クレジット:
UTokyo Online Education Education コンピュータシステム概論 2018 小林克志



東京大学
THE UNIVERSITY OF TOKYO

CC BY NC ND

# jupyter-notebook の基本

この資料は The Python Tutorial (https://docs.python.org/3.6/tutorial/index.html#the-python-tutorial) (日本語版 (https://docs.python.jp/3/tutorial/)) および Python for Data Analysis:Wrangling with Pandas, Numpy and IPython (http://shop.oreilly.com/product/0636920050896.do)を参考に作成した。
このチュートリアルでは他のプログラミング言語を習得している方を想定し、Python の特徴などを説明する。

## このコースでの Python の位置付け

講義は大別して３つのスキルを学ぶ。
このうち最初と最後では Python 言語によるプログラミングが必要となる。

- 多様なプラットフォームで動作
  - Linux, Windows, MacOS ...
- 多様な拡張ライブラリ・モジュール
  - データ処理, 機械学習分野に強い
- インタプリタ型言語
  - プログラムソースコードを逐次実行する
    Ruby, JavaScript
  - コンパイラ型よりも実行速度は遅い:
    C, C++, Java, Object-C
    - ハードウェアの高速化であまり問題にならない
    - 過負荷はクラウド的手法（スケールアウト）によって吸収
      - コンパイル不要でコーディング-実行までの時間が短い:
    - 市場変化の速さに対応した最近の開発手法(アジャイル、DevOps)と相性が良い

# Python (参考書)

解説書は多く出版されている。 プログラミング初心者向けというのは少なく、多くが他の言語の経験を前提としている。

- オーソドックスな教科書：
  - Bill Lubanovic, "Introducing Python : Modern Computing in Simple Packages", O'Reilly Media(2016)
  - Bill Lubanovic著, 斎藤康毅監訳, 長尾高広訳, "入門 Python 3", オライリジャパン (2014)
    - 日本語訳は少し古い、Jupyter には触れてない
- データ解析向け：
  - Wes McKinney, "Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython", O'Reilly Media(2017) (http://shop.oreilly.com/product/0636920050896.do)
    - Jupyter 環境に対応
- オリジナルのチュートリアル：
  - 英語: https://docs.python.org/3/tutorial/index.html (https://docs.python.org/3/tutorial/index.html)
  - 日本語: https://docs.python.jp/3/tutorial/index.html (https://docs.python.jp/3/tutorial/index.html)
    - 他の言語の経験（みなさんの場合は Ruby?）オリジナル (https://docs.python.org/3.6/tutorial/index.html#the-python-tutorial)のチュートリアルを見ればひととおりのプログラムは書ける（はず）。

# jupyter-notebook と Python 言語の関係

jupyter-notebook と Python 言語の関係をよく聞かれるので、整理しておく:

- jupyter-notebook : Web ブラウザを利用した対話的なプログラミング環境 データは jupyter-notebook 形式(拡張子は .ipynb)で保存する
- Python 言語 : プログラミング言語 Python プログラム(.py)として扱う

前者は jupyter-notebook 環境での利用に限られるが、 後者はコンピュータに Python インタープリタさえインストールされていれば動作する。

OS のコンソールから `foo.py` を実行するには以下のようにおこなう:

```
$ python foo.py
```
あるいは
```
$ python3 foo.py
```

**練習**

jupyter-notebook のコンソールで python プログラムを実行してみる。
残念ながら ECCS の Windows では動かない、MacOS に切り替えてください。

jupyter-notebook の Home 画面で、`New->Terminal` を選択して、ターミナル、プロンプトあるいはコンソールとも呼ばれる、を起動する。
OS の CLI (Command Line Interface) 画面となるので、例えば以下を実行する:

```
$ pwd
```

GitHub Classroom のレポジトリに移動し、以下を実行する:

```
$ python hello_world.py
```

## Python プログラムの文字コード

Python の標準プログラム形式では、ヘッダ行が定義されておりプログラムで使用する文字コードや、Unix 環境では Python インタープリタのファイルパスなどを記述する。 Python の標準文字コードは Unicode である。これを明示的に書くには:

```
# -*- coding: utf-8 -*-
```

これに代えて WIndows などで使われてきた Shift-JIS を利用する場合は、先頭行を以下のように記述する:

```
# -*- coding: shift-jis -*-
```

UNIX ではプログラムスクリプトの先頭行には、そのスクリプトを読み込むインタプリタを指定する（shebang 行）。
このケースでは先頭行は例えば、以下のように 2 行となる:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

# jupyter-notebook で Python プログラムをあつかう

jupyter-notebook で Python プログラムをあつかうには大きく２種類の方法がある:

1. jupyter-notebook で Python プログラムを開く
2. jupyter-notebook 形式を Python プログラム(.py)に変換する

## 1. jupyter-notebook で Python プログラムを開く

jupyter-notebook で直接 Python プログラムファイルを開くには。 （jupyter-notebook 起動時に表示される）ファイルマネージャ画面で、 `New -> Text file` を選択すると、エディタ画面に遷移する。 エディタで .py 拡張子をもつファイル名に変更すれば良い。

## 2. jupyter-notebook 形式を Python プログラム(.py)に変換する

講義で利用している jupyter-notebook を .py としてセーブするには、 `File -> Download as -> Python(.py)` を選択すればよい。 コードセルだけがプログラム行として有効になりその他の行は # でコメントアウトされた Python プログラムファイルがダウンロードファイルに生成される。 環境によっては、.py ではなく .htmlファイルとして保存されるかもしれないが、ファイル名を変更すれば Python プログラムファイルとして利用できる。 jupyter-notebook を Python ファイルとして保存した場合、全てのコードセルの内容を一度に実行するプログラムとして保存されます。jupyter-notebook のようにセル単位の実行とはならないことに注意する必要があります。

## 講義での利用方法（＊＊重要＊＊）

講義では、とくに指定のない限り（動作する） Python プログラム形式 (.py)として提出すること。 実際には、セルで動作を確認したプログラムスクリプトをクリップボードにコピー、Python プログラムエディタにペーストする という方法が現実的と思われる。
もちろん自身が普段使い慣れているエディタを利用してもかまわない。

## 練習

Github による Python プログラムの課題提出に慣れる目的でおこなう。自信のある受講生はこの課題をスキップしてよい。 一連のスライドが終わったあとで、step-by-step でやり方を示す。

`print()` 関数で文字列 Hello U-Tokyo を印字する、Python プログラム、`hellow_u_tokyo.py` を作成せよ。 提出は指示にしたがって GitHub Classroom でおこなうこと。

```
In [1]: print("Hello World")
```

```
Hello World
```

# jupyter-notebook を 電卓代わりに使ってみる

ここから先は Python チュートリアル (https://docs.python.jp/3/tutorial/) ([英語版](https://docs.python.jp/3/tutorial/index.html)) の (https://docs.python.jp/3/tutorial/index.html))%E3%81%AE) *3.1 Python を電卓として使う* を jupyter-notebook 用にアレンジしてある。チュートリアルの内容も試しながら読みすすめること。

## code セルの実行

前に説明したように、code, Markdown の 2 種類のセルを利用する。 入力プロンプト `In[ ]` が表示された、code セルで、「実行」、すなわち　Cell->Run Cell、あるいは Run ボタンをクリック、あるいは Shift-Enter を入力、するとセル内のスクリプト（プログラム行）が実行される。
実行結果が `Out[ ]` が表示されたセルに出力される。

以下のセルを実行してみる。

In [9]:
```
10 + 10
```

Out[9]:
```
20
```

出力セルには、セルの最終行のオブジェクト、この場合実行結果、が表示される:

In [7]:
```
10+10
1+1
```

Out[7]:
```
2
```

途中経過の情報が必要であれば、`print()`関数などで標準出力に表示させる:

In [8]:
```
print(10 + 10)
1+1
```

```
20
```

Out[8]:
```
2
```

エラーも出力される:

In [5]:
```
1/0
```

```
---------------------------------------------------------------------
----------
ZeroDivisionError                         Traceback (most recent
call last)
<ipython-input-5-9e1622b385b6> in <module>()
----> 1 1/0

ZeroDivisionError: division by zero
```

## エラー出力

エラーがでても慌てず出力をよく読むこと。

# 数

## 整数型・浮動小数点型

整数(int)、浮動小数点型(float)などが使える。
自動的に変換されるので単純な計算では型を気にせず使うことができる。

## べき乗、剰余

四則演算 +-*/に加えて、べき(冪)に、 **、除算（小数点以下切り捨て）に//剰余に % が使えます。

```
In [ ]: 2**3
```

```
In [3]: 123 % 50
```

```
Out[3]: 23
```

```
In [16]: (50 - 5 * 6) / 8  # division always returns a floating point number
```

```
Out[16]: 2.5
```

```
In [4]: 4 * 3.75 - 1
```

```
Out[4]: 14.0
```

## 複素数型（課題で使います）

さらに標準で複素数型(complex)もサポートしている。虚数部は j または J 接尾辞で示す:

```
In [1]: (3 + 1j) * (3 - 1j) # 3**2 - (j**2)
```

```
Out[1]: (10+0j)
```

以下はエラーになるので注意する:

In [2]:
```
(3 + j) * ( 3 - j)
```

```
---------------------------------------------------------------------
----------
NameError                                 Traceback (most recent
call last)
<ipython-input-2-5de6974dc9da> in <module>()
----> 1 (3 + j) * ( 3 - j)

NameError: name 'j' is not defined
```

# 変数

等号(=)は変数に代入、厳密には名前とオブジェクトを束縛(binding)、するときに使う:

In [13]:
```
width = 20
height = 5
width * height
```

Out[13]: 100

未定義の変数はエラーになります:

In [1]:
```
del a   # 万一変数が定義されていたときのおまじない。
a
```

```
---------------------------------------------------------------------
----------
NameError                                 Traceback (most recent
call last)
<ipython-input-1-3f786850e387> in <module>()
----> 1 a

NameError: name 'a' is not defined
```

***変数(続き)***

## _(参考)

対話モードで最後に表示された結果は変数 _ に代入される。 電卓として使う際に有用である:

In [ ]:
```
100+100
```

In [ ]:
```
_ * 1.08    # The price including tax
```

*変数(続き)*

## 補足

同じ変数に何度でも代入できる。
変数の名前には、アルファベット(a-zA-Z)、数字のならび(0-9)、アンダースコア(_)、が使える。

- ただし最初の文字に数字は使えない
- 変数で最初の文字の _ は特別な意味に使われることがある。
  一般の変数では使わない方が無難。

# 関数の実行

以下のように簡単に関数を呼び出せる。この場合 `print()` が関数。

In [9]:
```
x = 10
print(x)
```

```
10
```

# セルで実行中の Python プログラムの停止

セル内のプログラム実行中は、プロンプト`In[*]`が表示される。
当該セルは command モードとなる。
実行中のプログラムは、カーネル割り込み "Kernel" -> "Interrupt" あるいは四角いアイコンによって停止させることができる。必ずしも停止するとは限らないが....

In [ ]:
```
# Run program with Shift-Enter
# Then try to stop program by using kernel-interrupt

# This program is not an infinite loop
i = 1000000000
while(i > 0):
    i = i - 1
```

# タブキーによる自動補完

セルの入力にあたってはタブキーによる自動補完が利用できる。jupyter-notebook の自動補完ではオブジェクト、関数、ディレクトリなどあらゆる変数の名前空間が検索対象となる。

In [ ]:
```
# Variables completion example
apple = 27
ant = 42
```

In [ ]:
```
# Type a<Tab>
a
```

In [ ]:
```
# Methods completion example
b = [1,2,3]
```

In [ ]:
```
# Type "b<Tab>"
b
```

In [ ]:
```
# Completion example on modules
import datetime
```

In [ ]:
```
# Type "datetime.<Tab>"
datetime.
```

# イントロスペクション

クエスチョンマーク(?)を前後に入力、セルを実行することで変数、関数、オブジェクト、の情報を得ることができる:

In [3]:
```
# Introspect variables
b = [1,2,3]
```

In [4]:
```
# Type "b?" then shift-enter
b?
```

関数の場合:

In [ ]:
```
# Introspect functions
# Shift-Enter (Run Cells)
print?
```

関数の情報(Docstringが定義されていれば)?あるいはソースコード??:

In [ ]:
```
# Introspect own defined functions and show its source code.
def do_nothing(a,b):
    """
    Docstring part of
    do nothing function
    """
    pass
```

In [ ]:
```
# Shiift-Enter
do_nothing?
```

In [ ]:
```
# Shiift-Enter
do_nothing??
```

よく覚えていないときはワイルドカード(*)も利用できる。

In [ ]:
```
# Wildcard with introspect
# Shiift-Enter
p*t*?
```

# jupyter-notebook の magic コマンド

code セルでは、Python スクリプト以外に、% で開始される magic コマンドもサポートされている。
magic コマンドによって、プログラムに読み込み、OS コマンドの実行などが jupyter-notebook からおこなえる。

## Python プログラムの実行、読み込み

%run プログラム.py を実行すれば、外部の Python プログラムが実行される。
また、%load プログラム.pyを実行すれば、外部プログラムを読み込む。

## カレントディレクトリの表示、移動

%pwd で現在の作業ディレクトリ(カレントディレクトリ)を表示する。
%cd ディレクトリ名 で指定されたディレクトリにカレントディレクトリを移動する。

In [ ]:
```
# Print current directory
# Shift-run
%pwd
```

## 外部コマンドの実行

!プログラム名でセルから外部プログラムを実行することができる。 もちろん実行パスに入っている必要がある:

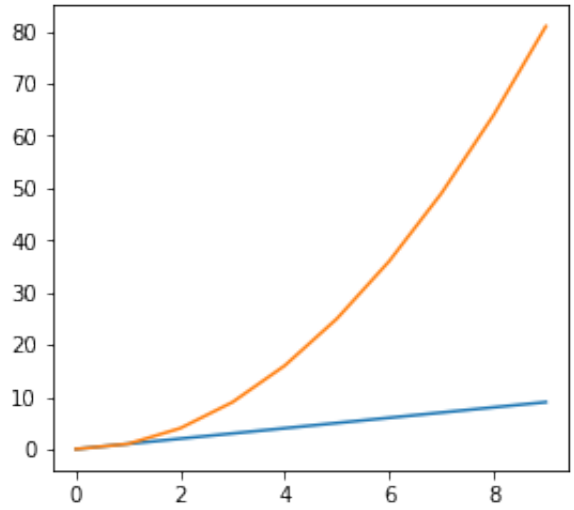
In [ ]:
```
# For Linux, MacOS and Unix
# Call external programs
# Shift-Enter
!pwd
!ls
```

In [ ]:
```
# For Windows
# Call external programs
# Shift-Enter
!cwd
!dir
```

# matplotlib との統合（課題で使います）

matplotlib はデータ可視化に広く利用されている Python モジュールである。
jupyter-notebook では matplotlib の出力をセルに表示できる。
この機能を利用するには、`%matplotlib inline` を実行する:

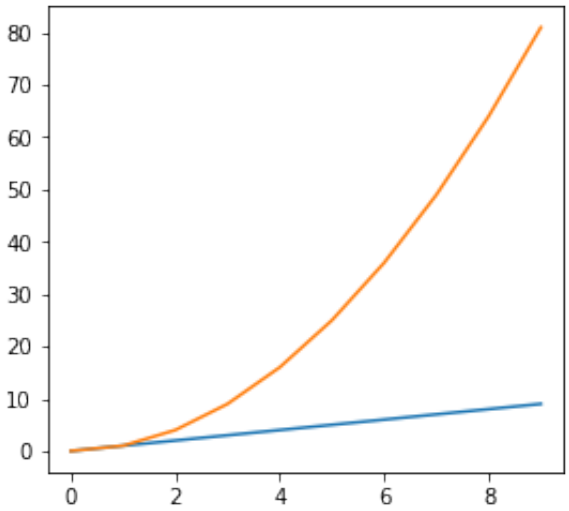
In [9]:
```
%matplotlib inline
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
x = [0, 1, 2, 3,  4,  5,  6,  7,  8,  9]
y = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  # プログラムについて教えていないことになっているので要素を書き出します。
ax.plot(x, x)                             # y = x の直線
ax.plot(x, y)                             # Default で重ね書きします。
plt.show()
```




matplotlib の出力を画像ファイルとして保存するには `matplotlib.pyplot.savefig()`を利用すれば良い。
`savefig()`は `matplotlib.pyplot.show()`の前に呼ぶこと:

In [10]:
```
%matplotlib inline
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
x = [0, 1, 2, 3,  4,  5,  6,  7,  8,  9]
y = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  # プログラムについて教えていないことになっているので要素を書き出します。
# 前のセルの描画はすでに壊れているのでもう一度呼ぶ
ax.plot(x, x)                                          # y = x の直線
ax.plot(x, y)                                          # Default で重ね書きします。
ax.set_aspect(0.1)                                     # 縦横比を指定したいとき
plt.savefig("sample.png", bbox_inches="tight")  # グラフのハミ出しを抑制するオプションをいれてある
plt.show()
```




# 練習

実行後、保存したファイルを表示してみること。