




# 第1回 Python入門

---

秋山研 M2 伊井良太





# 今日やること

---

- pyenvとAnacondaのインストール
- 対話モードやJupyterでコードを動かしてみる
- ライブラリの挙動を確認
  - Numpy
  - Matplotlib
- 演習

# 環境構築

- pyenvをインストールしてパスを通す

```
$ git clone git://github.com/yyuu/pyenv.git ~/.pyenv
$ git clone https://github.com/yyuu/pyenv-pip-rehash.git ~/.pyenv/plugins/pyenv-pip-rehash
$ echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc
$ echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc
$ echo 'eval "$(pyenv init -)"' >> ~/.bashrc
$ source ~/.bashrc
```

- Anacondaのインストール

```
$ pyenv install -l | grep anaconda3
$ pyenv install anaconda3-4.1.0
$ pyenv global anaconda3-4.1.0
```



# 対話モード

- ターミナル上でpythonと入力
- jupyter notebookの起動
- jupyter labのインストール



```
$ pip install jupyterlab  
$ jupyter serverextension enable --py  
$ jupyterlab --sys-prefix
```



# 算術計算

---

- 除算
  - Python3系では整数どうしの除算の結果は浮動小数点
  - Python2系では整数どうしの除算の結果は整数
- 累乗
  - \*\*で表す
  - Ex)  $3 ** 2 = 9$
  - `pow(3, 2)`でも計算可能



# 制御文・関数（：で終わる）

---

- if文
  - 条件文を追加するときはelif
  - 条件に当てはまらないときはelse
- for文
  - 反復処理 `for i in range(len(リスト)):`
- break
  - 一番内側のforやwhileを抜ける
- pass
  - 処理がないときに利用
- 関数定義
  - 関数の宣言はdef



# クラス

- コンストラクタ
  - インスタンスが生成されるときに自動的に呼び出されるメソッド
  - 定義するときは\_\_init\_\_
- メソッドの第一引数には自分自身を表すselfを書く

```
class クラス名:  
    def __init__(self, 引数, ...): # コンストラクタ  
        ...  
    def メソッド名1(self, 引数, ...): # メソッド1  
        ...  
    def メソッド名2(self, 引数, ...): # メソッド2  
        ...
```

```
インスタンス = クラス名(引数) # インスタンス生成  
インスタンス.メソッド名()    # メソッド呼び出し
```



# リスト






---

- appendメソッド
  - 引数に追加したい要素の値を入れて、リストの最後尾に新たな要素を追加できる
- in演算子
  - “要素 in リスト”で戻り値はTrue or False
  - リストにある特定の要素が含まれているかどうか検索できる
- len関数
  - 引数にリストを入れて、そのリスト内の要素の数をカウントできる



# スライス

- 配列の中で特定の範囲の要素を抜き出す
- start:end:step
  - startからend-1番目までstepおきの要素を取得

要素					
index	0	1	2	n-2	n-1
	-n	-n-1	-n-2	-2	-1


- 多次元配列の場合は、カンマで区切って行と列をそれぞれ指定



# ディクショナリ

---


- keyとvalueをコロンで区切ったペア
- 辞書型には順序がない
- keysメソッド
  - 辞書内のkeyをすべて取得することができる
- valuesメソッド
  - 辞書内のvalueをすべて取得することができる
- itemsメソッド
  - 返り値は各要素について(key, value)のペアを取得することができる



# 複数の値を保存するデータ型

---

- リスト[]
  - 直接変更、要素追加、要素削除などが可能
- タプル()
  - タプルを定義したあと、変更や追加、削除ができない
  - タプル自体をリストに追加することは可能
- 辞書{}
  - 要素の順序が管理されていなくて、個々の値に一意的なキーで管理する



# Numpyによるデータ生成

---

- `numpy.arange([start, ]end, [step, ])`
  - endの値は含まない配列
  - startとstepの値は指定しなくてもよい
- `numpy.linspace(start, end, num=50)`
  - endの値を含む配列
  - startとend、stepの値を指定する

# Numpyでのaxis指定

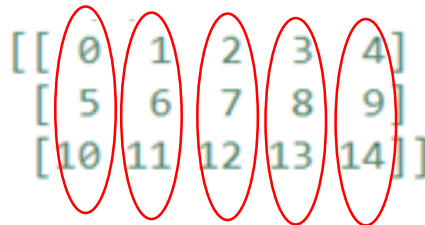
- 軸を固定した演算
  - axis=0を付けると列ごとに、axis=1を付けると行ごとに演算

```
import numpy as np
```

```
a = np.arange(15).reshape(3, 5)  
print(a.shape)  
# (3, 5)
```

```
print(a)  
# [[ 0  1  2  3  4]  
#   [ 5  6  7  8  9]  
#   [10 11 12 13 14]]
```

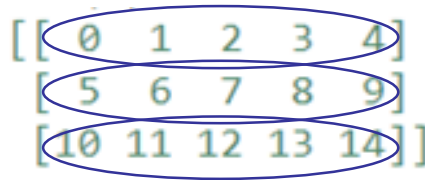
```
print(np.sum(a))  
# 105
```



```
[[ 0  1  2  3  4]  
 [ 5  6  7  8  9]  
 [10 11 12 13 14]]
```




```
print(np.sum(a, axis=0))  
# [15 18 21 24 27]
```



```
[[ 0  1  2  3  4]  
 [ 5  6  7  8  9]  
 [10 11 12 13 14]]
```



```
print(np.sum(a, axis=1))  
# [10 35 60]
```



# Numpyのshape

---

- $(X, )$ や $(X, 1)$ について
  - 1次元配列のshapeは $(X, )$
  - 1次元配列を転置した縦ベクトルの場合のshapeは $(X, 1)$

```
import numpy as np

a = np.array([1,2,3,4,5])
print(a.shape)
# (5,)

b = np.array([[1], [2], [3], [4], [5]])
print(b.shape)
# (5, 1)
```

# ブロードキャスト

```
import numpy as np

a = np.array([[8], [4]])
b = np.array([3])
c = np.array([[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]])

print( a + b + c)
#[[12 13 14]
#  [11 12 13]]

#[[18 19 20]
#  [17 18 19]]
```

行と列方向に要素をコピーして拡張

$(2, 1) \xrightarrow{a} (1, 2, 1)$

$(1,) \xrightarrow{b} (1, 1, 1)$

$(2, 2, 3) \xrightarrow{c} (2, 2, 3)$

$(1, 2, 1) \xrightarrow{a} (2, 2, 3)$

$(1, 1, 1) \xrightarrow{b} (2, 2, 3)$

$(2, 2, 3) \xrightarrow{c} (2, 2, 3)$

次元数が異なるときは  
shapeの先頭に1が入る

shapeのそれぞれの次元において  
最も要素数の多いものに  
合わせられる

# Ex) 2値化処理

- C言語

```
for(y = 0; y < height; y++) {  
    for(x = 0; x < width; x++) {  
        if(array[y][x] >= 100) {  
            array[y][x] = 255;  
        }  
        else {  
            array[y][x] = 0;  
        }  
    }  
}
```

Numpyを使うことで  
短いコードで  
かつ高速な処理速度で  
実装できる

- リスト

```
for y in range(0, height):  
    for x in range(0, width):  
        if array[y][x] >= 100:  
            array[y][x] = 255  
        else:  
            array[y][x] = 0
```

- Numpy

```
array[array<100] = 0  
array[array>=100] = 255
```





# matplotlib.pyplotによるグラフの描画

- プロット位置をリストで与える
  - プロット位置 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ に対してxlistとylistを用意
  - 各リスト要素の順番が大事
  - $xlist = [x_1, x_2, \dots, x_n]$        $ylist = [y_1, y_2, \dots, y_n]$
- 「plt.plot」で位置をプロット
- 「plt.savefig」で図をファイルへ出力
- 日本語を使いたいときは凡例だけpropで、その他はfontproperties



# 演習（１）

---

- $5 \times 5$ の乱数行列を作成して最大値を1、最小値を0になるように正規化せよ。
  - `numpy.random.rand()`: 0～1の一樣乱数を生成
  - `numpy.min()`: 最小値 / `numpy.max()`: 最大値
- 1～10の範囲の乱数を生成し、そのベクトル内において与えられたvalueに最も近い値を出力せよ。
  - `numpy.abs(x)`: 配列xの絶対値を算出
  - `numpy.argmin(x)`: 配列xの最小値要素のインデックスを取得

# 演習（２）

- 乱数で自然数を30個生成し、その配列の中で上位5つの値を取得し、昇順に並べよ。
  - `numpy.random.randint(a, b, n)`:  $a \sim b-1$ の整数を $n$ 個生成
  - `np.argsort()`: ソート結果の配列のインデックスを返す
- $10 \times 5$ の乱数行列を作り、各列ごとに値を平均0、標準偏差1になるように正規化した行列を作成せよ。
  - `numpy.mean()`: 平均 / `numpy.std()`: 標準偏差