

　　ファイルを変更したのに、セーブせずに終了したいときは、`:q!`　　`!`は強制実行

挿入モードでの操作：

５）入力した文字は、そのままファイルに挿入される

６）Excキーで、挿入モードからノーマルモードへ戻る

<脱線終り>

次に、pyenvを使ってPythonをインストール。　以下をコマンド入力　　（Python version 3.7.3が最新）

```
pyenv install 3.7.3
```

確認のため

```
pyenv versions
```

<脱線>　　pyenvのサブコマンド

pyenvのサブコマンドは　`pyenv help`で確認できる。たとえば

　　local Set or show the local application-specific Python version　　global Set or show the global Python version　　install Install a Python version using python-build　　uninstall Uninstall a specific Python version　　version Show the current Python version and its origin　　versions List all Python versions available to pyenv

<脱線終り>

`mkdir`コマンドで作業ディレクトリを作って、`cd`コマンドでそこへ移動、`pyenv local`でそのディレクトリの環境をPython3.7.3にする。

```
mkdir work
cd work
pyenv local 3.7.3
```

<脱線>　　コマンド　`mkdir`　と　`cd`

`mkdir <ディレクトリ>` コマンド：　ディレクトリを新たに作る

　　`mkdir work` は、カレントディレクトリの下にworkというディレクトリを新しく作る（相対パス指定）

`cd` <ディレクトリ>　コマンド：　指定したディレクトリに移る　（カレントディレクトリを移動する）

　　`cd work` は、カレントディレクトリの位置を、現在のカレントディレクトリの直下の`work`に移す（相対パス）

　　絶対パスを指定してもよい。`cd /home/yamanouc/project1` など。

<脱線終り>

（見えている）Pythonのバージョンの確認

```
python -V　　⇒ 3.7.3と出るか？
```

jupyter notebookのインストール

```
pip install jupyter notebook
（結構時間がかかる）
```

<脱線>　　`pip`コマンド

`pip`コマンドは、OSレベルでのコマンド　　（`python -m pip ...`　としてもよい）

Pythonではライブラリパッケージを共有する仕組としてpipを使う。

レポジトリ（https://pypi.org）に登録されたライブラリパッケージを、`pip`コマンドでダウンロードしインストールできる。

`pip list`　手元PCにインストール済みのパッケージの一覧

`pip install <package>`、`pip uninstall <package>`　パッケージのインストール・削除

　　`pip install --upgrade <package>`　最新版へのアップグレードインストール

`pip search <keyword>`　パッケージの検索

なお、パッケージ名と、インストール後のパッケージの呼び出しの名前とは、異なることがあるので注意

<脱線終り>

インストールできたら起動

```
jupyter notebook
```

ブラウザ（の新しいタブ）が開く。（使える状態になる。ここからブラウザ内で操作）

ここからPythonを試してみる

> ブラウザで、右上の`new`をプルダウン。Python3を選ぶ。

開いた新規ウィンドウへ入力

```
s = 'おーい'
print(s)
```

Fileタブから`Rename`を選んで、ファイル名を変更する。拡張子は`ipynb`。

runボタンをクリック　→プログラムを実行する

jupyter notebookを終了するときは、

プログラムを書いたウィンドウは、Fileタブから`Save and checkpoint`で保存し（自動で保存されている）

　もう一度Fileタブから`Close and Halt`を押す。→ウィンドウが閉じる。

一番初めに開いたウィンドウがファイルのリストになっているはず。　これを閉じるには、

　右上の`Logout`をクリック。→ウィンドウが消えるはず。

最後に、ターミナル画面で、`Ctrl-C`（Controlを押しながらC）→　終わってよいか訊かれる　→　`y`

時間があれば、ターミナル画面でのコマンドの例をいくつか

`ls`　　ファイルをリストする

　パラメータを指定しなければ、カレントディレクトリの下のファイルを（１層分）リストする。

　ファイル名を指定すれば、そのファイルをリスト（この場合は属性を表示したいことが多い）

　ファイル名にワイルドカード（主に`*`）を使って、パターン一致するものを表示

　　「`*`」はマッチする文字列すべて（0文字も含む）　`ls *`なら、すべてのファイルを表示

　　`ls *.txt`は拡張子が`txt`であるファイルをすべて表示

　　`ls Sample23*.bam`なら、`Sample23_1-4.bam`や`Sample231chilled.bam`などを表示

　`-l`を指定すると、ロングフォーマットで、ファイル長や更新日時などを表示

　`-t`を指定すると、更新日時順（逆順）にソートして表示　`-lt`とすると便利

`less/more`　　スクリーン表示をページ単位で止める

　スクリーン表示の行が多いとき、ページ単位に表示するフィルター。　たとえば

　　`ls -l | less`　　`ls -l`で多数のファイルを表示するとき、１画面ごとに止める

　制御のためのキー操作は、`man less`で表示できる。　たとえば、

　　次のページに進む　　`スペース`

　　前のページに戻る　　`b`

　　１行進む　　`リターン`

　　１行戻る　　`y`

　　ファイル先頭に戻る　　`g`

　　終了する　　`q`

＜脱線＞　　フィルタとは

　前提知識：　標準入出力

プログラムの入力や出力を、ファイル名を指定するのではなくて、標準入力・標準出力（・標準エラー）を指定してプログラムすることができる。<プログラムの中にそのように指定しておく>

標準入出力は、プログラム起動時に（シェル/OSが）デフォルトではキーボード・画面に繋ぐ。

起動時に、他のもの（ファイル等）に繋ぐことができる（redirectionと呼ぶ）。たとえば

標準出力をファイルに変更： `cat infile` ⇒ `cat infile > outfile`

標準入力をファイルに変更： `cat > outfile` ⇒ `cat < infile2 > outfile`

この例は分かりにくいが、`cat`はパラメータを指定しないと標準入力から読み込むが、その

その標準入力をredirectionによって`infile2`に変更した

前提知識： パイプ

`bash`（やその他のシェル）では、２つのコマンドAとBを `A|B` のように繋いで（パイプと呼ぶ）、Aの標準出力をBの標準入力に繋ぐことができる。

本来ならAの出力をファイルに作って、そのファイルをBに読み込ませるのだが、それをまとめられる。

更に、ファイルを作らないので（データバッファで渡す）高速になり、Aが出力しながらBが入力するので、AとBが並列に動くメリットがある。

フィルタ：

Unixでは、処理単位ごとに、標準入力から標準出力へデータを変換する形のプログラムとして作り、それを繋ぐことで、まとまった処理をするという考え方・スタイルがあった。この場合、各ステージを（流れていくデータに対する）フィルタと考え、そう呼んでいた。