Week2 Pythonによる科学計算（Numpy）

講師 渡部泰樹

Numpy（Numerical Python）とは

ベクトルや行列に対する演算を効率良く行うために開発された「ライブラリ」

Numpyでは、np.ndarrayというN次元配列が用意されており、Pythonのリストと比べて行列演算が高速かつ容易に行うことができる

データサイエンスとNumpyの繋がり

データサイエンス：膨大なデータ量に対して一括で演算を施す必要がある

Numpy：多次元配列に対して四則演算など**高速な計算**が可能、forループを用いずに配列を**一括で処理**できる

＜データサイエンスのフロー＞

最初のフェーズに用いられることが多い

データサイエンスとNumpyの繋がり

各カラムの平均値や標準偏差を求める

→高次元配列を扱えるNumpyが有効

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	width	height	curb-weight
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824

Numpyの１次元配列

Numpyの性質

ユニバーサル

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

b = np.array([2, 2, 3, 6, 0])

a * b = [2 4 9 24 0]

a ** 2 = [1 4 9 16 25]

np.sin(a)

= [0.84147098 0.90929743 0.14112001 -0.7568025 -0.95892427]

→ 要素ごとに計算する：ユニバーサル

Numpyの性質

ブロードキャスト

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

3 * a = [3 6 9 12 15]

インデクシング

0	1	2	3	4	5	6	7	8
2	9	9	7	9	2	4	5	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

a = np.array([2, 9, 9, 7, 9, 2, 4, 5, 8])

a[[1, 3, 0, 6]] = [9 7 2 4]

複数のインデックスをリストとして指定することで、
複数の値を取得できる

Numpyの性質

スライシング

a = np.arange(30) # 0~29を要素に持つnp.ndarray

a[1: 13: 2] = [1 3 5 7 9 11]　　　(a[start: end: step])

0 1 2 3 4 5 6 7 8 9 10 11 12 13

[0 1 2 3 4 5 6 7 8 9 10 11 12 13 …]

end**未満**
なので注意！

Numpyの２次元配列

Numpyの性質

Reshape

a = np.array([1, 1, 2, 3, 5, 8])

a.reshape(3, 2) = [[1 1]
[2 3]
[5 8]]

a.reshape(3, -1) = [[1 1]
[2 3]
[5 8]]

$$3\left\{\overbrace{\begin{matrix}[[1 & 1] \\ [2 & 3] \\ [5 & 8]]\end{matrix}}^{2}\right.$$

3x2行列

Numpyの性質

axisと集約関数

a = np.array([88, 78, 76, 98, 88, 100, 64, 78, 77, 89, 67, 78]).reshape(4, 3)

集約関数：全体に対して一つの値を返す

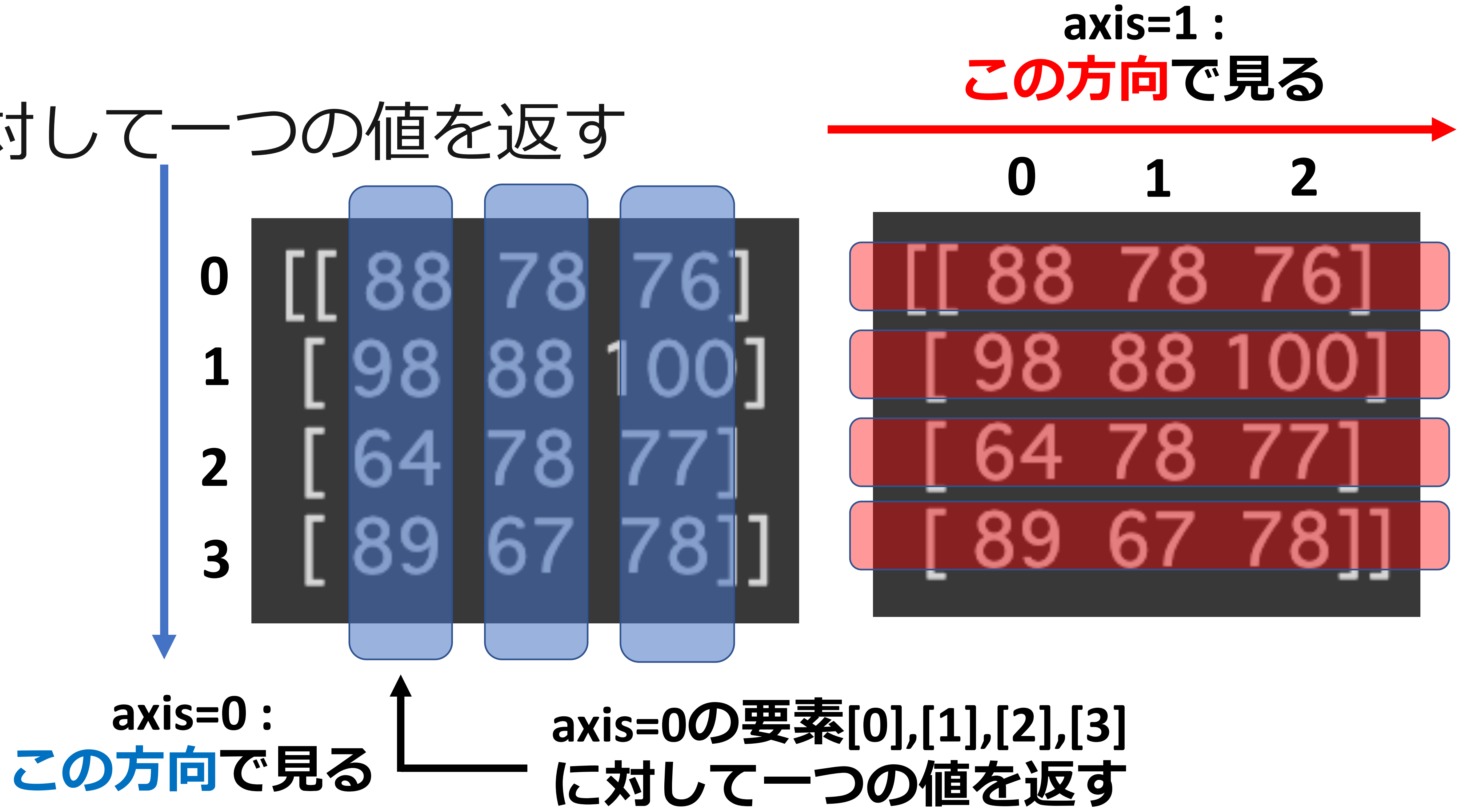
np.max(a) = 100

np.max(a, axis=0)

= [98 88 100]

np.max(a, axis=1)

= [88 100 78 89]



Numpyの2次元配列の基本操作

a = np.arange(12).reshape(3, 4)

a[0][1] = a[0, 1] = 1

axis=0 axis=1 axis=0 axis=1

・**axis=0の0行目と2行目をとる**

a[[0, 2]] = [[0 1 2 3]

axis=0

[8 9 10 11]]

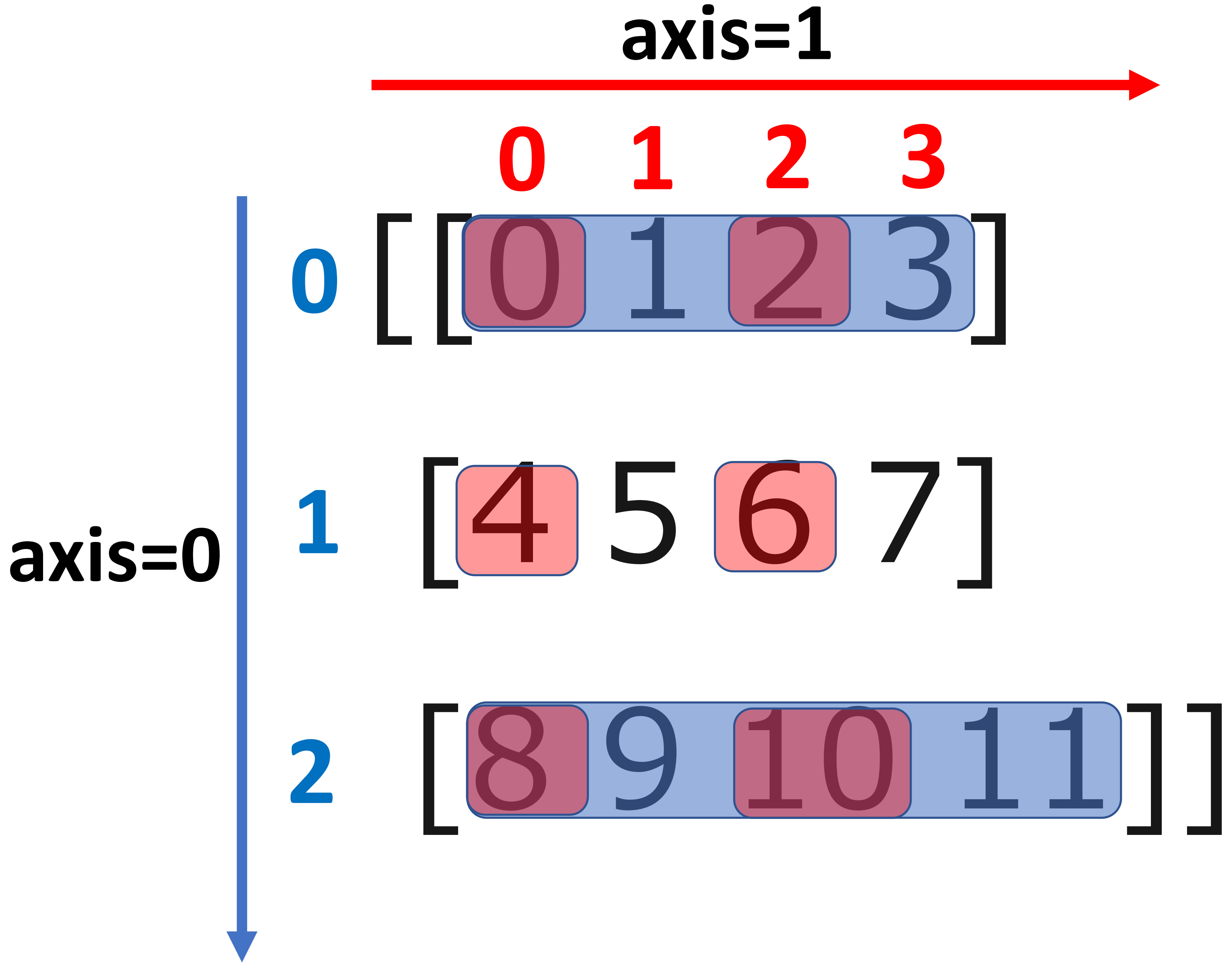
・**axis=1の0列目と2列目をとる**

a[:, [0, 2]] = [[0 2]

axis=1

[4 6]

[8 10]]



Numpyの2次元配列の基本操作

axis=0の0行目と2行目、axis=1の1列目をとる

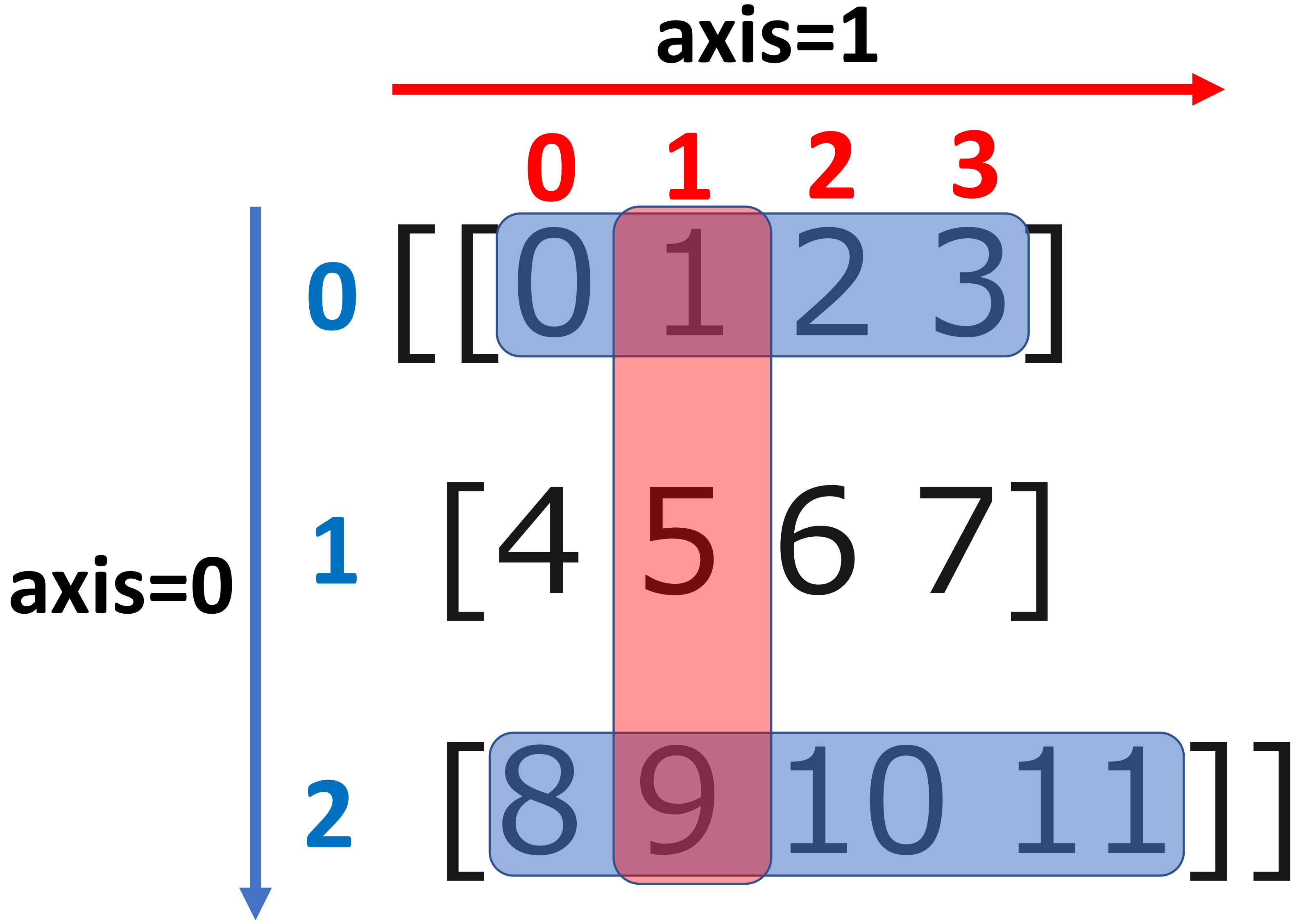
・二次元配列になっている

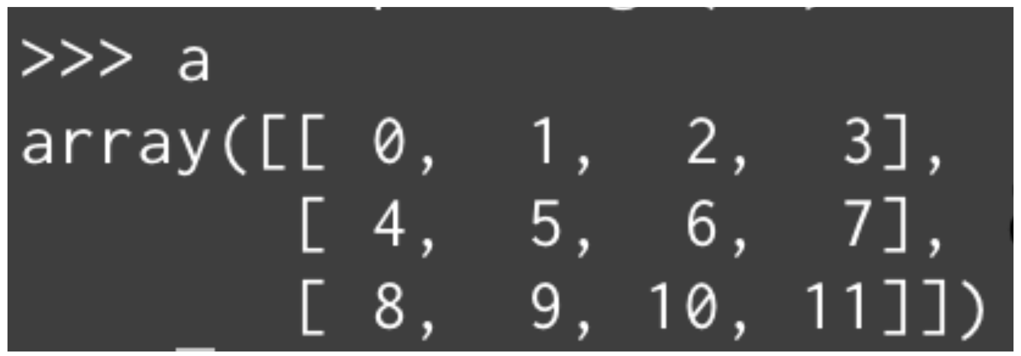
a[[[0], [2]], 1] = [[1]
                              [9]]

縦方向に指定して
あげることが
重要！！！

・一次元配列になってしまう

a[[0, 2], 1] = [1 9]





```
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Numpyの2次元配列の基本操作

axis=0の2行目、axis=1の1列目と3列目をとる

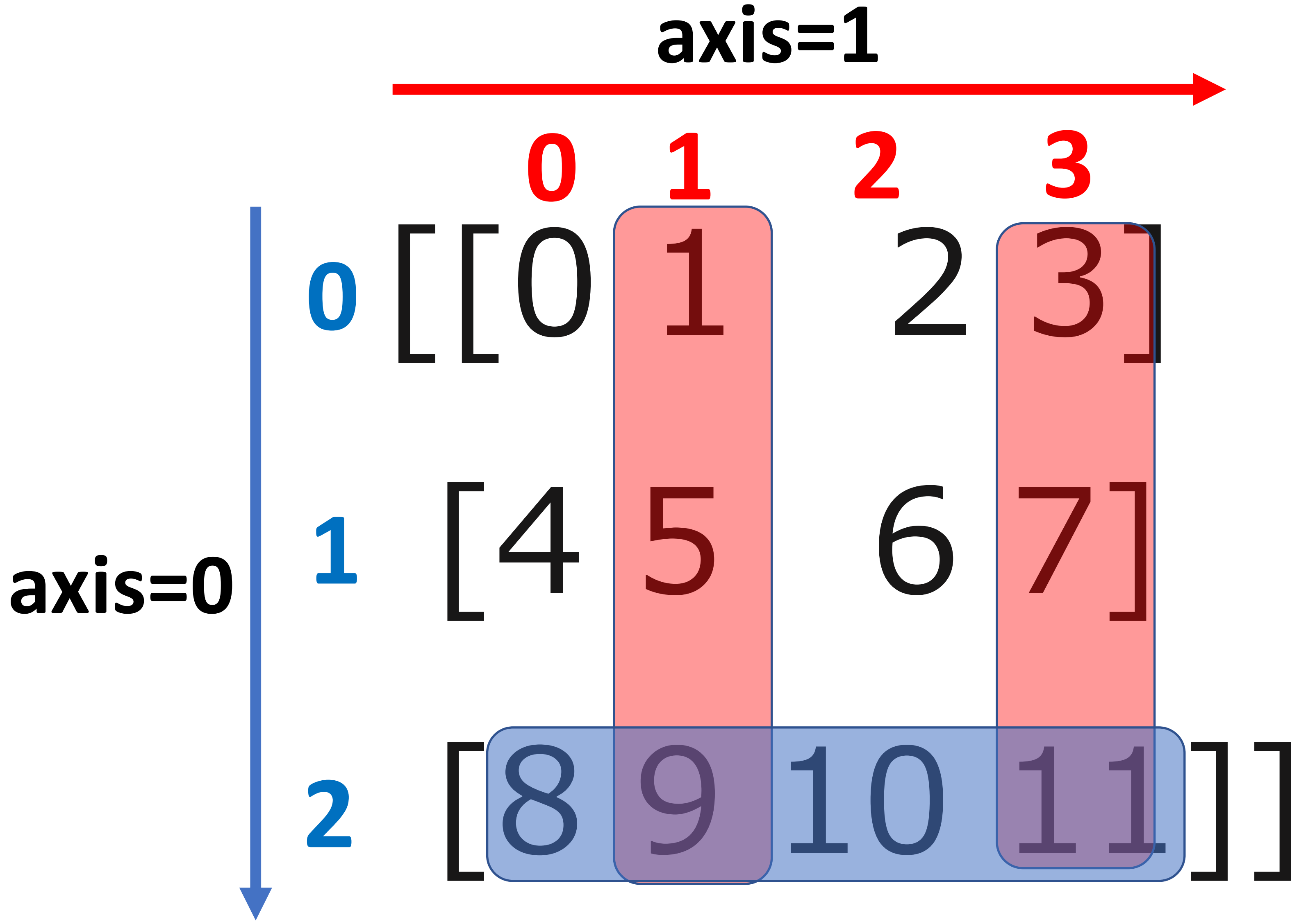
・二次元配列になっている

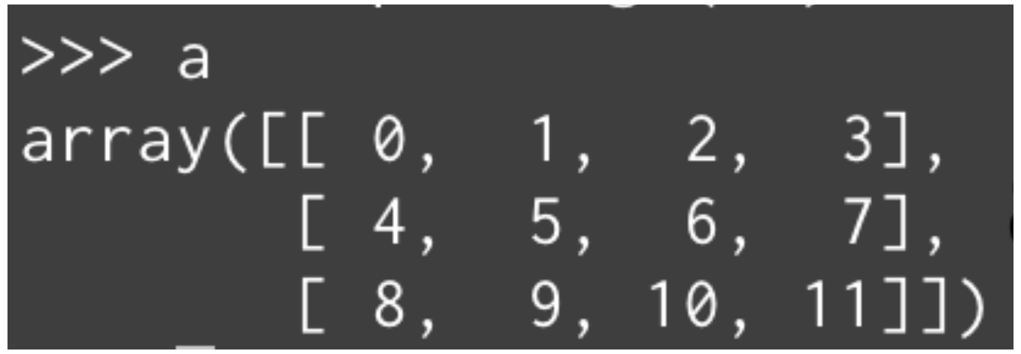
a[2, [[1, 3]]] = [[9 11]]

横方向に指定して
あげることが
重要！！！

・一次元配列になってしまう

a[2, [1, 3]] = [9 11]





```
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Numpyの2次元配列の基本操作

axis=0の0行目と2行目、axis=1の1列目と3列目をとる

・二次元配列になっている

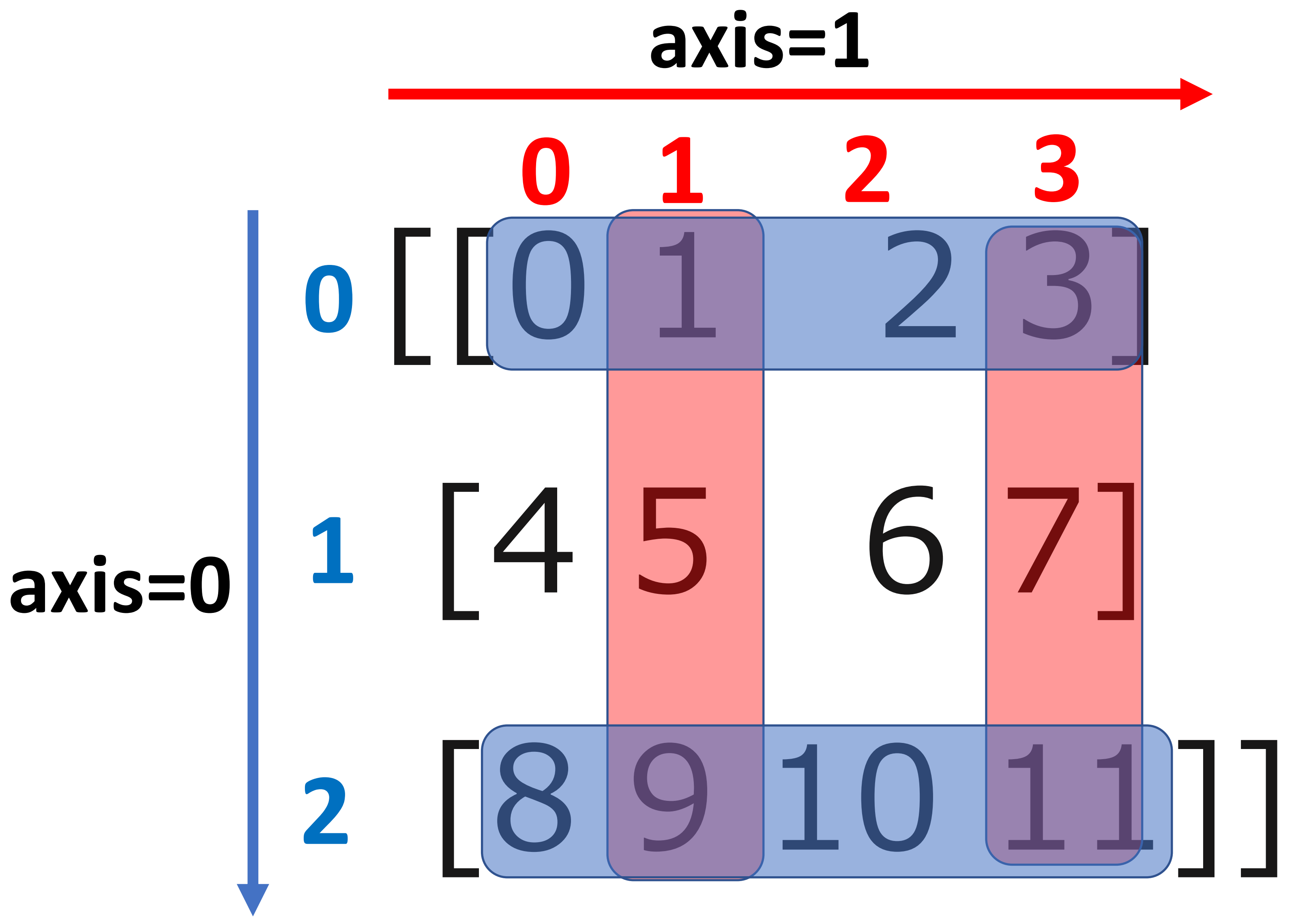
a[[[0], [2]], [[1, 3]]] = [[1 3]
[9 11]]

縦方向に指定　横方向に指定

・一次元配列になってしまう

a[[0, 2], [1, 3]] = [1 11]

→ **a[0, 1]とa[2, 3]を取ってきてしまう**




```
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Numpyの2次元配列の基本操作

axis=0の偶数行目、axis=1の奇数行目をとるには？

・a = np.arange(100).reshape(10,10)

・idx1 = np.array([0,2,4,6,8])

・idx2 = np.array([1,3,5,7,9])

axis=1

axis=0

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

a[np.ix_(idx1, idx2)]

a[[[idx1[0]], [idx1[1]], [idx1[2]], [idx1[3]], [idx1[4]]], [idx2]]

axis=0　　　　　axis=1

※スライスを使用する方法もある

Numpyの2次元配列の基本操作

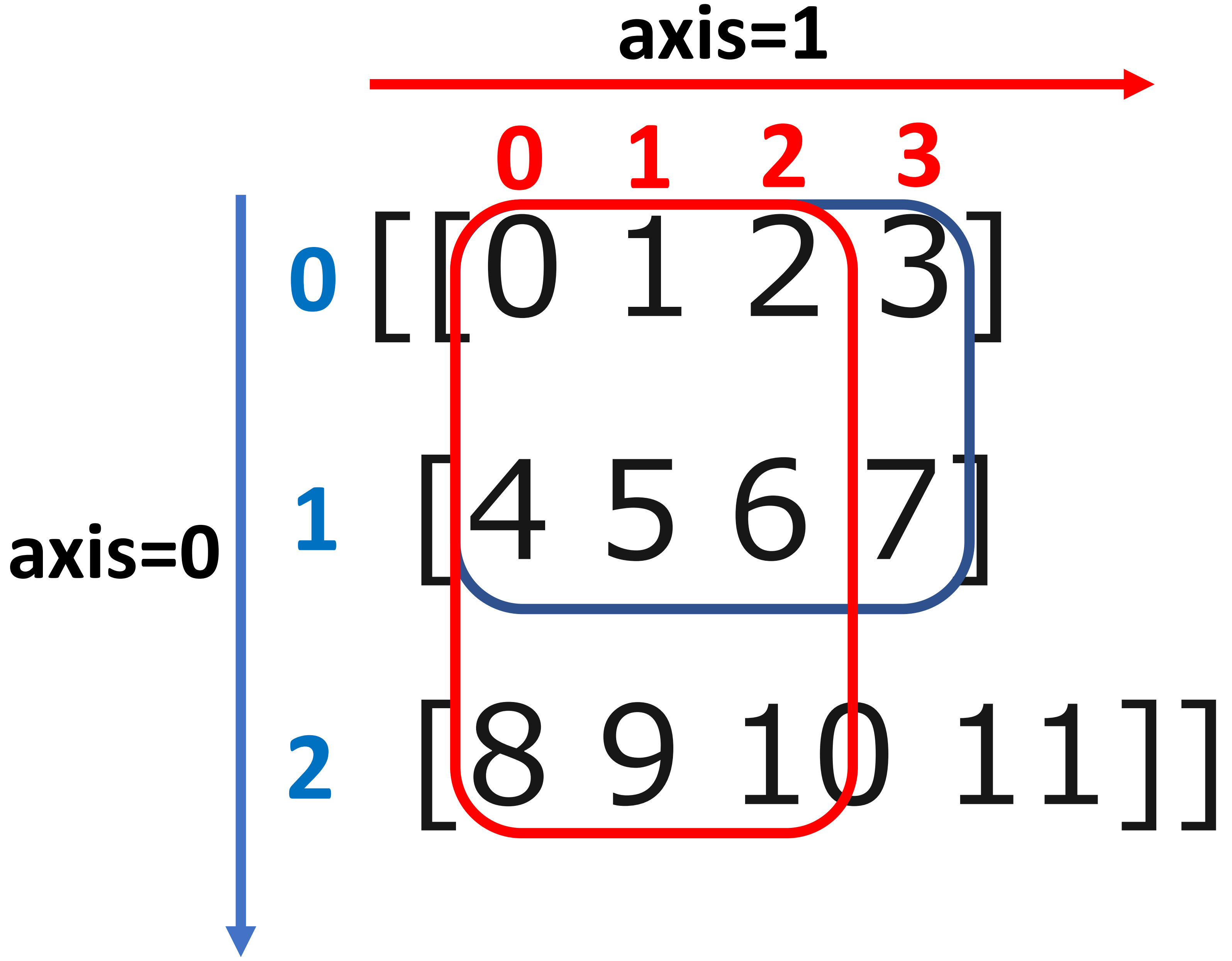
スライシング

・axis=0の0行目から1行目までとる

a[0:2] = [[0 1 2 3]
axis=0
[4 5 6 7]]

・axis=1の0列目から2列目までとる

a[:, 0:3] = [[0 1 2]
axis=0 axis=1
[4 5 6]
[8 9 10]]



Numpyの2次元配列の基本操作

axis=0の0行目から1行目、axis=1の0列目から2列目まで取る

$$a[\underbrace{0:2}_{axis=0},\ \underbrace{0:3}_{axis=1}] = \begin{matrix}[[0\ 1\ 2] \\ [4\ 5\ 6]]\end{matrix}$$

axis=1

$$\textbf{axis=0}\quad \begin{array}{c|l} & \quad 0\quad 1\quad 2\quad 3 \\ 0 & [[\boxed{0\ 1\ 2}\ 3] \\ 1 & \ [\boxed{4\ 5\ 6}\ 7] \\ 2 & \ [8\ 9\ 10\ 11]] \end{array}$$

Numpyの2次元配列の基本操作

ブロードキャスト

※一次元配列のブロードキャストとは、少しニュアンスが異なる

A = [[1 2 3]
[4 5 6]
[7 8 9]]
x = [1 2 3]

np.matmul(A, x)

$$= \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 14 \\ 32 \\ 50 \end{pmatrix}$$

np.matmul(x, A)

$$= \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

$$= \begin{pmatrix} 30 & 36 & 42 \end{pmatrix}$$

(参考) 線形代数

- 転置：Aの(i,j)成分と(j,i)成分を入れ替えたもの

$$A = \begin{pmatrix} 1 & 8 & 4 \\ 7 & 1 & 3 \\ 2 & 2 & 3 \end{pmatrix} \quad A^T = \begin{pmatrix} 1 & 7 & 2 \\ 8 & 1 & 2 \\ 4 & 3 & 3 \end{pmatrix}$$

- 逆行列：Aと行列積をとると単位行列になるもの

$$A = \begin{pmatrix} 1 & 8 & 4 \\ 7 & 1 & 3 \\ 2 & 2 & 3 \end{pmatrix} \quad A^{-1} = \begin{pmatrix} 0.04 & 0.2133 & -0.266 \\ 0.2 & 0.066 & -0.33 \\ -0.16 & -0.1866 & 0.733 \end{pmatrix}$$

$$AA^{-1} = A^{-1}A = I$$

単位行列 $I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

(参考) 線形代数

- 行列式

$$
\begin{aligned}
|A| = \begin{vmatrix} 1 & 8 & 4 \\ 7 & 1 & 3 \\ 2 & 2 & 3 \end{vmatrix} &= 1 \times 1 \times 3 + 2 \times 8 \times 3 + 4 \times 7 \times 2 \\
&-4 \times 1 \times 2 - 1 \times 3 \times 2 - 8 \times 7 \times 3 \\
&= -75
\end{aligned}
$$

詳しくは「サラスの公式」を参照

（参考）線形代数

- ノルム

$$L_2 ノルム：\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

$$L_1 ノルム：|x_1| + |x_2| + \cdots + |x_n|$$

$$L_\infty ノルム：\sqrt[p]{x_1^p + x_2^p + \cdots + x_n^p} \fallingdotseq |x_k|$$

絶対値が最大の成分の絶対値