

# 大阪大学医学部Python会 新入生・初心者向けオンライン勉強会

第一回：Python会とは？ / Pythonの基礎

※Python会内の勉強会を特別に一般公開いたします



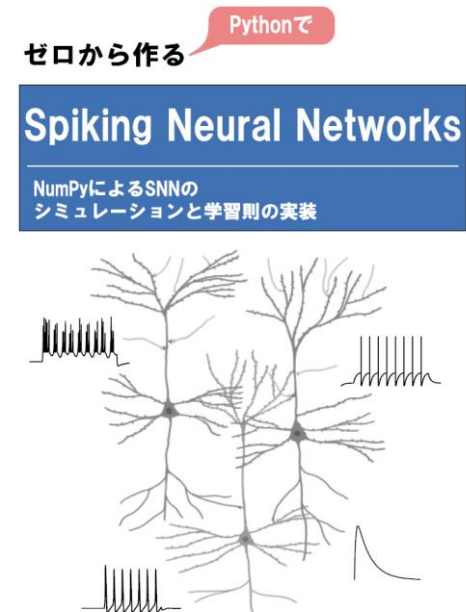
OU medical python

大阪大学医学部Python会代表  
大阪大学医学部医学科4年次

山本 拓都

# 自己紹介

- 医学部4年次
- 主に神経科学の研究をしています (´・ω・`)
- 脳生理学と神経機能形態学と脳外科(ほとんど研究できてない)に所属しています。
- Pythonで神経系のシミュレーションをする同人誌を昨年書きました
- Twitter: @tak\_yamm
- 学会抄録の締め切り3日前…



# 目次

一般に公開はしてはいますが、メインターゲットが新入生向けなのでご了承ください

[https://github.com/takyamamoto/python\\_tutorial](https://github.com/takyamamoto/python_tutorial)

- Python会とは（´・ω・`）？
- Pythonの環境構築
- Pythonを今後学ぶには
- 今後の勉強会の予定

Python会とは (´・ω・`)?

# Q. Python会って何ですか？

## 大阪大学医学部 Python会

*Now is better than never.*

### Python会について

Python会とは、大阪大学医学部の学生が中心となり、プログラミング言語Pythonを軸に、医療分野と絡めて個人の研究・趣味に取り組む会です。

RNA発現データ、がんゲノムの遺伝子変異、患者の画像データなど、医療データはこれまでの常識では考えられないスピードで増加しています。bioinformatics、脳科学、医療情報学や画像解析など様々な分野において、メンバーそれぞれが日々研鑽しています。

学部1回生から6回生まで参加しており、学年を超えて自由に活動しています。定期的に勉強会・交流会を開き、メンバー同士の情報交換を目的に交流しております。Python会に興味のある方はぜひ、[Contact](#)のページからご連絡ください。お待ちしております。

—Pythonで医学に貢献を—

<https://oumpy.github.io/>



# 実験医学への寄稿



[SHARE] [ツイート](#) [シェア](#) [B!ブックマーク](#)

本コーナーでは、実験医学連載「Opinion」からの掲載文をご紹介します。研究者をとりまく環境や社会的な責任が変容しつつある現在、若手研究者が直面するキャリア形成の問題や情報発信のあり方について、現在の研究現場に関わる人々からの生の声をお届けします。（編集部）

## 第105回 Pythonで医学に貢献を

「実験医学2019年3月号掲載」

Pythonをご存じだろうか。名前だけでも聞いたことがあるかもしれない。Pythonとは、汎用プログラミング言語であり、シンプルなコードと幅広い応用分野を特徴とする。Pythonを駆使するとさまざまなことが可能となる。簡単なグラフが描けるだけではない。微分方程式を解くことも機械学習でレントゲンやCTの画像を解析することも、1細胞RNA-seqやChIP-seqの解析だってできる。例えば、Kaggleなどのデータサイエンスのコンペティションではほとんどの参加者がPythonかRを用いて解析を行っている。ちなみに最近の私の趣味はドローンをPythonで操ることだ。



安水さん

安水 良明「Pythonで医学に貢献を」実験医学 Vol. 37 No. 4 (2019).  
<https://www.yodosha.co.jp/jikkenigaku/opinion/vol37n4.html>

# Q. Python以外はダメなんですか？



~~(F...)~~ ~~か...~~ @biocycle1985 · 4月12日

誰か阪医Julia会と阪医R会を立ち上げて阪医Python会と血で血を洗う抗争を繰り広げて欲しい。



A. ダメではないです。むしろ発足時はPythonよりもR使っている人も多かったぐらいです。プログラミングの入門としてPythonから入るといいよ、程度。

# 必須事項さえ守ってくれば 基本的には自由にしてくれてOK

『Python会ルール 2020年度版』より抜粋

会員の必須事項は、**Slackの常時確認**と**リレー投稿への参加**の2つです。  
勉強会その他はすべて自由参加。もちろん積極的な参加・企画が推奨されます。

【よくある質問】

Q. 兼部していいか？

A. いいぞ

Q. 忙しいので辞めたい

A. いいぞ

Q. 研究が始まったのでもう一度入りたい

A. いいぞ





# Pythonの環境構築

# Pythonの環境構築

- Pythonの公式サイト(<https://www.python.org/>)からPythonだけをinstallしても使えるが、パッケージ管理や環境管理のためにAnacondaを使うと便利(ここで不要じゃという声が飛んでくる)。
- 少なくともWindowsだと使った方が楽
- Anacondaそのままは全部盛りに近い(=要らないものなども自動でinstallされる)ので、最小限のパッケージに留めたMinicondaをinstallする
- Miniconda入れる前にpyenv入れた方が良いという話もある(MacOS, Linux)

# Minicondaのinstall

miniconda



[すべて](#)

[動画](#)

[画像](#)

[ニュース](#)

[地図](#)

[もっと見る](#)

[設定](#)



約 325,000 件 (0.32 秒)

[docs.conda.io](#) > [latest](#) > [miniconda](#) ▼ [このページを訳す](#)

## [Miniconda — Conda documentation](#)

**Miniconda** is a free minimal installer for conda. It is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib and a few others.

[Installation](#) · [Miniconda hash information](#) · [Help and support](#) · [Conda license](#)

このページに 3 回アクセスしています。前回のアクセス: 20/03/24

# Minicondaのinstall

## Miniconda

<https://docs.conda.io/en/latest/miniconda.html>

Miniconda is a free minimal installer for conda. It is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib and a few others. Use the `conda install command` to install 720+ additional conda packages from the Anaconda repository.

[See if Miniconda is right for you.](#)

## Windows installers

### Windows

Python version	Name	Size	SHA256 hash
Python 3.7	Miniconda3 Windows 64-bit	51.6 MiB	1701955cd637d1dad5a84958fd470649b79de973d1570541eb52857664b5056c
	Miniconda3 Windows 32-bit	52.2 MiB	ca74cb6eb0731db2b972c0fb512e29661a84c3f01ac6133121b4372eb1c41f46
Python 2.7	Miniconda2 Windows 64-bit	50.9 MiB	8647c54058f11842c37854edeff4d20bc1fbdad8b88d9d34d76fda1630e64846
	Miniconda2 Windows 32-bit	48.7 MiB	0d106228d6a4610b599df965dd6d9bb659329a17e3d693e3274b20291a7c6f94



Python3.7の方を選ぶ  
(いつになったらPython2  
は消えるんだ…)

## MacOSX installers

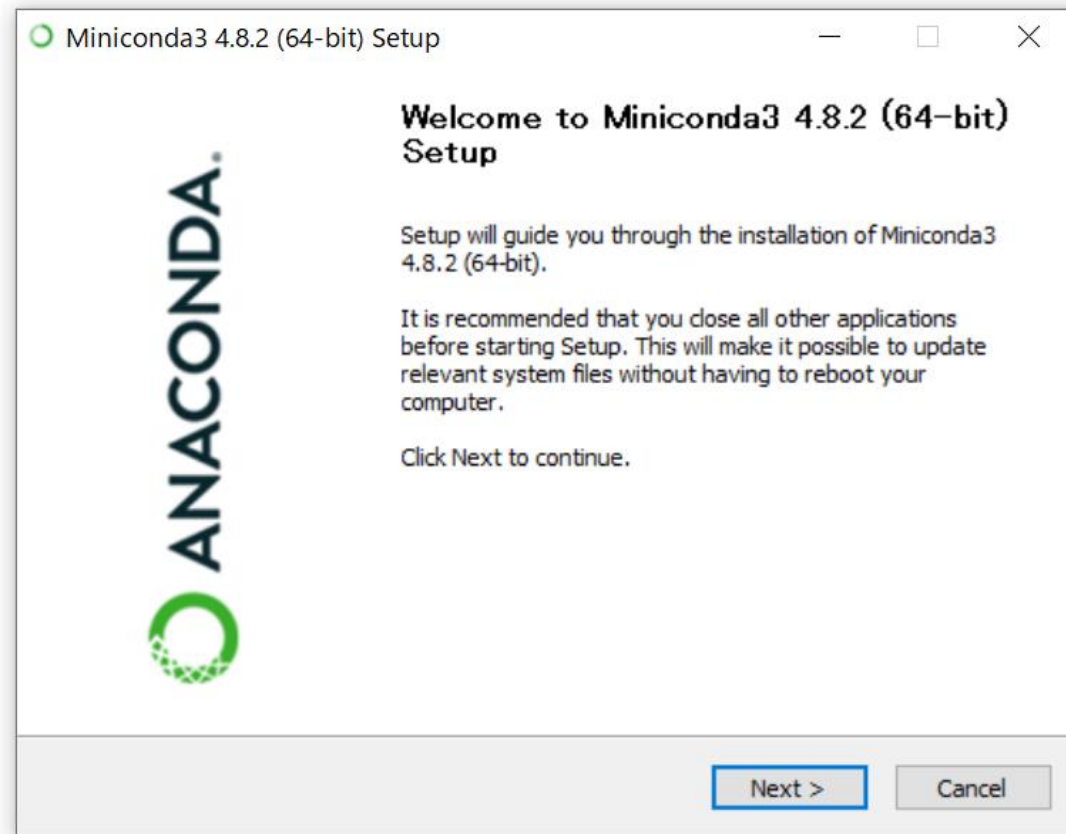
### MacOSX

Python version	Name	Size	SHA256 hash
Python 3.7	Miniconda3 MacOSX 64-bit bash	50.3 MiB	d1fca4f74f9971c27220122723843f6c879a5d13ff59c01fca17ef62a1576732
	Miniconda3 MacOSX 64-bit pkg	61.3 MiB	f3ede3a58d82fb5dcbca52d291a9edb5cd962d84d823a20693dd4bb27506cdd0
Python 2.7	Miniconda2 MacOSX 64-bit bash	39.4 MiB	0db8f4037e40e13eb1d2adc89e054dfb165470cc77be45ef2bf9cb31c8b72f39
	Miniconda2 MacOSX 64-bit pkg	47.8 MiB	fcc30b2e18f7a292b34b2e24ad855786a66423f860157fa2b77e48b6392f0abb



# Minicondaのinstall (Windows)

`Miniconda3-latest-Windows-x86_64.exe` を実行するとこのようになる  
後はNextを押して規約に同意し、installしたい場所などを選択する



# Minicondaのinstall (MacOS)

手元にMacが無いので以下の記事参照

minicondaでスマートに環境構築 for Mac

<https://qiita.com/hamapon/items/9520a8e7e569137256de>

Minicondaを使う

<https://qiita.com/Scstechr/items/10e460c66fb3cbf3bb22>

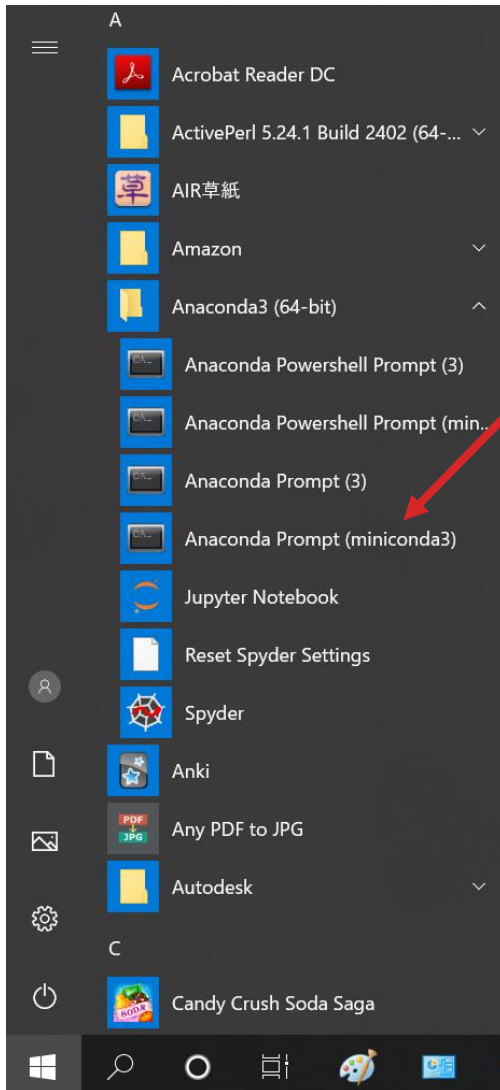
基本的には**Miniconda3-latest-MacOSX-x86\_64.sh**をdownloadして

```
$ bash Miniconda3-latest-MacOSX-x86_64.sh
```

を実行

# Pythonとcondaのinstallの確認

Windowsなら**Anaconda prompt**, MacOSなら**terminal**を起動して**python** と入力して実行(Enter)



```
Anaconda Prompt (miniconda3)

(base) C:\Users\user>python
Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
:: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()

(base) C:\Users\user>
```

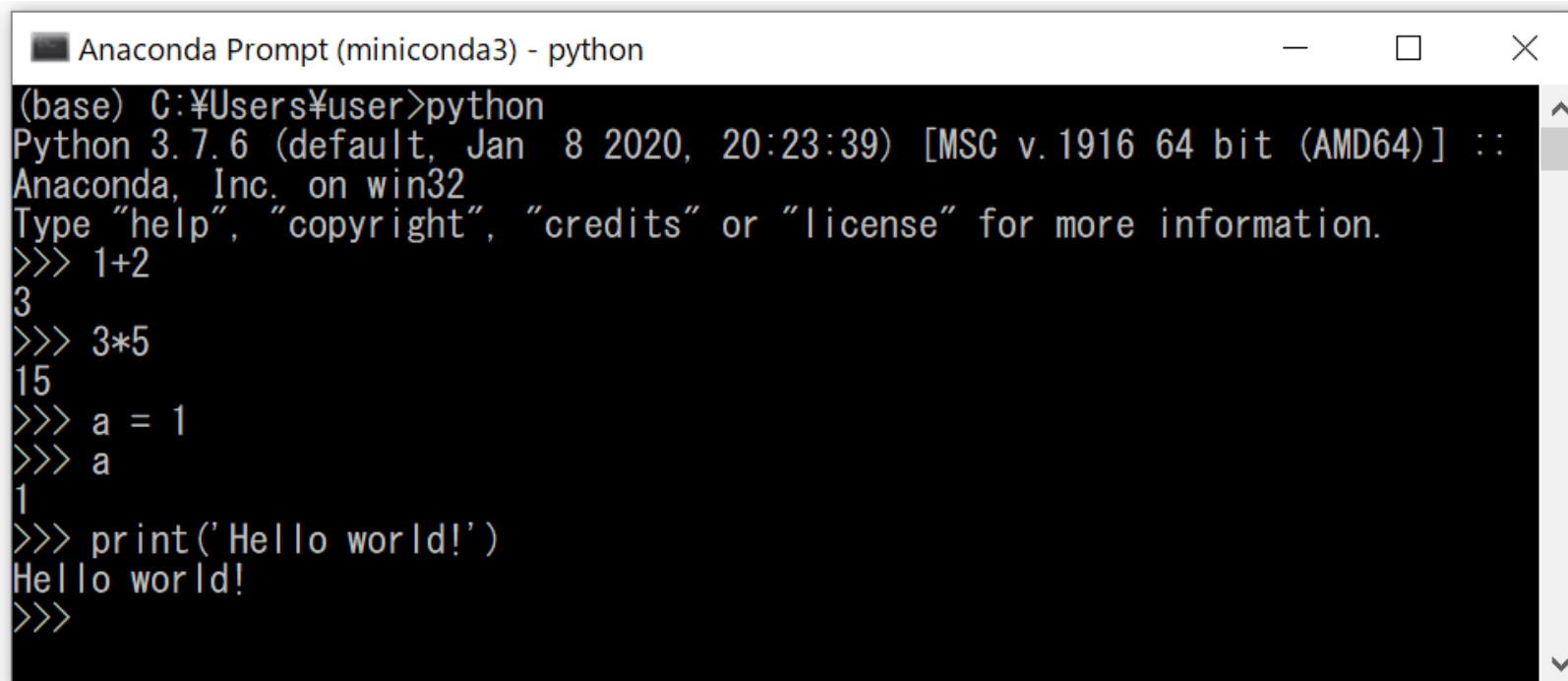
Pythonが起動されれば成功。Pythonを終了するには **quit()** を入力して実行

```
(base) C:\Users\user>conda -V
conda 4.8.3
```

**conda -V** と入力して condaが入っているかも確認 (バージョン確認のコマンド)

# Pythonを実行してみる

Windowsなら**Anaconda prompt**, MacOSなら**terminal**を起動して **python** と入力して実行(Enter)

A screenshot of an Anaconda Prompt terminal window. The title bar reads "Anaconda Prompt (miniconda3) - python". The terminal content shows the following sequence of commands and outputs:

```
(base) C:\¥Users¥user>python
Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)] ::
Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+2
3
>>> 3*5
15
>>> a = 1
>>> a
1
>>> print('Hello world!')
Hello world!
>>>
```

>>> の後ろに入力して実行すると、その下に結果が表示される  
(このような言語を**インタプリタ型言語**と言います)



# IDE（統合開発環境）を入れる

- **IDE（統合開発環境）**はコーディングやコンパイル、デバッグなどができるソフトウェア
- とりあえずcondaで入る**Spyder**（Anacondaを入れた場合はデフォルトで入っている）を入れる
- Visual Studio CodeやPyCharmなど、他にも色々あります。好きなもの使ってください…。

Installには **conda install spyder** を実行。その後、installするかをyes (y) or no (n)で聞かれるので **y**を入力して実行

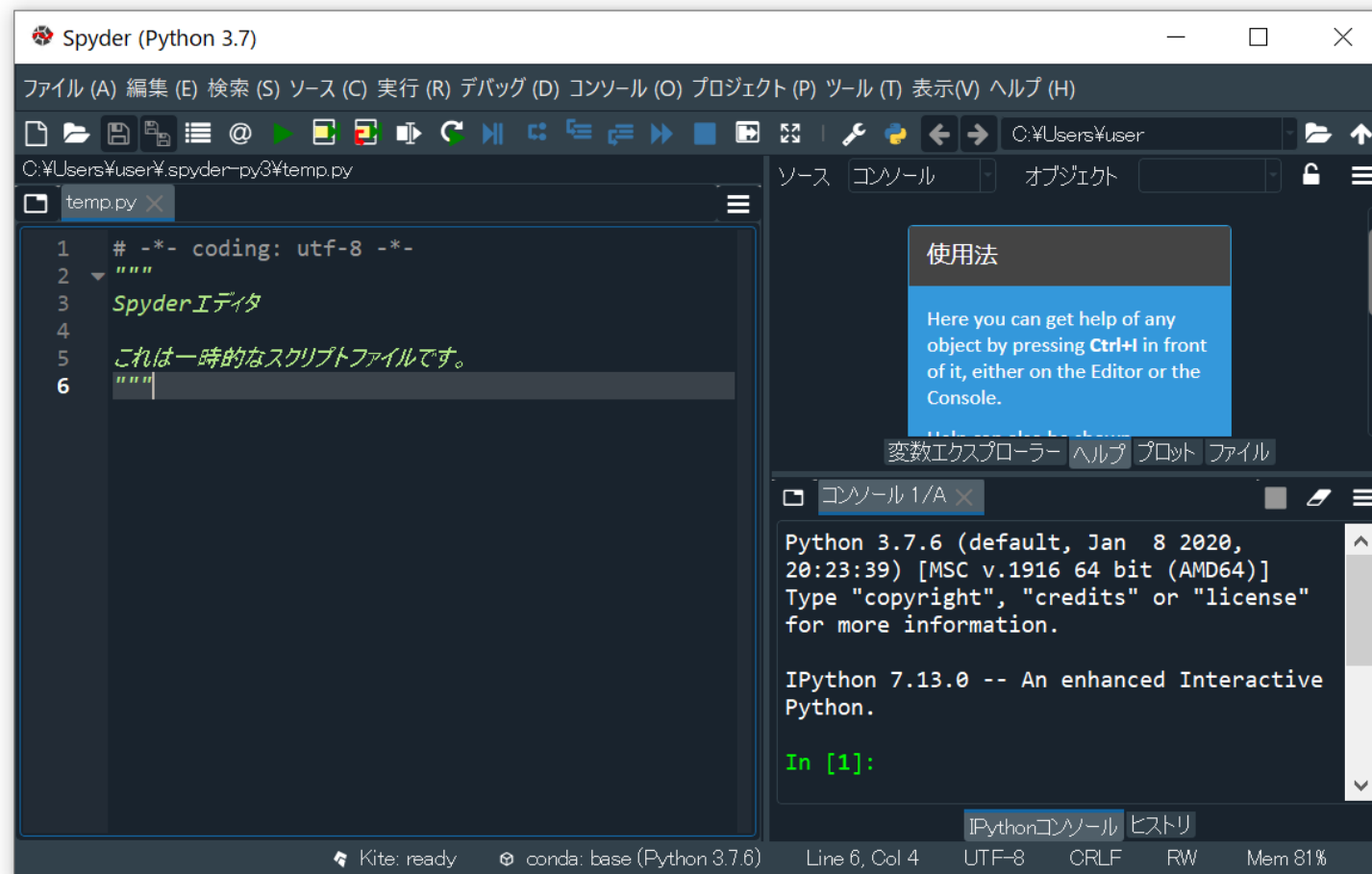
- Windows (Anaconda prompt)  
**(base) > conda install spyder**
- Mac (terminal)  
**\$ conda install spyder**

Uninstallには **conda uninstall spyder** を実行

installしても起動しない場合もあるけど、その場合はAnaconda prompt or terminalから実行してエラーを見てみる（pipでライブラリを入れる必要あり）

# Spyderの起動

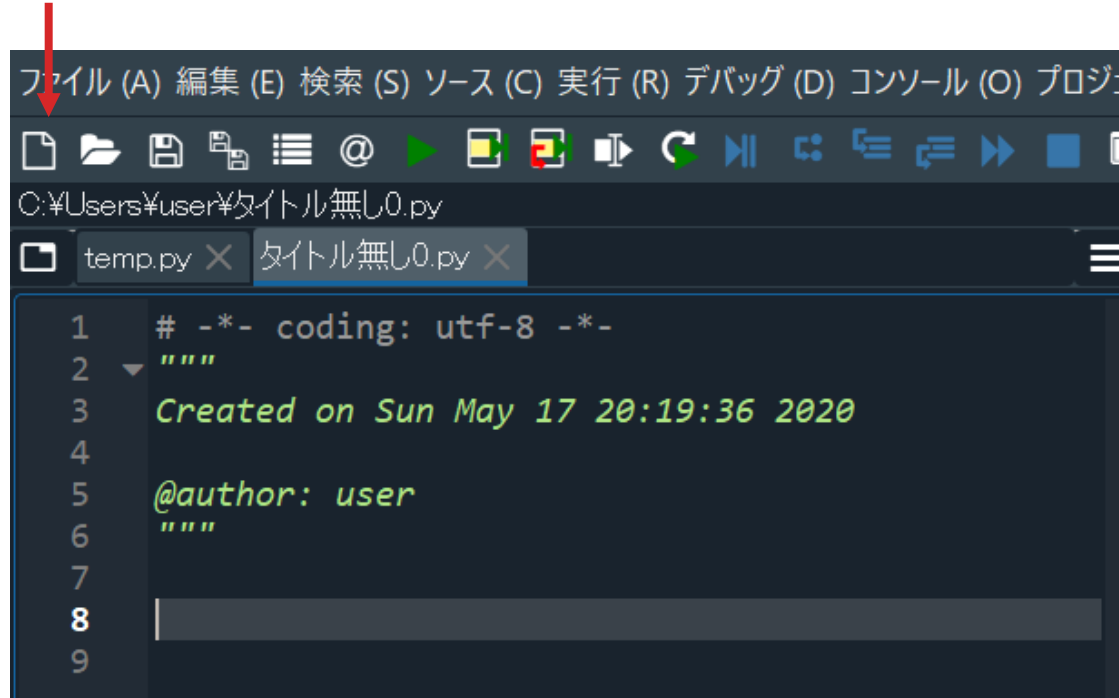
\$ Spyder と入力して実行し（またはスタートメニューからSpyderを選択して実行し）、以下のような画面が出ればOK（Kiteというコード補間ソフトのinstallの表示も出るが入れた方が便利）



# Pythonファイルの作成 ①

好きなところにお好きな名前のフォルダ(ディレクトリ)を作成しておく。  
(自分はDesktopに作成。PATHは C:¥Users¥user¥Desktop¥Python\_test )

新規ファイル作成

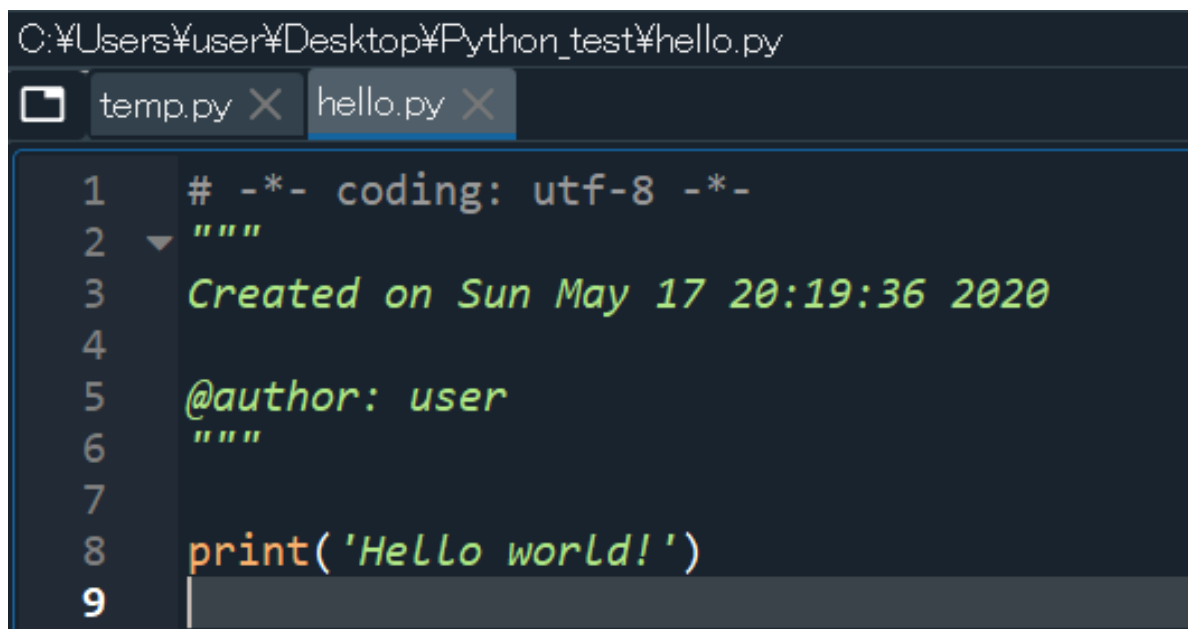


```
ファイル (A) 編集 (E) 検索 (S) ソース (C) 実行 (R) デバッグ (D) コンソール (O) プロジェ  
C:¥Users¥user¥タイトル無し0.py  
temp.py × タイトル無し0.py ×  
1 # -*- coding: utf-8 -*-  
2 """  
3 Created on Sun May 17 20:19:36 2020  
4  
5 @author: user  
6 """  
7  
8  
9
```

「新規ファイル」か Ctrl+N (Windowsの場合) で新規のpythonファイル(拡張子は .py)を作成する。

# Pythonファイルの作成 ②

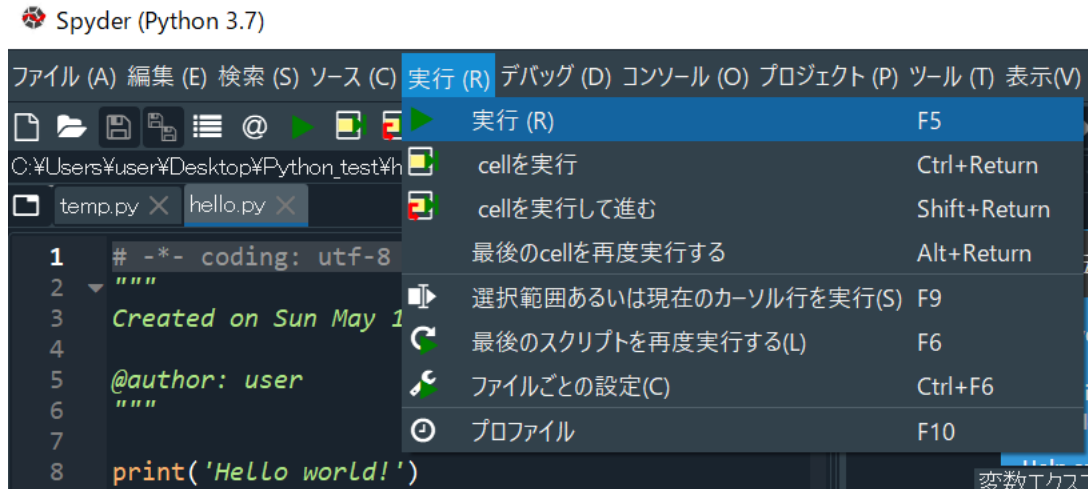
print('Hello world!') と入力し、ファイル > 形式を指定して保存から、先ほど作成したフォルダ(ディレクトリ)にお好きな名前を付けて保存する(今回は hello.py としておく)



```
C:\Users\user\Desktop\Python_test\hello.py
temp.py x hello.py x
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun May 17 20:19:36 2020
4
5 @author: user
6 """
7
8 print('Hello world!')
9
```

# Pythonファイルの実行 ①

- 上のメニューから「実行」を選択して(またはF5キーで)実行。初回は実行設定がでると思うが、「現在のコンソールで実行」を選択して下の「実行」を押す。
- 実行すると初期配置右下のIPythonコンソール上で hello.py が実行されて Hello world! が表示される。

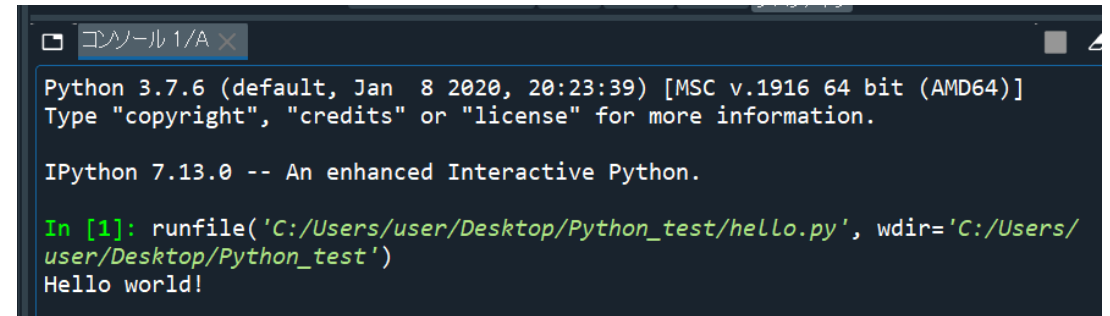


Spyder (Python 3.7)

実行 (R) デバッグ (D) コンソール (O) プロジェクト (P) ツール (T) 表示 (V)

- 実行 (R) F5
- cellを実行 Ctrl+Return
- cellを実行して進む Shift+Return
- 最後のcellを再度実行する Alt+Return
- 選択範囲あるいは現在のカーソル行を実行(S) F9
- 最後のスクリプトを再度実行する(L) F6
- ファイルごとの設定(C) Ctrl+F6
- プロファイル F10

```
1 # -*- coding: utf-8
2 """
3 Created on Sun May 1
4
5 @author: user
6 """
7
8 print('Hello world!')
```



```
Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

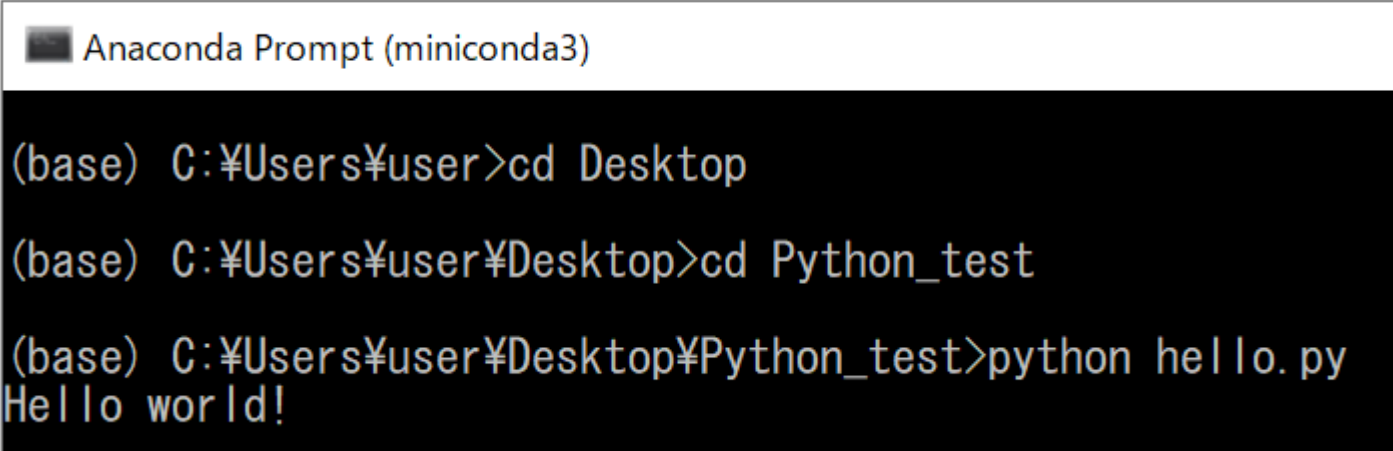
IPython 7.13.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/user/Desktop/Python_test/hello.py', wdir='C:/Users/
user/Desktop/Python_test')
Hello world!
```

# Pythonファイルの実行 ②

CUIで実行する方法（慣れると基本はこれ）。

Anaconda Prompt or Terminalを起動して、cdコマンドで hello.py を保存したディレクトリに移動する。移動したら python hello.py で実行できる



```
Anaconda Prompt (miniconda3)
(base) C:\Users\user>cd Desktop
(base) C:\Users\user\Desktop>cd Python_test
(base) C:\Users\user\Desktop\Python_test>python hello.py
Hello world!
```

Desktopに移動 →

Python\_testに移動 →

実行 →

出力 →

※初心者向けに言っておくとTABキーを押すとコマンドが補完されます  
例えば python h まで入力してTABキーを打てば hello.py が補完されます

# ライブラリを入れる

- **ライブラリ**(頭いい人が書いてくれた便利プログラム集)を入れる。これを使うと複雑な処理を簡単に記述できる。
- まずはPythonでよく使われる**NumPy, Matplotlib, Seaborn, Pandas**をpipコマンドで入れる。

Anaconda prompt or terminalを起動して

```
$ pip install numpy matplotlib seaborn pandas
```

を実行 (pip install numpyのように1つずつinstallしてもOK)

ライブラリはcondaでもinstallできるけど混ぜるな危険。condaとpipの違いについては<https://insilico-notebook.com/conda-pip-install/>等を参照。

# NumPyで数値計算 ①

- Pythonのforループは型推論等により遅いので、C/C++やFortranで書かれたライブラリであるNumPyを使うと高速化される
- コードを書く人が行列演算及び線形代数を深く理解していればPythonで高速な処理も可能（はい、そこJulia使えよとか言わない）
- いいかい学生さん、分からないことがあったらな、ネット上の変な記事じゃなくてな、公式ドキュメント、公式ドキュメントを読むんだ…



<https://twitter.com/nekomath271828>



# NumPyで数値計算 ②

## コード (plot\_test.py)

```
(1) import numpy as np
(2) x = np.arange(0, 1, 0.1)
    print('x:', x)
(3) y = 2*x + 10
    print('y:', y)
```

実行  
→

## 実行結果

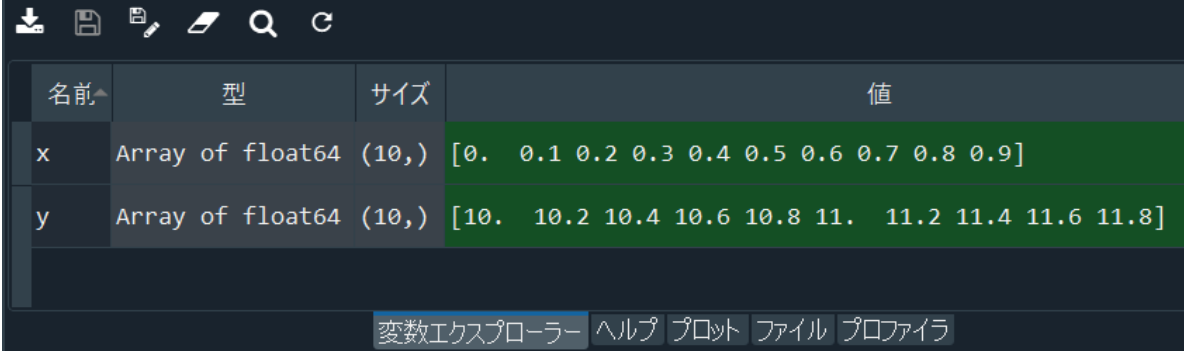
```
x: [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
y: [10. 10.2 10.4 10.6 10.8 11. 11.2 11.4 11.6 11.8]
```

右上の変数エクスペローラーパネルに  
値が記録されていることも確認

(1) numpyというライブラリをimport(使用できるようにすること)してnpという省略した形式で以後使用

(2) numpyにより0から1まで0.1刻みの配列(ベクトル)を作成

(3) xの要素を2倍して10を加算



名前	型	サイズ	値
x	Array of float64	(10,)	[0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
y	Array of float64	(10,)	[10. 10.2 10.4 10.6 10.8 11. 11.2 11.4 11.6 11.8]

変数エクスペローラー ヘルプ プロット ファイル プロファイル

# Matplotlibで図を描画する ①

Matplotlibは図を描画するためのライブラリ。どんな図が描画できるかは公式ドキュメントのギャラリー(<https://matplotlib.org/gallery/index.html>)を見よう



Installation Documentation Examples Tutorials Contributing

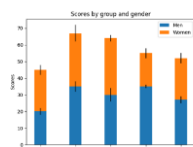
home | contents »

## Gallery

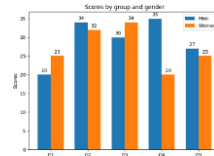
This gallery contains examples of the many things you can do with Matplotlib. Click on any image to see the full image and source code.

For longer tutorials, see our [tutorials page](#). You can also find [external resources](#) and a [FAQ](#) in our [user guide](#).

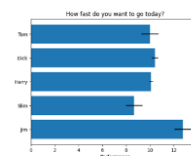
### Lines, bars and markers



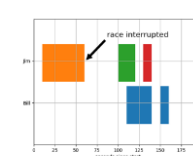
Stacked bar chart



Grouped bar chart with labels



Horizontal bar chart



Broken Barh

# Matplotlibで図を描画する ②

コード (plot\_test.py)

※追加した部分を赤で表示

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 1, 0.1)
print('x:', x)

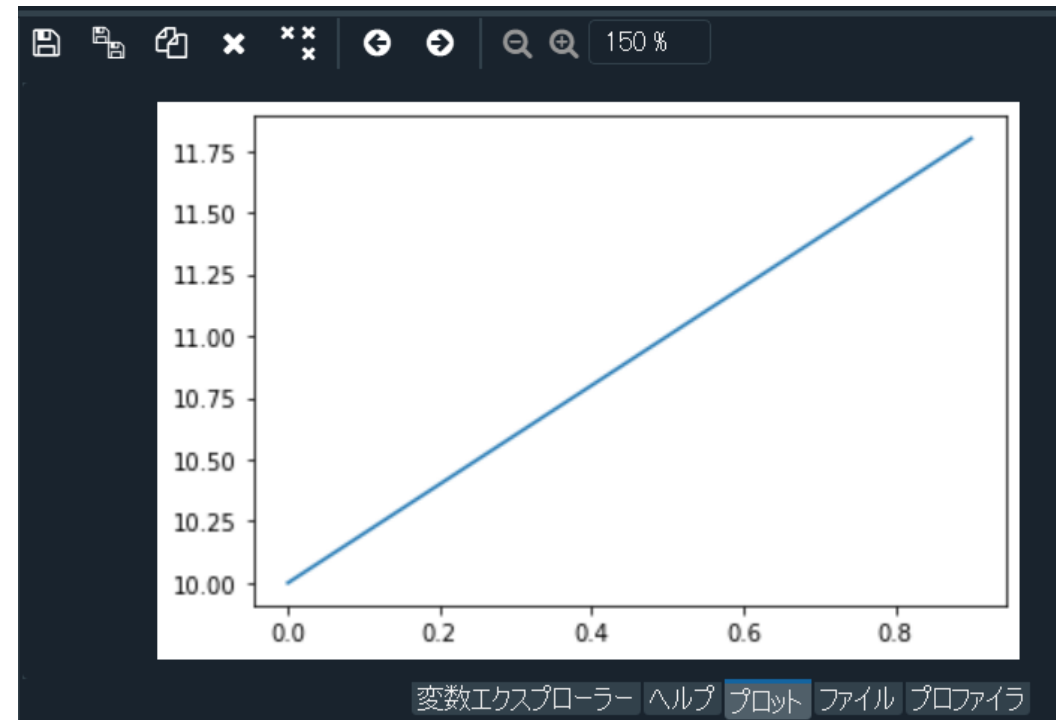
y = 2*x + 10
print('y:', y)

plt.plot(x, y)
plt.show()
```

実行  
→

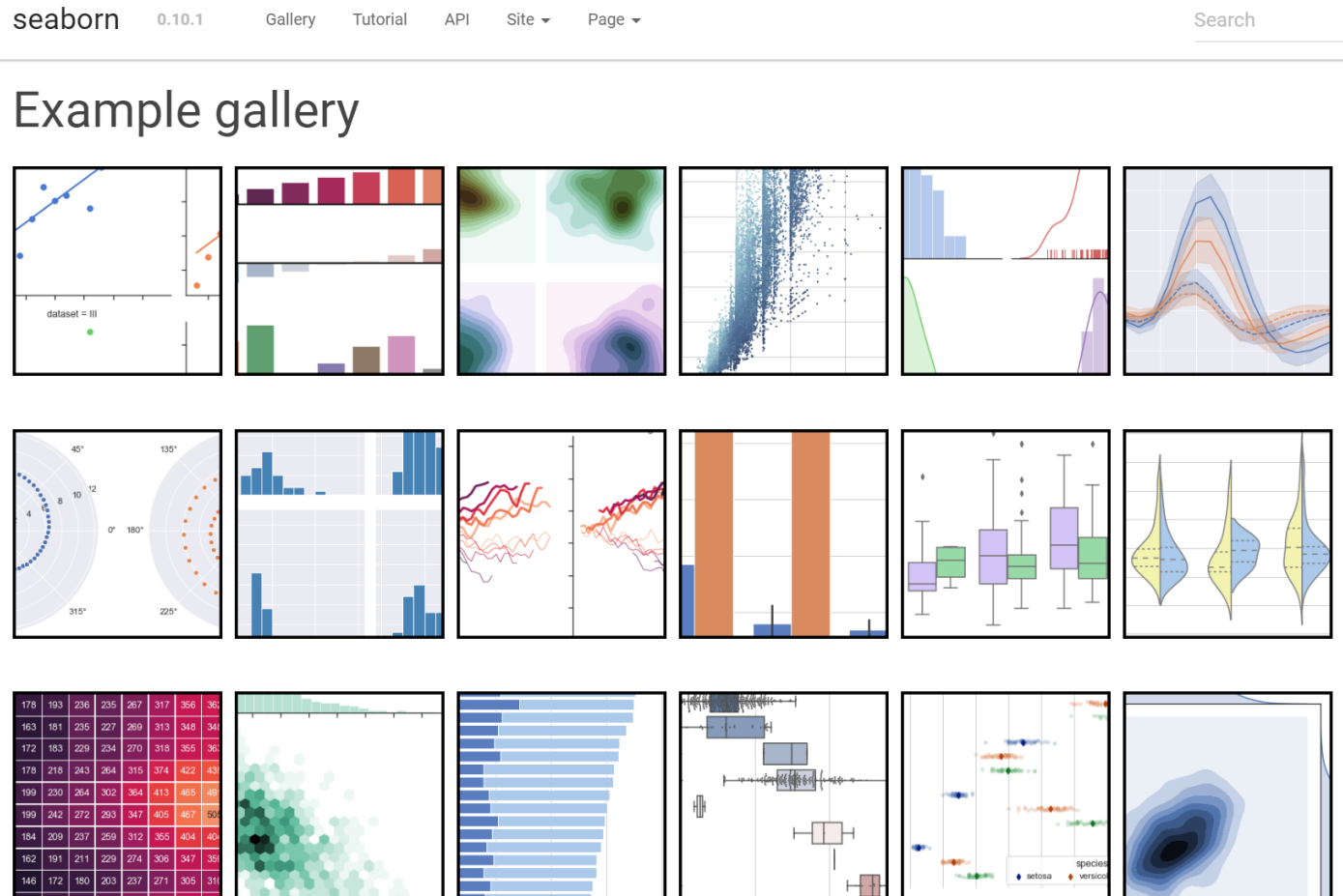
実行結果

右上のプロットパネルに図が表示されていることを確認



# Seabornで図を描画する

SeabornはMatplotlibがベースのライブラリ。より綺麗で複雑な図を簡単に描画できる。  
公式ドキュメントのギャラリー ( <https://seaborn.pydata.org/examples/index.html> ) を見よう



# Seabornによる回帰直線の描画

## コード (reg.py)

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

np.random.seed(0) # 乱数seedの設定

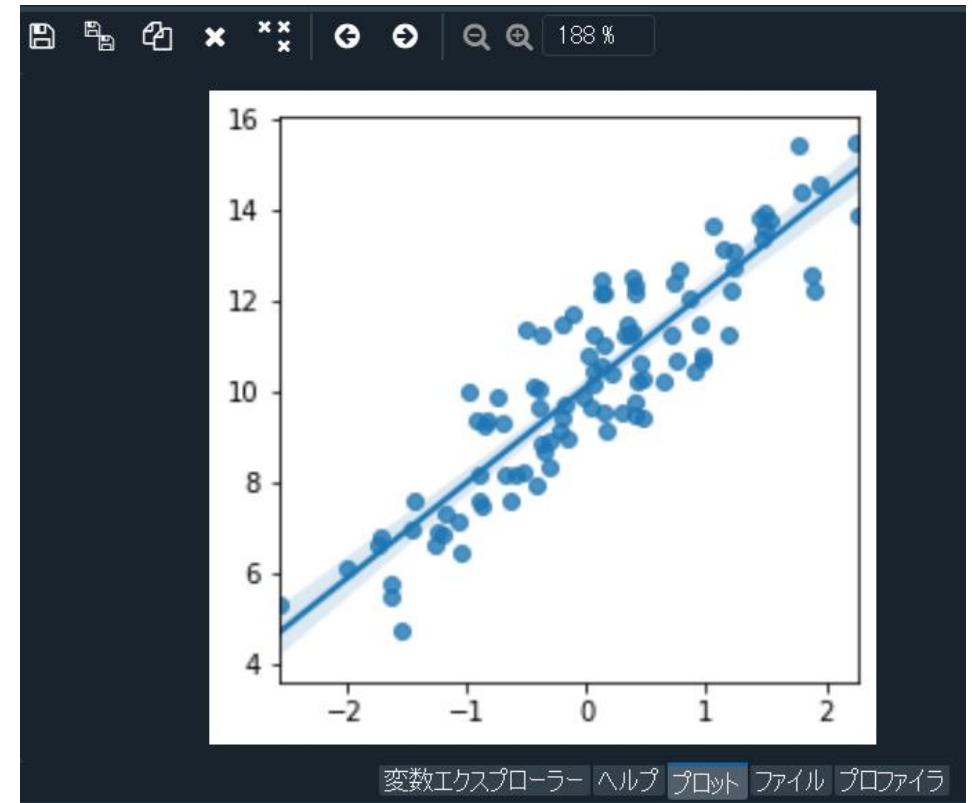
# Toyデータ (y=2*x+10+noise)の作成
n_data = 100 # データ数
x = np.random.randn(n_data)
y = 2*x + np.random.randn(n_data) + 10

# 回帰直線のplot
plt.figure(figsize=(4,4))
sns.regplot(x, y) # 回帰(regression)の実行
plt.savefig("regplot_sns.png") # 画像保存
plt.show() # 画像表示
```

実行



右上のプロットパネルに図が表示されていることを確認



Pythonファイルと同じ場所に  
**regplot\_sns.png** が保存されていることも確認

# Jupyter Notebookを使う ①

※ Jupyter = Julia + Python + R のこと

Anaconda prompt or terminalを起動して

```
$ pip install jupyter
```

を実行。その後、

```
$ jupyter-notebook
```

で起動(以下のような表示がされた後、ブラウザが開きます)。

```
(base) C:\Users\user>jupyter-notebook
[I 22:42:31.524 NotebookApp] JupyterLab extension loaded from c:\Users\user\miniconda3\lib\site-packages\jupyterlab
[I 22:42:31.524 NotebookApp] JupyterLab application directory is c:\Users\user\miniconda3\share\jupyter\lab
[I 22:42:32.170 NotebookApp] [JupyterText Server Extension] Deriving a JupyterTextContentsManager from LargeFileManager
[I 22:42:32.172 NotebookApp] Serving notebooks from local directory: C:\Users\user
[I 22:42:32.172 NotebookApp] The Jupyter Notebook is running at:
[I 22:42:32.172 NotebookApp] http://localhost:8888/?token=5ca247402532ae5b4eeb0145fef9031f5e0352f243384e2f
[I 22:42:32.173 NotebookApp] or http://127.0.0.1:8888/?token=5ca247402532ae5b4eeb0145fef9031f5e0352f243384e2f
[I 22:42:32.174 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 22:42:32.289 NotebookApp]
```

# Jupyter Notebookを使う ②

ディレクトリを移動したら右上のNewで新しいノートブックを作成（このときPython3を選択）



Quit

Logout

Files

Running

Clusters

Select items to perform actions on them.

Upload

New ▾



<input type="checkbox"/> 0 ▾	📁 / Desktop / Python_test	Name ↓	Last Modified	File size
<input type="checkbox"/>	📁 ..		数秒前	
<input type="checkbox"/>	📄 hello.py		2時間前	114 B
<input type="checkbox"/>	📄 plot_test.py		1時間前	252 B
<input type="checkbox"/>	📄 reg.py		1時間前	537 B
<input type="checkbox"/>	📄 regplot_sns.png		1時間前	15.1 kB

# Jupyter Notebookを使う ②

以下のような画面が表示される。ここで緑色枠内が1つのセルと呼ばれる。この中にコードを書いて実行する。

The image shows the Jupyter Notebook interface with several red arrows pointing to specific elements, each with a Japanese label:

- 保存** (Save) points to the File menu.
- セルの追加** (Add cell) points to the '+' icon in the toolbar.
- 名前の変更** (Change name) points to the 'Untitled (unsaved changes)' text.
- 選択しているセルの実行** (Execute selected cell) points to the 'Run' button in the toolbar.
- 選択しているセルの種類の変更** (Change cell type) points to the 'Code' dropdown menu in the toolbar.

The interface includes a top bar with the Jupyter logo, the text 'jupyter Untitled (unsaved changes)', a Python logo, and a 'Logout' button. Below this is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The toolbar contains icons for save, add, undo, redo, up/down arrows, run, stop, refresh, and a cell type dropdown set to 'Code'. The main area shows a single code cell with a green border, containing the prompt 'In [ ]: |'.



# Jupyter Notebookを使う ③

セルにはコードを書いたり、セルの種類を変えることでMarkdown記法によるコメントが書ける

Codeセル



In [1]: ▶ 1+2

Out[1]: 3

In [2]: ▶ a = 1

a

Out[2]: 1

Markdownセル  
(実行前)



**## 布団はいいぞ**  
布団の中で過ごしたい

Markdownセル  
(実行後)



**布団はいいぞ**  
布団の中で過ごしたい

In [ ]: ▶

# Markdown記法

- **Markdown**はプレーンテキスト形式で手軽に書いた文書からHTMLを生成するための言語
- 書き方は**Markdown記法 サンプル集**(<https://qiita.com/tbpgr/items/989c6badefff69377da7>)等を見るとよい
- 普段から使う場合は**Typora**(<https://typora.io/>)を使うのがおススメ。TeX形式の数式も書ける。PandocをインストールすればWordやLaTeXなどに変換可能。

## # (Rao & Ballard, 1999) モデル

### ## 観測世界の階層的予測

構築するネットワークは入力層を含め、3層のネットワークとします。網膜への入力として画像  $\mathbf{I} \in \mathbb{R}^{n_0}$  を考えます。画像  $\mathbf{I}$  の観測世界における隠れ変数、すなわち**潜在変数** (latent variable)を  $\mathbf{r} \in \mathbb{R}^{n_1}$  とし、ニューロン群によって発火率で表現されているとします (真の変数と  $\mathbf{r}$  は異なるので文字を分けるべきですが簡単のためにこう表します)。このとき、

```
$$  
\boldsymbol{I} = f(U\boldsymbol{r}) + \boldsymbol{n} \tag{1}  
$$
```

が成立しているとします。ただし、 $f(\cdot)$  は活性化関数 (activation function)、 $U \in \mathbb{R}^{n_0 \times n_1}$  は重み行列です。 $\boldsymbol{n} \in \mathbb{R}^{n_0}$  は平均0、分散  $\sigma^2$  のGaussian ノイズ項とします。

潜在変数  $\mathbf{r}$  はさらに高次 (higher-level)の潜在変数  $\mathbf{r}^h$  により、次式で表現されます。

HTMLに  
変換  
→

## (Rao & Ballard, 1999) モデル

### 観測世界の階層的予測

構築するネットワークは入力層を含め、3層のネットワークとします。網膜への入力として画像  $\mathbf{I} \in \mathbb{R}^{n_0}$  を考えます。画像  $\mathbf{I}$  の観測世界における隠れ変数、すなわち**潜在変数** (latent variable)を  $\mathbf{r} \in \mathbb{R}^{n_1}$  とし、ニューロン群によって発火率で表現されているとします (真の変数と  $\mathbf{r}$  は異なるので文字を分けるべきですが簡単のためにこう表します)。このとき、

$$\mathbf{I} = f(U\mathbf{r}) + \mathbf{n} \tag{1}$$

が成立しているとします。ただし、 $f(\cdot)$  は活性化関数 (activation function)、 $U \in \mathbb{R}^{n_0 \times n_1}$  は重み行列です。 $\mathbf{n} \in \mathbb{R}^{n_0}$  は平均0、分散  $\sigma^2$  のGaussian ノイズ項とします。

潜在変数  $\mathbf{r}$  はさらに高次 (higher-level)の潜在変数  $\mathbf{r}^h$  により、次式で表現されます。

$$\mathbf{r} = \mathbf{r}^{td} + \mathbf{n}^{td} = f(U^h \mathbf{r}^h) + \mathbf{n}^{td} \tag{2}$$

# Google Colabを使う ①

Google Colaboratory (<https://colab.research.google.com>)はJupyter Notebookをブラウザ上で使えるようにGoogleが提供しているサービス。Pythonをインストールする必要はなし。

+ コード + テキスト | 📄 ドライブにコピー

## Colaboratory とは

Colaboratory (略称: Colab) では、ブラウザから Python を記述し実行できるほか、次の特長を備えています。

- 構成が不要
- GPU への無料アクセス
- 簡単に共有


Colab は、**学生、データサイエンティスト、AI リサーチャー**の皆さんの作業を効率化します。詳しくは、[Colab のご紹介](#)をご覧ください。下からすぐに試してみることもできます。

## ▼ はじめに

ご覧になっているドキュメントは静的なウェブページではなく、**Colab ノートブック**という、コードを記述して実行できるインタラクティブな環境です。

たとえば次の**コードセル**には、値を計算して変数に保存し、結果を出力する短い Python スクリプトが含まれています。

```
[ ] seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

 86400

# Google Colabを使う ②

今回のコードをノートブックでまとめています。

[https://colab.research.google.com/github/takyamamoto/python\\_tutorial/blob/master/Python\\_tutorial.ipynb](https://colab.research.google.com/github/takyamamoto/python_tutorial/blob/master/Python_tutorial.ipynb)

## Python チュートリアル

### NumPyで数値計算

```
[ ] import numpy as np
```

```
[ ] x = np.arange(0, 1, 0.1)
    print('x:', x)
```

```
x: [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
```

```
[ ] y = 2*x + 10
    print('y:', y)
```

```
y: [10.  10.2 10.4 10.6 10.8 11.  11.2 11.4 11.6 11.8]
```

# Google Colabを使う ③

Python会の一部のブログは記事をそのままGoogle Colabで開くことができます (GitHubにアップロードしたJupyter Notebookファイル(.ipynb)はURLを修正するだけでColabで開くことができる)

(例) ICA(独立成分分析)について (<https://oumpy.github.io/blog/2020/03/ica.html>)


## 大阪大学医学部 Python会

*Now is better than never.*

### ICA (独立成分分析) について

2020-03-29(Sun) - Posted by 奥田 in 技術ブログ

Tag Statistics

 View On GitHub

 Open in Colab



Open in Colabをクリック



#### 概要

脳の機能解析に使用されるEEG(脳波計)やMEG(脳磁図)の計測結果 (つまり、電圧や磁場の時系列データ) には色ばEEGでは電源ノイズ、筋電、まばたき、などが主要なノイズとして知られています。電源ノイズのように周波でノイズを除去できますが、筋電やまばたきのような周波数が分からないものでは難しいです。そこで、計測信号)とノイズに分離するために、2000年前後から独立成分分析Independent Component Analysis(ICA)という方波計測に限らずけっこう便利なので、紹介します (音声認識分野での応用が盛んなようです)。

詳細な原理や数式は[1]にあるのでそちらを見てほしいのですが、基本コンセプトは、「ある点で観測される信号:る各信号の線形結合で表されると仮定したとき、観測した各信号を無相関にするような線形変換が存在する」といえば、「ある仮定を置き、ある条件を満たせたとき、混ざりあった信号を元の信号に分離できる」ということ

脳波計や脳磁図では脳表面の複数の点で計測を行うので、この複数の信号を使って、互いに独立な信号に分ける:信号一つ一つが、電源ノイズだったり、筋電だったり、所望の脳波だったりすることが期待されるわけです。

以下、例を実装してみます。

#### 実装

Scikit-LearnのFastICAという関数を使います。

```
[ ] import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import FastICA
```

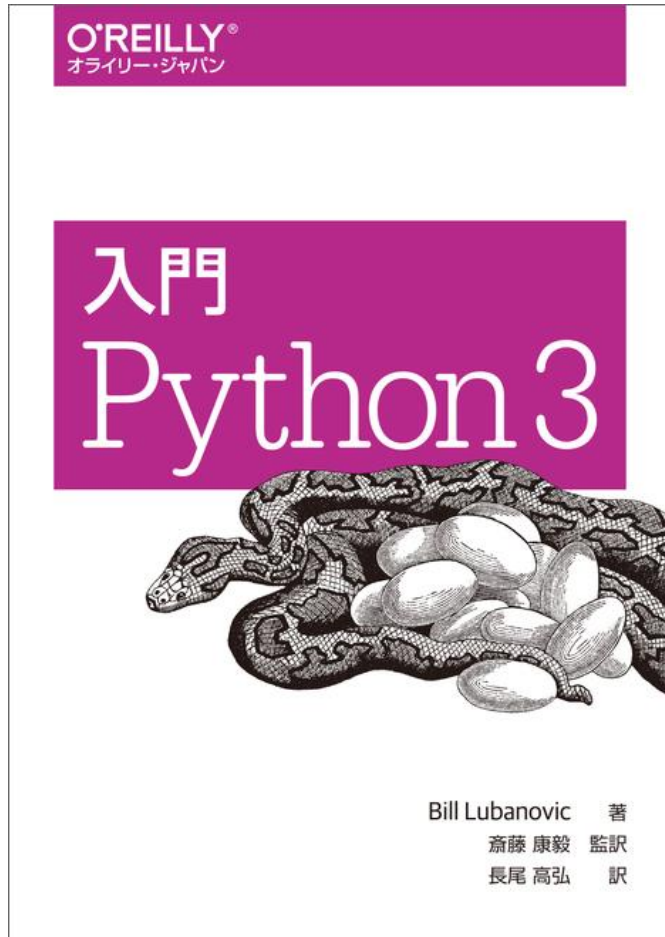
信号源を3つ設定します。例として代表的な脳波であるα波(10Hz)、電源ノイズ(60Hz)、平均0・分散1のガウスノ

```
[ ] t = np.arange(0, 1, 0.001)

s1 = np.sin(2.0 * np.pi * 10 * t)
s2 = np.sin(2.0 * np.pi * 60 * t)
s3 = np.random.normal(loc=0, scale=1, size=len(t))
```

Pythonを今後学ぶには

## Q. Pythonを勉強するのにオススメの本は？



A. プログラミング初心者の場合、初めに左のようなPythonの文法解説書は買わない方が良いでしょう。

道具の使い方は書いてるけど道具を使って具体的に何かを作成するわけではない。目的が見えないままに道具の使い方だけ知るとするのは虚無

文法は検索すれば出ることも多い  
(逆にそこそこ慣れた人は高速化や可読性のために読んだ方が良いでしょう)



# 学習における目的意識の重要性

例えばの話である、あなたが教室の中に入ると、机の上に長さ10cmほどの竹片とカッターが置いてあり、先生がその竹片からカッターで非常に細い棒状の一片を切り出すように言ったとする。

どういうつもりなのかは良くわからないが、とにかく言われた以上、そうするしかない。そしてカッターを取り上げ、何度か失敗した後、ようやくそれに成功する。

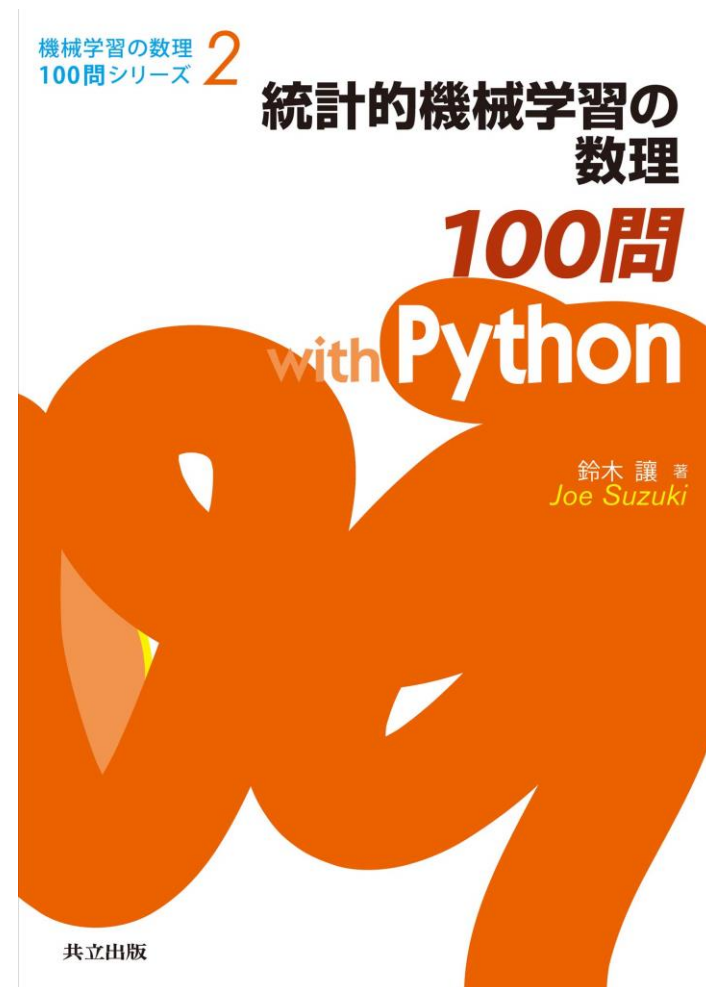
すると次に先生は、それをバーナーで燃やして黒焦げの糸を作るように言う。依然としてそれが何を意味するのかわからないが、やはりそうするしかない。ところが黒焦げの糸は作ったそばからぼろぼろくずれてしまう。くずれてしまったなら、再び前の工程に戻って最初からやり直さなければならない。

こんなことを3回も繰り返そうものなら、もうあなたの神経は忍耐の限度を越えてしまうだろう。この場合、作業の難しさもさることながら、フラストレーションの主たる源は、先生が初めに、これから作るものが初期の白熱電球のフィラメントなのだということについて、一言コメントしておいてくれなかったことにある。大学の数学の講義というのはえてしてこのようなものであり、一体何のためにそういうことを行うのかについて、あまり明確に語ってくれないのである。

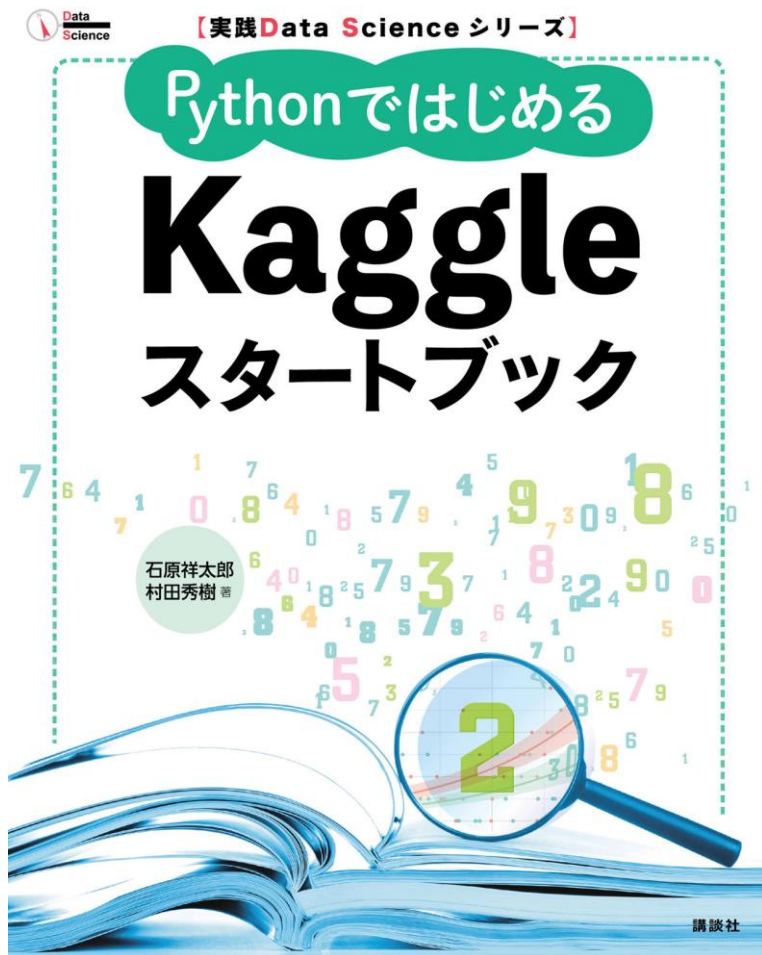
長沼伸一郎.『物理数学の直観的方法』序文より



何らかのテーマを持った本を買うのがおススメ



何らかのテーマを持った本を買うのがおススメ



バイオインフォに興味ある人

<http://kazumaxneo.hatenablog.com/entry/2020/05/02/173051>

# 今後の勉強会の予定

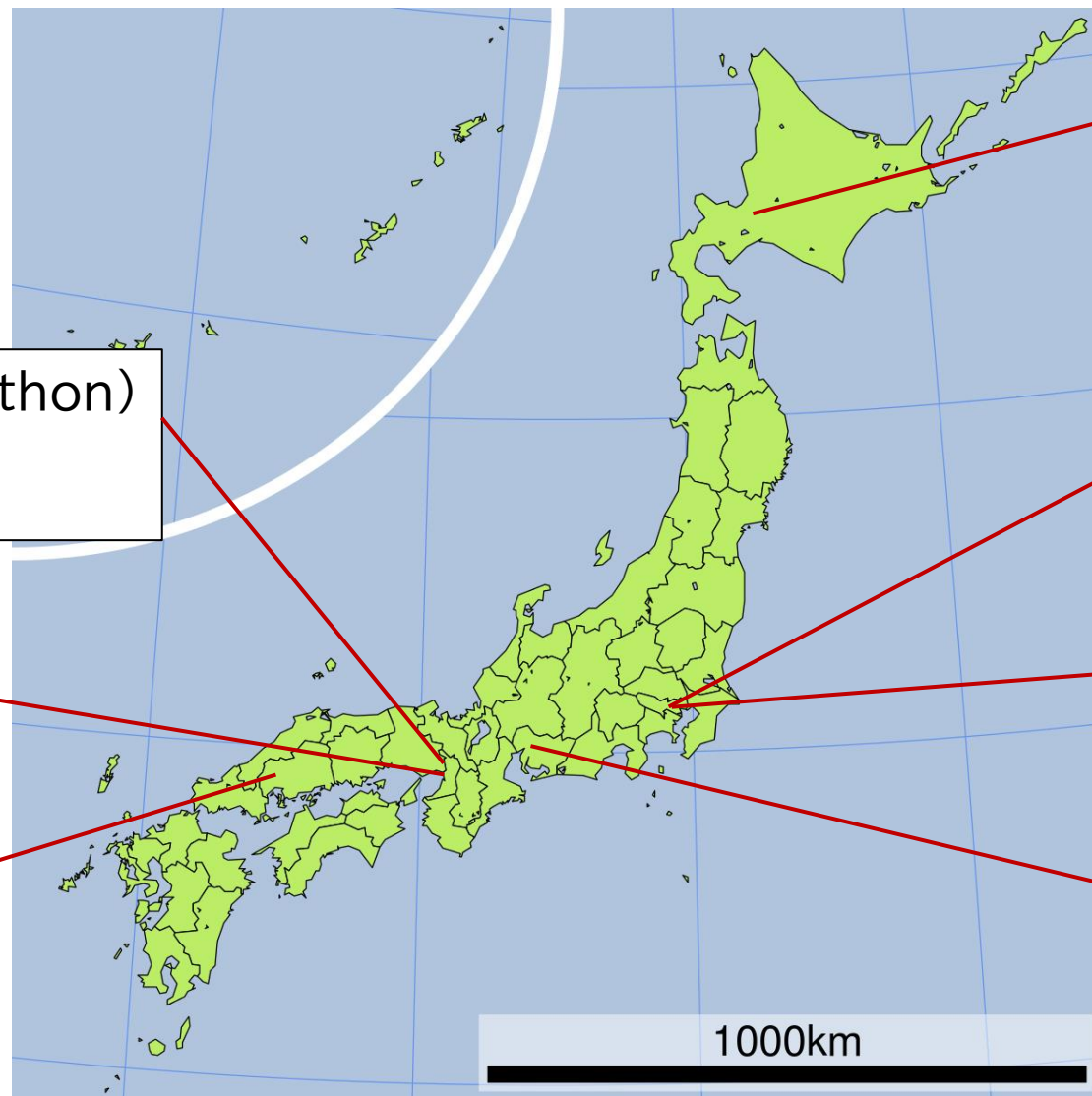
# Python会の準会員枠

- 現役医学部生 → 本会員
- その他の方でCOVID19後においても直接面会できる方  
(要は近畿圏の方) → 準会員

準会員に興味がある方は連絡してください。  
今後の勉強会は内部に限定します(資料は公開します)

理由：昨年度、外部に出過ぎて内部がおろそかになり迷走したため

# 「医学×AI（機械学習）サークル」



Sapmed IML  
(@Sapmed\_IML)

東大医学部IT研究会  
(TITAN)  
(東大, @tokyomed\_IT)

Tea Party  
(東京医科歯科大,  
@tpt\_ochanomizu)

名古屋大学医学部AI研究会  
(名大, @numedaisc)

阪医Python会 (@oumed\_python)  
阪大AI×ディカル研究会  
(@ou\_aims)

OCU医療IT研究会  
(大阪市立大, @ocu\_mit)

HIT MED  
(広島大,  
@HITMED6)