

# 第3回 Pythonによる科学計算 (Numpy)

---

# 第3回 Pythonによる科学計算 (Numpy)

## 自己紹介



熊澤 倫之 (くまざわ ともゆき)



所属

国内メーカー勤務

受講歴

GCI winter 2023

講義前に一言

初学者の方にも理解がしやすいよう、  
極力わかりやすく、特に理解いただきたい点、いまは知っておくだけで大丈夫な点など、強弱をつけた講義を心がけたいと考えています。

本日の講義の内容を 1 ページで解説

(本日の講義でできるようになること)  
Numpyと呼ばれるライブラリ (ツールのようなもの) の基本的な使用ができるようになる

### 1. Numpyとは何か、その特徴

### 2. 1次元配列

2-1. 計算の基本 (ユニバーサル関数・ブロードキャスト・集約関数)

2-2. 中身のデータ参照方法 (インデクシング)

### 3. 2次元配列

3-1. 計算の基本 (ユニバーサル関数・ブロードキャスト)

3-2. 縦軸・横軸の概念 (axis・集約関数)

3-3. 中身のデータ参照方法 (インデクシング)

# Numpy (Numerical Python) とは



Numpy : Pythonで複雑な科学計算を行うためのライブラリ

## Numpyの特徴

1. N次元配列を効率的に扱うことができる
2. for文なしで複雑な計算ができる (次ページ)
3. 計算が高速 (Numpy内部は高速なC言語が主に使われている)
4. 科学計算のための豊富な関数がある

ライブラリ : プログラムを書く際に便利なツールや機能を集めたもの

1次元配列

```
[0, 1, 2, 3, 4]
```

2次元配列

```
[[70, 95, 80],  
 [34, 42, 50],  
 [74, 37, 65]]
```

# Numpy(Numerical Python)とは

Numpy: Pythonで複雑な科学計算を行うためのライブラリ

配列は前回講義で扱ったリストととても似ているが違いあり  
リストとの違いの例)ユニバーサル関数によって、for文なしで計算ができる

Numpyを使って計算する場合

a : [1, 2]

b : [3, 4]

→ [4, 6]

input

```
sum = a + b
```

} 少ない記述で済む！

リストのまま計算する場合

input

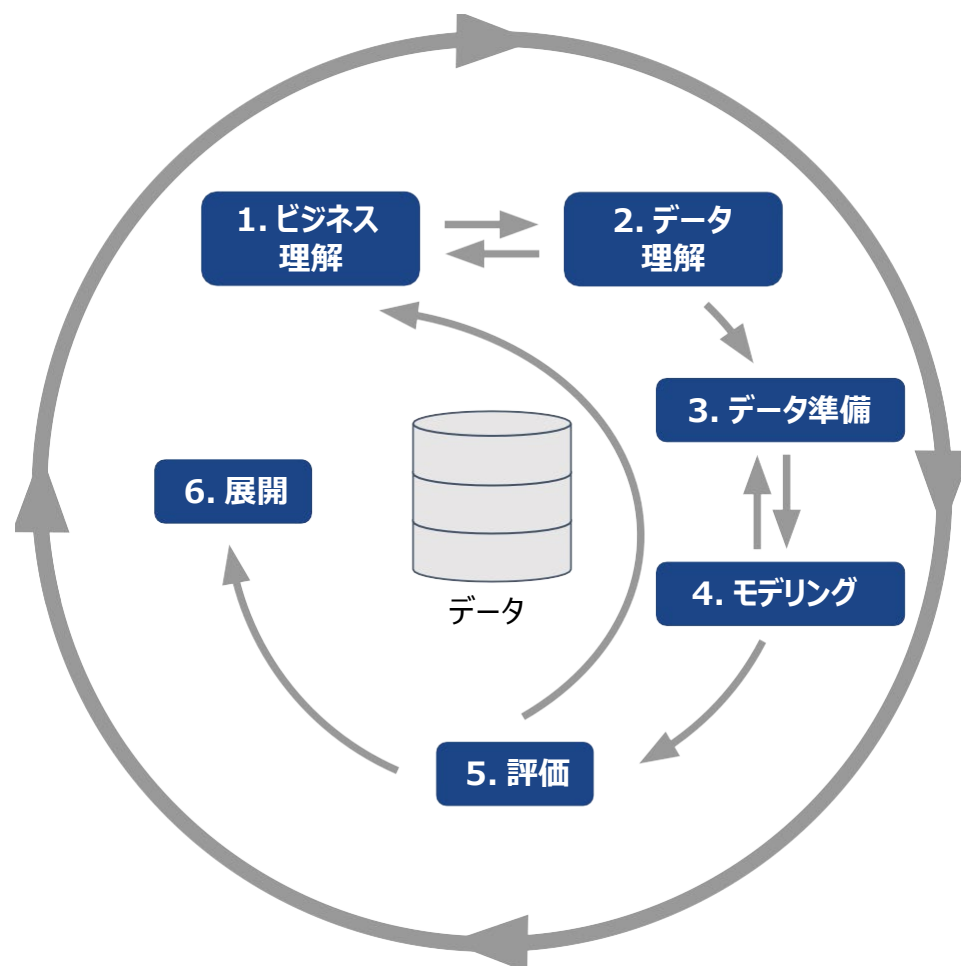
```
sum = []  
for i in range(len(a)):  
    sum.append(a[i] + b[i])
```

}

# 第3回 Pythonによる科学計算 (Numpy) Numpy (Numerical Python) とは



データマイニング：大量のデータから有用な知識や規則性を発見・抽出するプロセス



なぜNumpyを学習する必要があるか

**データマイニング・データ分析にはデータ準備が不可欠であるため**

データ準備のために、データを適切な形式に整える・・・  
データを分析やモデリングに適した形式に 整形するにはNumpyやPandas（次回講義）のようなライブラリが必要になる

# 第3回 Pythonによる科学計算(Numpy)

## Numpy(Numerical Python)とは



### Numpyの始め方① インポート方法

慣習により、Numpyは**np**と略して扱います。

以下のようにimportしてください。

input

```
import numpy as np
```

※ as ○○ で略称を指定する

## Numpyの始め方② 関数

np.○○({処理対象のオブジェクト})

様々な関数名を○○部分に指定して、()内のオブジェクトに処理を適用します。

例えば、以下のように使うことができます。

input

```
array_a = np.array([5, 6, 8])
```



# 第3回 Pythonによる科学計算 (Numpy)

## 始めに強調してお伝えしたいこと



★都度調べればよい

まず, Numpyの機能についてどこまで覚える必要があるのか？ということですが, これに関しては自分から覚えるべきことは少ないと思っています。

特に, 今回の講義だと, 1, 2次元配列の基本的な扱い方を覚えるだけでよく, 他の機能については, **その都度調べればよい**のです。

そうすることで, 自然とよく使うものは覚える一方, 滅多に使わないようなものをたくさん覚える必要もなくなります。実際, プロのエンジニアでも自分が普段扱わない機能については知らないことはたくさんあり, その都度調べています。

# 第3回 Pythonによる科学計算 (Numpy)

## 始めに強調してお伝えしたいこと

### 調べながら学ぶ



関数は暗記するものではなく、調べるものです。

Numpyには(Numpyに限らず)、膨大な数の関数が用意されています。

やってみたいことがあれば、公式ドキュメントなどで検索してみましょう！

The screenshot shows the NumPy documentation website. The header includes the NumPy logo and navigation links: User Guide, API reference, Development, Release notes, and Learn. A search bar is present on the left. The main content area is titled 'How to import NumPy' and explains that to access NumPy and its functions, it should be imported in Python code. A code block shows `import numpy as np`. Below the code block, it states that the imported name is shortened to `np` for better readability and that this is a widely adopted convention. The left sidebar contains a 'GETTING STARTED' section with links to 'What is NumPy?', 'Installation', 'NumPy quickstart', and 'NumPy: the absolute basics for beginners' (highlighted in blue). Below this is a 'FUNDAMENTALS AND USAGE' section.



本講義で扱うこと/扱わないこと

	扱う	扱わない
配列の次元	1 次元・ 2 次元配列	3 次元以上の配列
内容の複雑度	Numpyの基礎的な操作 (Numpy公式ドキュメント参照)	Numpyを用いた複雑な操作 (機械学習モデルの定義など)
演習用notebook	GCIで今後も出てくる 基本的な操作や関数	練習問題, 線形代数に関する詳しい解説

いま説明したのが赤枠部分

(本日の講義でできるようになること)

Numpyと呼ばれるライブラリ (ツールのようなもの) の基本的な使用ができるようになる

### 1. Numpyとは何か、その特徴

#### 2. 1次元配列

2-1. 計算の基本 (ユニバーサル関数・ブロードキャスト・集約関数)

2-2. 中身のデータ参照方法 (インデクシング)

#### 3. 2次元配列

3-1. 計算の基本 (ユニバーサル関数・ブロードキャスト)

3-2. 縦軸・横軸の概念 (axis・集約関数)

3-3. 中身のデータ参照方法 (インデクシング)

# 1次元配列

## 配列の作成と基本操作

【復習】Pythonには、リストというデータ型がある

リストとは、`[]`で囲われ、カンマで区切られた複数の値を持つ形式です。

リストは以下のように作ることができます。

input

```
list_a = [0, 1, 2, 3, 4]  
list_a
```

output

```
[0, 1, 2, 3, 4]
```

## 配列の作成と基本操作

### Numpyの配列「ndarray」の作成

Numpyでは、**numpy.ndarray**というデータ構造で配列を扱います。

ndarrayは **np.array(リスト)** で作ることができます。

input

```
array_a = np.array([0, 1, 2, 3, 4])  
array_a
```

output

```
array([0, 1, 2, 3, 4])
```

ndarrayには、様々なNumpyの機能を適用できます。

## リスト型データとnumpy.ndarrayの比較

	リスト	numpy.ndarray
格納できる要素の型	異なる型の値を格納できる	同じ型の値しか格納できない
四則演算	リスト同士の四則演算は煩雑	ndarray同士の四則演算が容易
多次元配列の扱いやすさ	入れ子構造で擬似的作ることができるが、煩雑	axisという概念で効率的に多次元配列を扱える



# ユニバーサル関数 ufunc

ユニバーサル関数(ufunc): ndarrayを要素ごとに操作する関数

ndarrayは、同じ位置の要素同士で演算を行うことができます。

例) 足し算  $a, b$  は ndarray

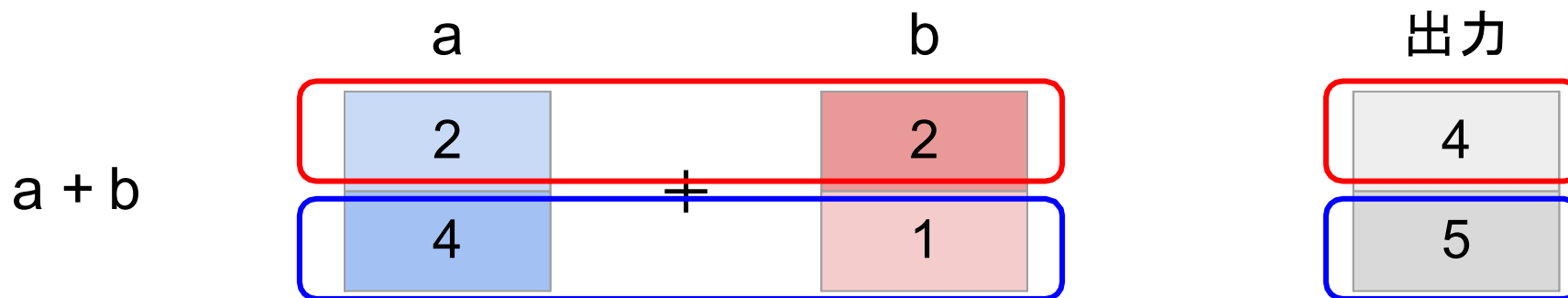
	a		b		出力
a + b	2	+	2		4
	4		1		5

# ユニバーサル関数 ufunc

ユニバーサル関数(ufunc): ndarrayを要素ごとに操作する関数

ndarrayは、同じ位置の要素同士で演算を行うことができます。

例) 足し算  $a, b$  は ndarray



※ 「 $a + b$ 」は、本来は「`np.add(a, b)`」と書きます。

ただし「+」演算子は特別で、使用すると自動的にufuncである `np.add(a, b)` が実行されます。

-: 減法、\*: 乗法、/: 除法、\*\*: 累乗も同様の書き方をすることができます。

## ndarrayの四則演算

### 引き算

`np.subtract(a, b)`

$$a - b$$

a
2
4

−

b
2
1

出力

0
3

### 掛け算

`np.multiply(a, b)`

$$a * b$$

a
2
4

\*

b
2
1

4

4

※内積や行列積ではない

### 割り算

`np.divide(a, b)`

$$a / b$$

a
2
4

/

b
2
1

1

4

## ndarrayの四則演算

実際にコードを書くときは、以下のようにします。

例) 足し算 具体的な実装

input

```
a = np.array([2, 4])  
b = np.array([2, 1])  
a + b
```

output

```
array([4, 5])
```

## 【復習】リストの場合の「+」演算子

※注意※ リストで要素同士の四則演算を行う場合はforループが必要です。

2つのリストを「+」演算子で結合すると、リスト同士が連結されます。

input

`a = [2, 4]``b = [2, 1]``a + b`

output

`[2, 4, 2, 1]`

また、「-、\*、/」などの算術演算子を使用するとエラーが返されます。

## ndarrayの条件演算

等号

`np.equal(a, a)`

`a == a`

a
2
4

`==`

a
2
4

出力

True

True

等号

`np.equal(a, b)`

`a == b`

a
2
4

`==`

b
2
1

True

False

大なり

`np.greater(a, b)`

※小なり `np.less(a, b)`

`a > b`

a
2
4

`>`

b
2
1

False

True

## 【復習】リストの場合の条件演算

※注意※ リストで条件演算を行うと、リスト全体に対する結果を返します。

input

```
a = [2, 4]
```

```
b = [2, 1]
```

```
a == a
```

output

```
True
```

input

```
a == b
```

output

```
False
```

# ユニバーサル関数 ufunc

ユニバーサル関数: ndarrayの要素ごとに演算を行う関数

`a = np.array([1, 2])` とすると、以下のような出力が得られます。

指数関数

input

```
np.exp(a)
```

output

```
array([2.71828183, 7.3890561])
```

対数関数

input

```
np.log(a)
```

output

```
array([0., 0.69314718])
```

aの要素はラジアンとして計算される

三角関数

input

```
np.sin(a)
```

output

```
array([0.84147098, 0.90929743])
```

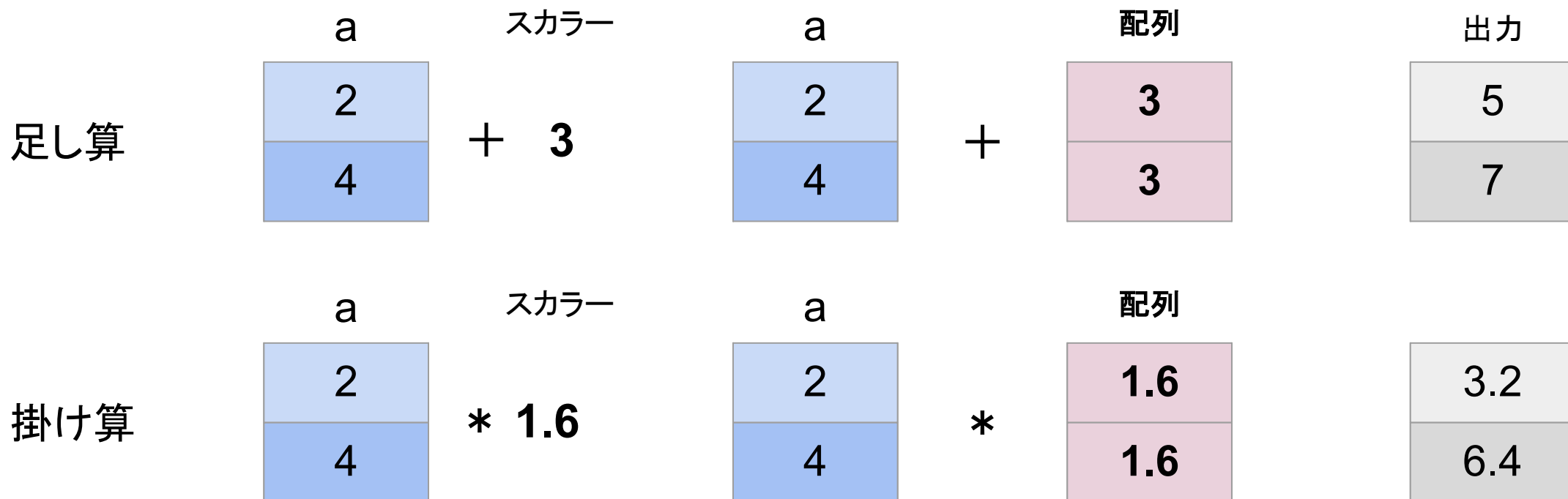
※ cosやtanも同様



# ブロードキャスト

ブロードキャスト: 自動的に形状を合わせて演算を可能にする機能

スカラーが配列の形状に合わせて拡張されて、演算が行われます。



# 集約関数

集約関数: NumPyの配列内の要素を1つの値に集約する関数

`a = np.array([1, 2, 3])` とすると、以下のような出力が得られます。

最大値

input

`np.max(a)`

output

3

合計値

input

`np.sum(a)`

output

6

平均値

input

`np.mean(a)`

output

2.0

標準偏差

input

`np.std(a)`

output

0.816496580927726

いま説明したのが赤枠部分

(本日の講義でできるようになること)

Numpyと呼ばれるライブラリ (ツールのようなもの) の基本的な使用ができるようになる

### 1. Numpyとは何か、その特徴

### 2. 1次元配列

2-1. 計算の基本 (ユニバーサル関数・ブロードキャスト・集約関数)

2-2. 中身のデータ参照方法 (インデクシング)

### 3. 2次元配列

3-1. 計算の基本 (ユニバーサル関数・ブロードキャスト)

3-2. 縦軸・横軸の概念 (axis・集約関数)

3-3. 中身のデータ参照方法 (インデクシング)

# インデクシング(1)



## 冒頭クイズ インデクシング(1)

インデクシングによって、30分ごとの風速データから値を取り出してみましょう。

配列a 30分ごとの風速

12:00 12:30 13:00 13:30 14:00 14:30 15:00 15:30 16:00 16:30 17:00 17:30 18:00 18:30 19:00 19:30 20:00 20:30 21:00

[12, 13, 15, 14, 9, 11, 10, 13, 9, 7, 8, -3, -5, -11, -9, -12, -8, 0, 1]

Q1. 13:00のデータのみを取り出すには？

Q2. 13:00~16:00のデータのみを取り出すには？  
(未満)

Q3. 00分ちょうどのデータのみを取り出すには？



インデクシングによって  
取り出す

## インデクシング(1)

インデクシング: ndarrayにインデックスを指定し、任意の要素を抽出

左から順に0, 1, 2, ...のインデックスを指定して任意の要素を抽出することができます。

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
配列a	0	3	6	9	12	15	18	21	24	27

input

```
a = np.array([0, 3, 6, 9, 12, 15, 18, 21, 24, 27])  
a
```

output

```
array([0, 3, 6, 9, 12, 15, 18, 21, 24, 27])
```

## インデクシング(1)

ndarrayにインデックスを指定し、任意の要素を抽出

左から順に0, 1, 2, ...のインデックスを指定して任意の要素を抽出することができます。

1つの値のみを取り出したいときは、startのみを指定します。

**a[1]**

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	3	6	9	12	15	18	21	24	27

input     **a[1]**

output    **3**

## ndarrayにインデックスを指定し、任意の要素を抽出

一番最後のインデックスの値を取り出したい場合は、-1を指定します。

**a[-1]**

**a[-1]**

27

ndarrayにインデックスを指定し、任意の要素を抽出

複数のインデックスを指定して値を取り出すこともできます。

欲しい要素のインデックスをリストで指定します。

**a[[1, 4, 8]]**      <sup>[0]</sup> <sup>[1]</sup> <sup>[2]</sup> <sup>[3]</sup> <sup>[4]</sup> <sup>[5]</sup> <sup>[6]</sup> <sup>[7]</sup> <sup>[8]</sup> <sup>[9]</sup>  
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]

input      `a[[1, 4, 8]]`

output      `array([3, 12, 24])`

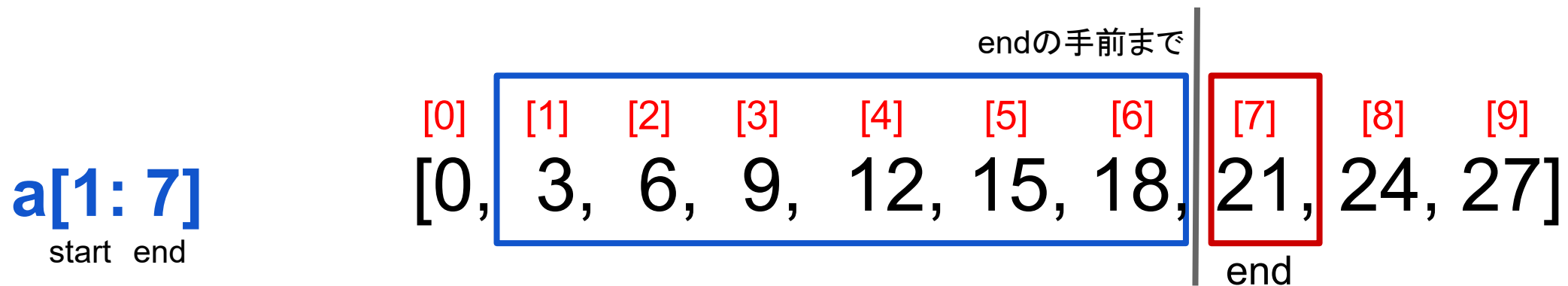


## インデクシング(1)

ndarrayに[start: end: step]を指定し、任意の要素を抽出

連続したインデックスの値を取り出したいときは、startとendを指定します。

startとendの間には、スライス「:」を挟みます。



input

```
a[1: 7]
```

output

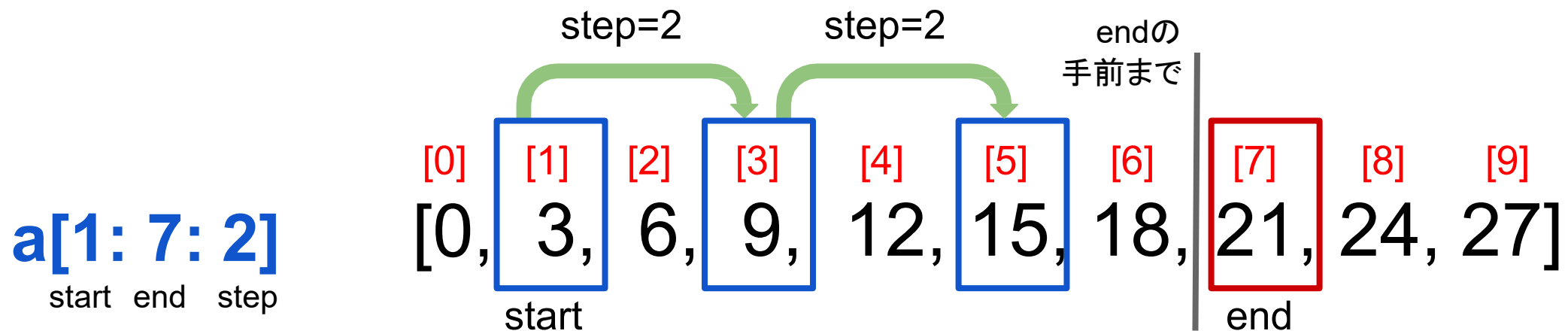
```
array([3, 6, 9, 12, 15, 18])
```

# インデクシング(1)

ndarrayに[start: end: step]を指定し、任意の要素を抽出

一定間隔でインデックスの値を取り出したいときは、start、end、stepを指定します。

startとendとstepの間には、スライス「:」を挟みます。



input

```
a[1: 7: 2]
```

output

```
array([3, 9, 15])
```

## インデクシング(1)

ndarrayに[start: end: step]を指定し、任意の要素を抽出

start以降の全ての値を指定したい場合はスライス「:」の後ろのendを省略できます。

**a[1:]**

start (end)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
[0,	3,	6,	9,	12,	15,	18,	21,	24,	27]

start

input

```
a[1:]
```

output

```
array([3, 6, 9, 12, 15, 18, 21, 24, 27])
```

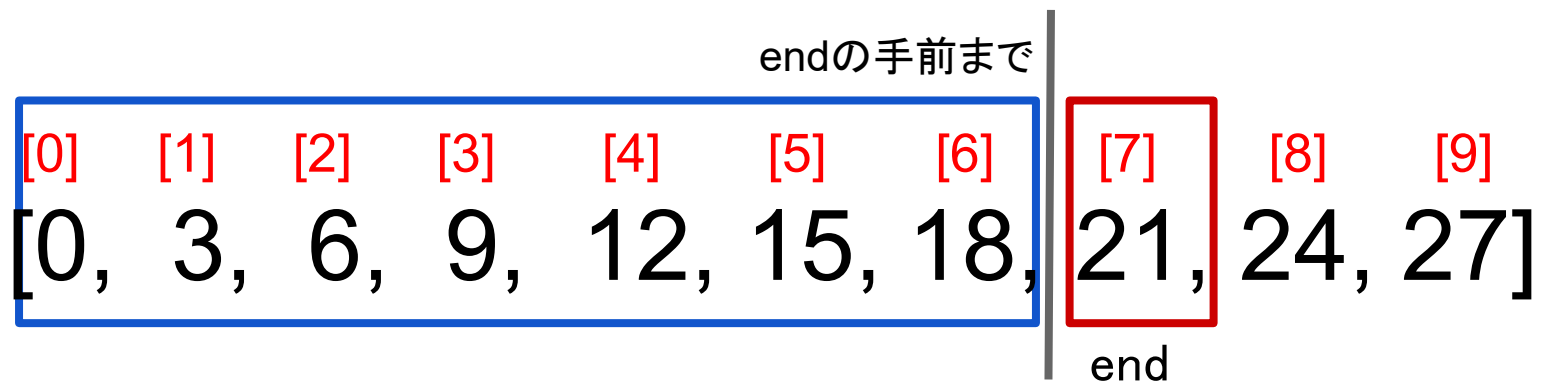
## インデクシング(1)

ndarrayに[start: end: step]を指定し、任意の要素を抽出

end以前の全ての値を指定したい場合はスライス「:」の前のstartを省略できます。

**a[: 7]**

(start) end



input

```
a[: 7]
```

output

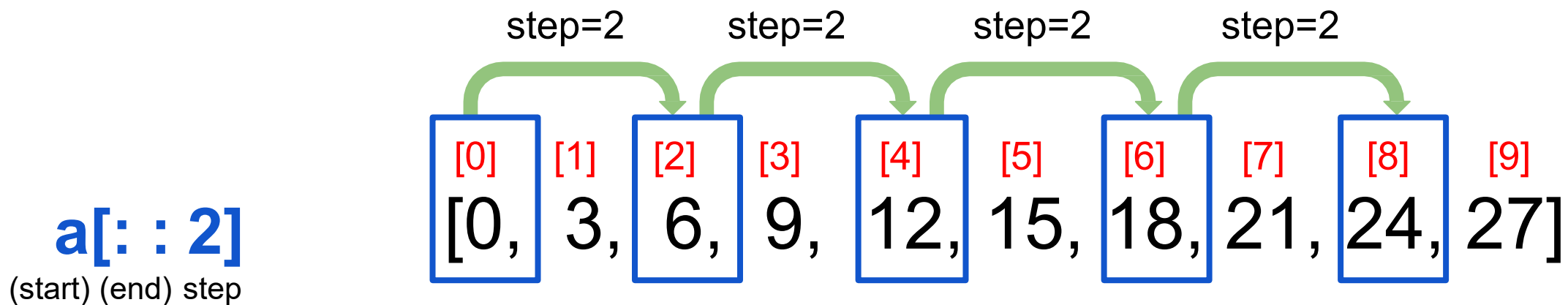
```
array([0, 3, 6, 9, 12, 15, 18])
```

## インデクシング(1)

ndarrayに[start: end: step]を指定し、任意の要素を抽出

配列内全ての一定間隔のインデックスの値を取り出したいときは、stepを指定します。

このとき、スライス「:」前後のstartとendは省略できます。



input    `a[: : 2]`

output    `array([0, 6, 12, 18, 24])`

## インデクシング(1)

冒頭のクイズの答え インデクシング(1)

配列a 30分ごとの風速

12:00	12:30	13:00	13:30	14:00	14:30	15:00	15:30	16:00	16:30	17:00	17:30	18:00	18:30	19:00	19:30	20:00	20:30	21:00
12	13	15	14	9	11	10	13	9	7	8	-3	-5	-11	-9	-12	-8	0	1
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]

Q1. 13:00のデータのみを取り出すには？

# インデクシング(1)

冒頭のクイズの答え インデクシング(1)

配列a 30分ごとの風速

12:00	12:30	13:00	13:30	14:00	14:30	15:00	15:30	16:00	16:30	17:00	17:30	18:00	18:30	19:00	19:30	20:00	20:30	21:00
[12,	13,	15	14,	9,	11,	10,	13,	9,	7,	8,	-3,	-5,	-11,	-9,	-12,	-8,	0,	1]
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]

Q1. 13:00のデータのみを取り出すには？

input

`a[2]`

output

15

## インデクシング(1)

### 冒頭のクイズの答え インデクシング(1)

配列a 30分ごとの風速

12:00	12:30	13:00	13:30	14:00	14:30	15:00	15:30	16:00	16:30	17:00	17:30	18:00	18:30	19:00	19:30	20:00	20:30	21:00
[12,	13,	15,	14,	9,	11,	10,	13,	9,	7,	8,	-3,	-5,	-11,	-9,	-12,	-8,	0,	1]
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]

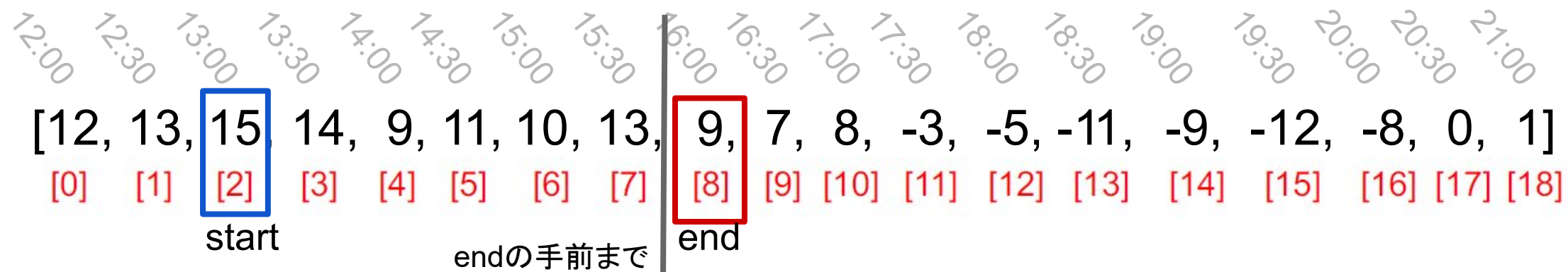
Q2. 13:00~16:00のデータのみを取り出すには？  
(未満)



# インデクシング(1)

冒頭のクイズの答え インデクシング(1)

配列a 30分ごとの風速



Q2. 13:00~16:00のデータのみを取り出すには？  
(未満)

input `a[2: 8]`

output `array([15, 14, 9, 11, 10, 13])`

## インデクシング(1)

冒頭のクイズの答え インデクシング(1)

配列a 30分ごとの風速

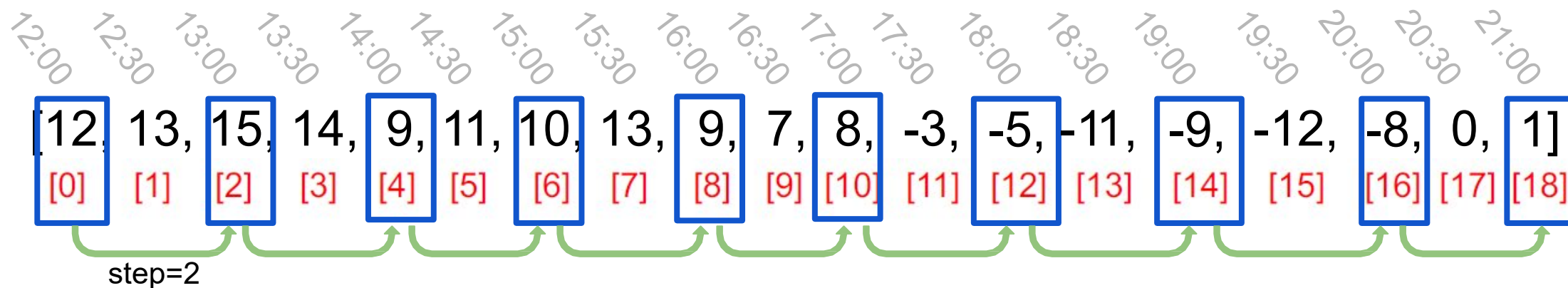
12:00	12:30	13:00	13:30	14:00	14:30	15:00	15:30	16:00	16:30	17:00	17:30	18:00	18:30	19:00	19:30	20:00	20:30	21:00
[12,	13,	15,	14,	9,	11,	10,	13,	9,	7,	8,	-3,	-5,	-11,	-9,	-12,	-8,	0,	1]
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]

Q3. 00分ちょうどのデータのみを取り出すには？

# インデクシング(1)

冒頭のクイズの答え インデクシング(1)

配列a 30分ごとの風速



Q3. 00分ちょうどのデータのみを取り出すには？

input

```
a[:: 2]
```

output

```
array([12, 15, 9, 10, 9, 8, -5, -9, -8, 1])
```

いま説明したのが赤枠部分

(本日の講義でできるようになること)

Numpyと呼ばれるライブラリ (ツールのようなもの) の基本的な使用ができるようになる

### 1. Numpyとは何か、その特徴

### 2. 1次元配列

2-1. 計算の基本 (ユニバーサル関数・ブロードキャスト・集約関数)

2-2. 中身のデータ参照方法 (インデクシング)

### 3. 2次元配列

3-1. 計算の基本 (ユニバーサル関数・ブロードキャスト)

3-2. 縦軸・横軸の概念 (axis・集約関数)

3-3. 中身のデータ参照方法 (インデクシング)

# 2次元配列

# N次元配列

numpy.ndarrayで、N次元配列データを表す

2次元以上(N次元)でもndarrayを作ることができます。

1次元の場合と同様に、基本的には以下のように作成します。

input

```
a = np.array([[5, 6, 8], [9, 2, 1]])
```

ネスト構造のリスト

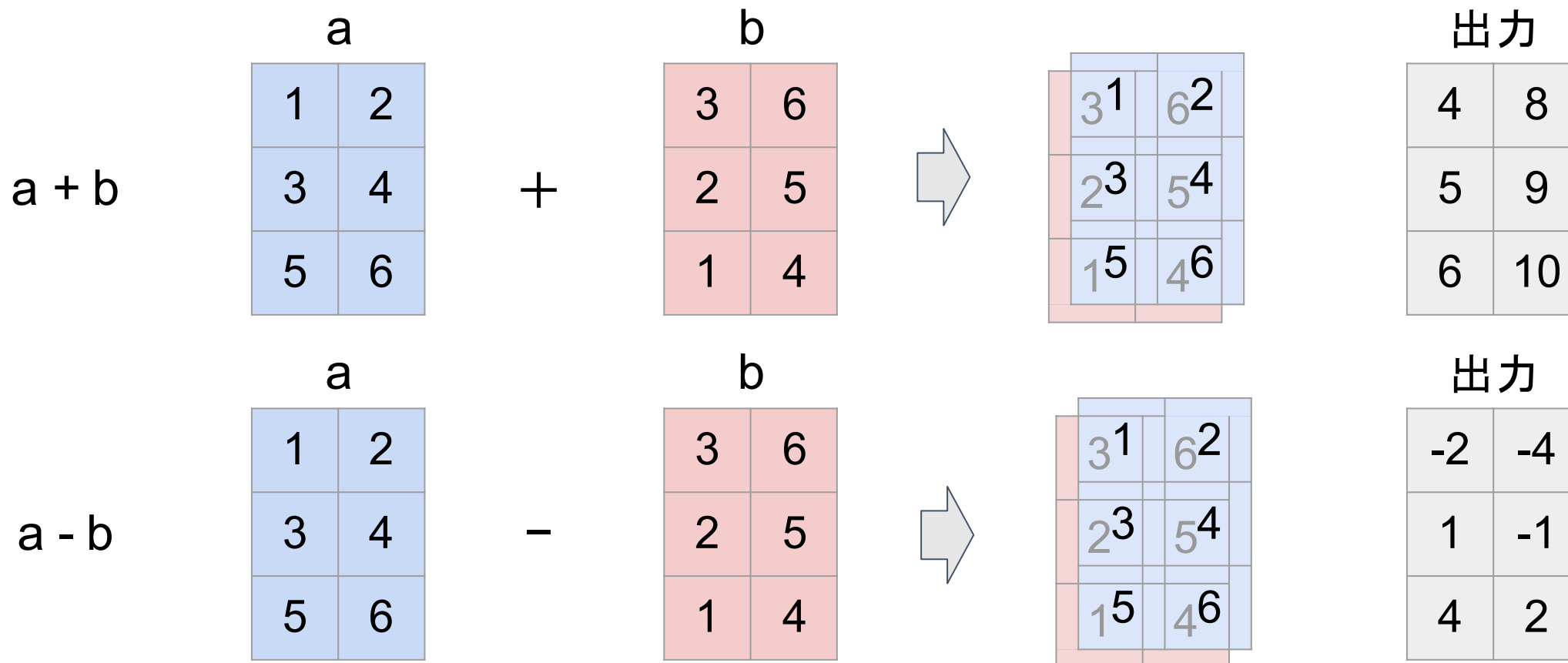
```
a = np.array([[5, 6, 8],  
               [9, 2, 1]])
```

という書き方でもOK  
(改行は実行に影響ありません)

ネスト構造: リストの中にリストが入っている構造

ユニバーサル関数(ufunc): ndarrayを要素ごとに操作する関数

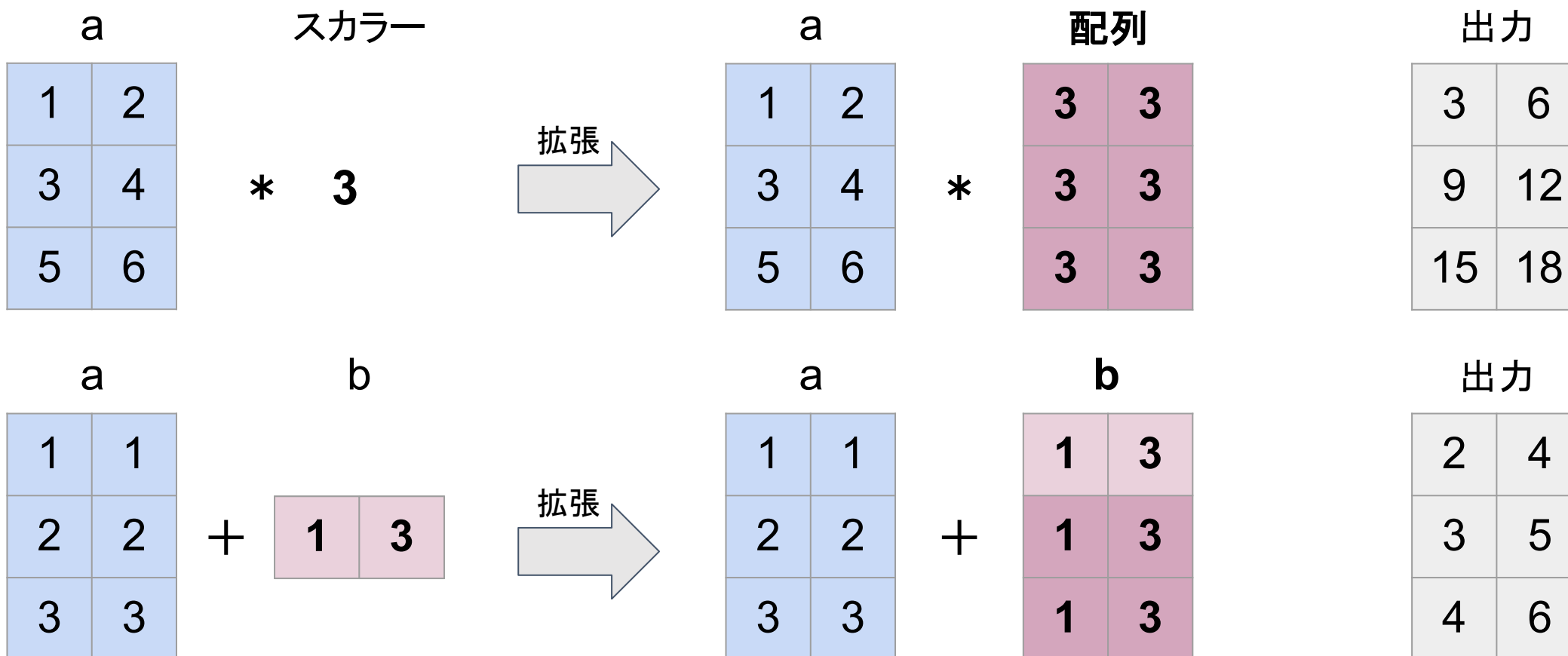
2次元配列でも、要素ごとの演算の四則演算を行うことができます。



# N次元配列 ブロードキャスト

ブロードキャスト: 自動的に形状を合わせて演算を可能にする機能

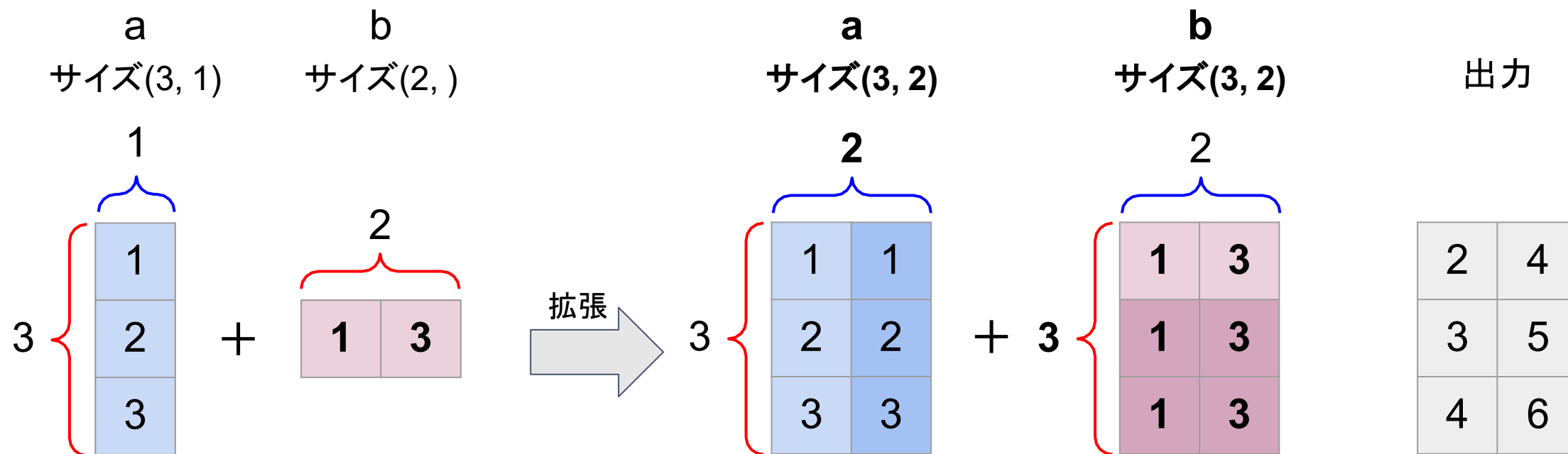
サイズが1の次元が、相手の配列の同じ位置の次元のサイズに拡張されます。





ブロードキャスト: 自動的に形状を合わせて演算を可能にする機能

サイズが1の次元が、相手の配列の同じ位置の次元のサイズに拡張されます。



※ 1次元配列には内部で次元が追加され、 $(2, ) \rightarrow (1, 2)$ の2次元配列に変換されてから拡張される

いま説明したのが赤枠部分

(本日の講義でできるようになること)

Numpyと呼ばれるライブラリ (ツールのようなもの) の基本的な使用ができるようになる

### 1. Numpyとは何か、その特徴

### 2. 1次元配列

2-1. 計算の基本 (ユニバーサル関数・ブロードキャスト・集約関数)

2-2. 中身のデータ参照方法 (インデクシング)

### 3. 2次元配列

3-1. 計算の基本 (ユニバーサル関数・ブロードキャスト)

3-2. 縦軸・横軸の概念 (axis・集約関数)

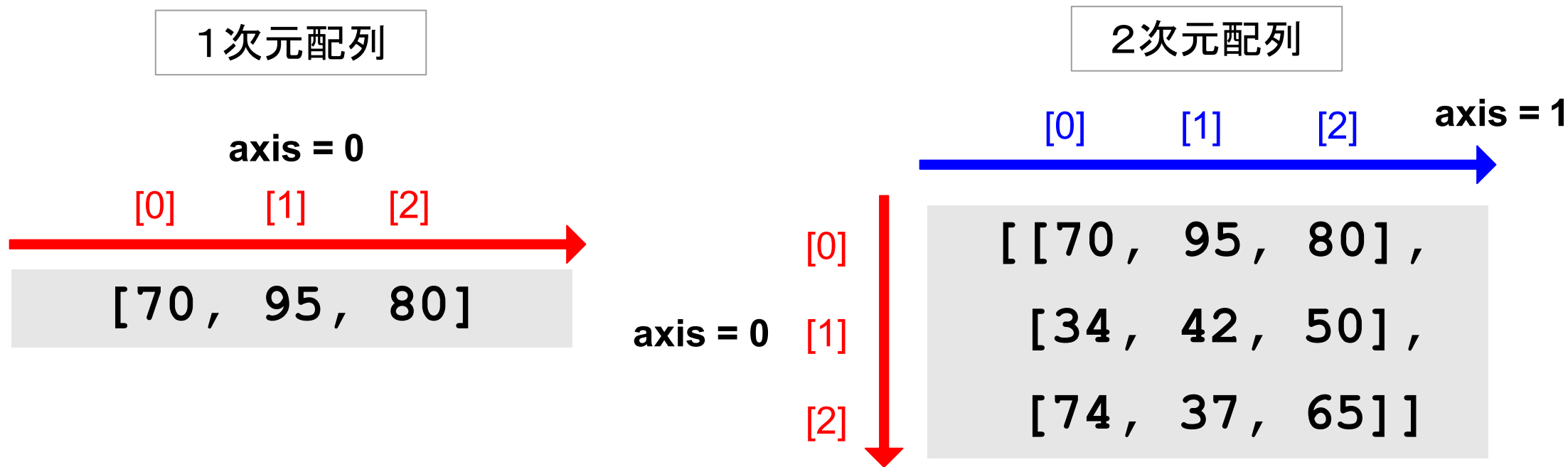
3-3. 中身のデータ参照方法 (インデクシング)

# N次元配列の軸「axis」

axis: N次元配列に対して簡単に特定の方向にアクセスするための概念

ndarrayでは、axisという概念で軸方向が区別されます。

2次元配列の場合、axis=0は行、axis=1は列を表します。

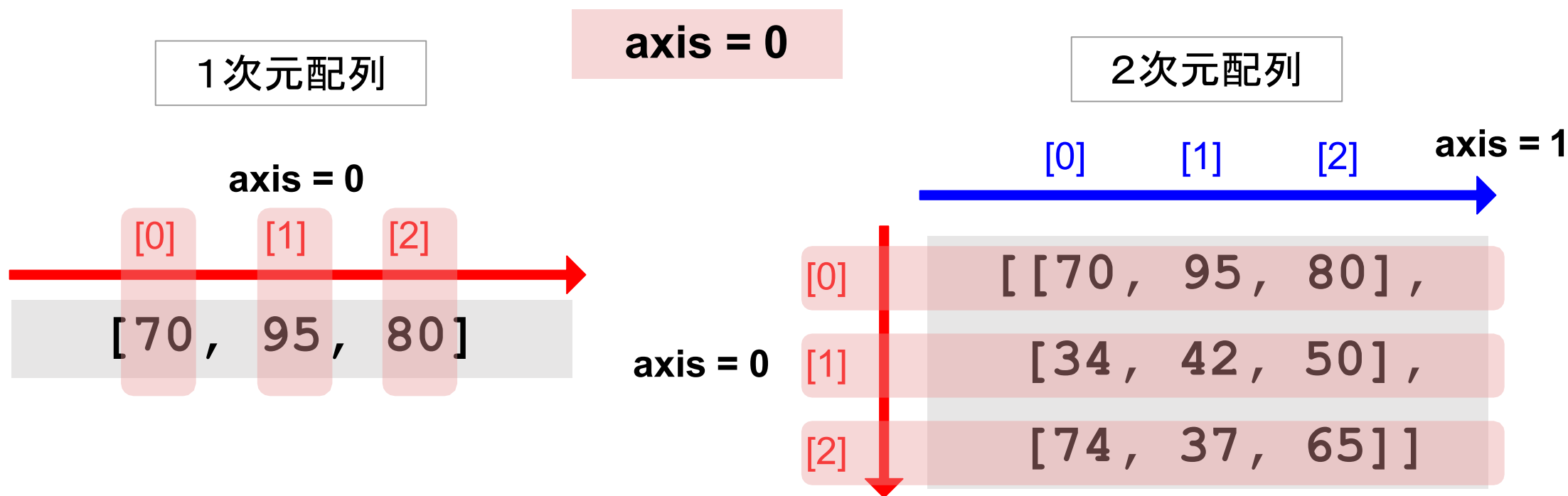


# N次元配列の軸「axis」

axis: N次元配列に対して簡単に特定の方向にアクセスするための概念

ndarrayでは、axisという概念で軸方向が区別されます。

2次元配列の場合、axis=0は行、axis=1は列を表します。

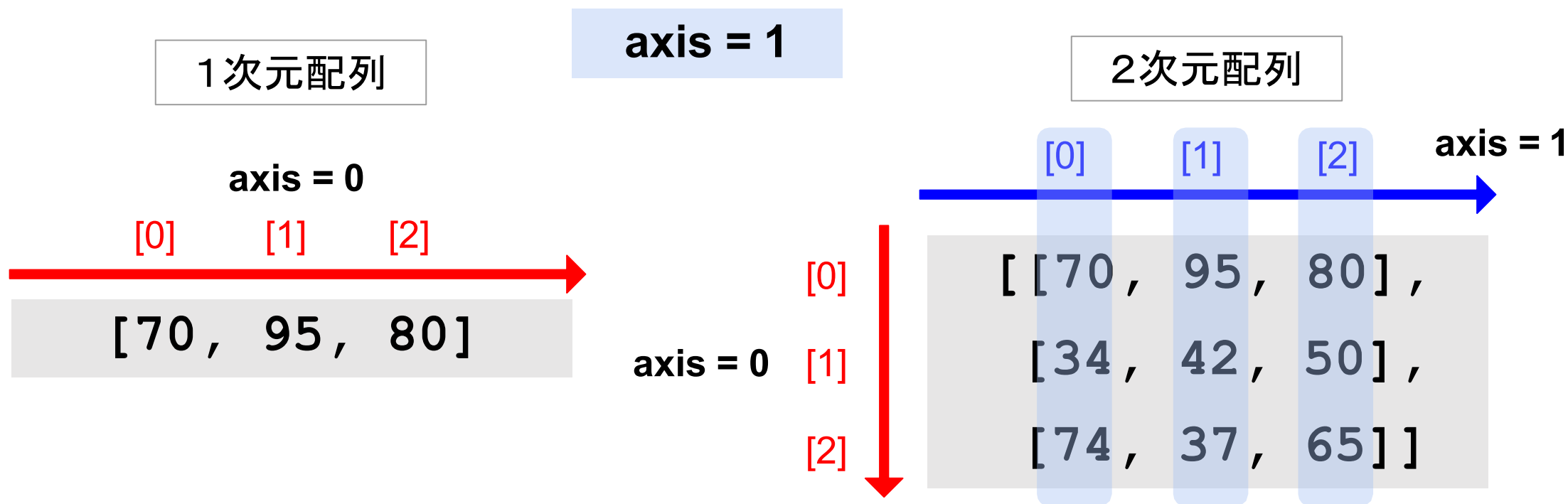


# N次元配列の軸「axis」

axis: N次元配列に対して簡単に特定の方向にアクセスするための概念

ndarrayでは、axisという概念で軸方向が区別されます。

2次元配列の場合、axis=0は行、axis=1は列を表します。



# 集約関数

集約関数: NumPyの配列内の要素を1つの値に集約する関数

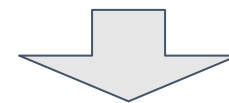
集約関数を用いて、各次元ごとの最大値を出してみましょう。

2次元配列 a

	Aさん [0]	Bさん [1]	Cさん [2]	Dさん [3]
国語 [0]	75	60	82	80
数学 [1]	90	73	75	85
英語 [2]	82	92	80	63

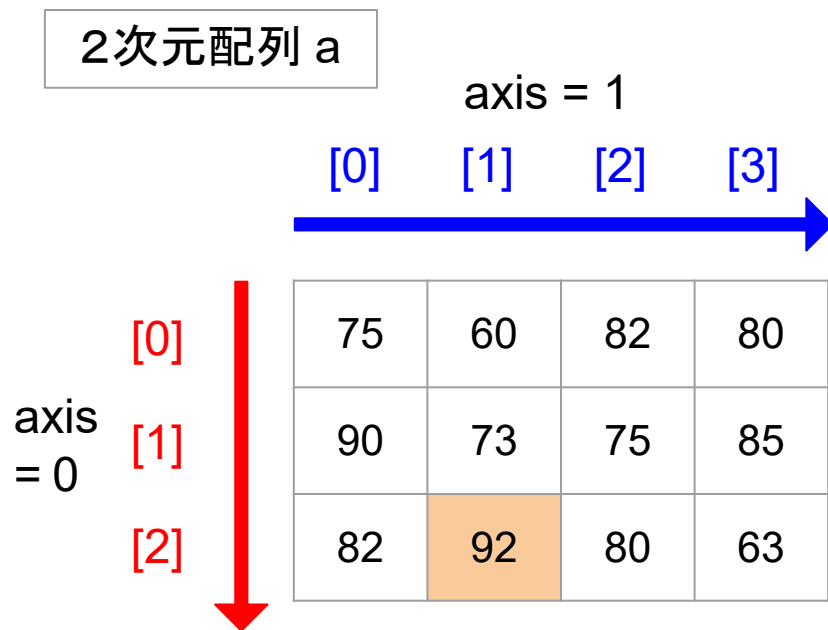
Q1. ADさんのそれぞれの  
最高得点は？

Q2. 各教科の最高得点は？



集約関数で調べる

集約関数: NumPyの配列内の要素を1つの値に集約する関数



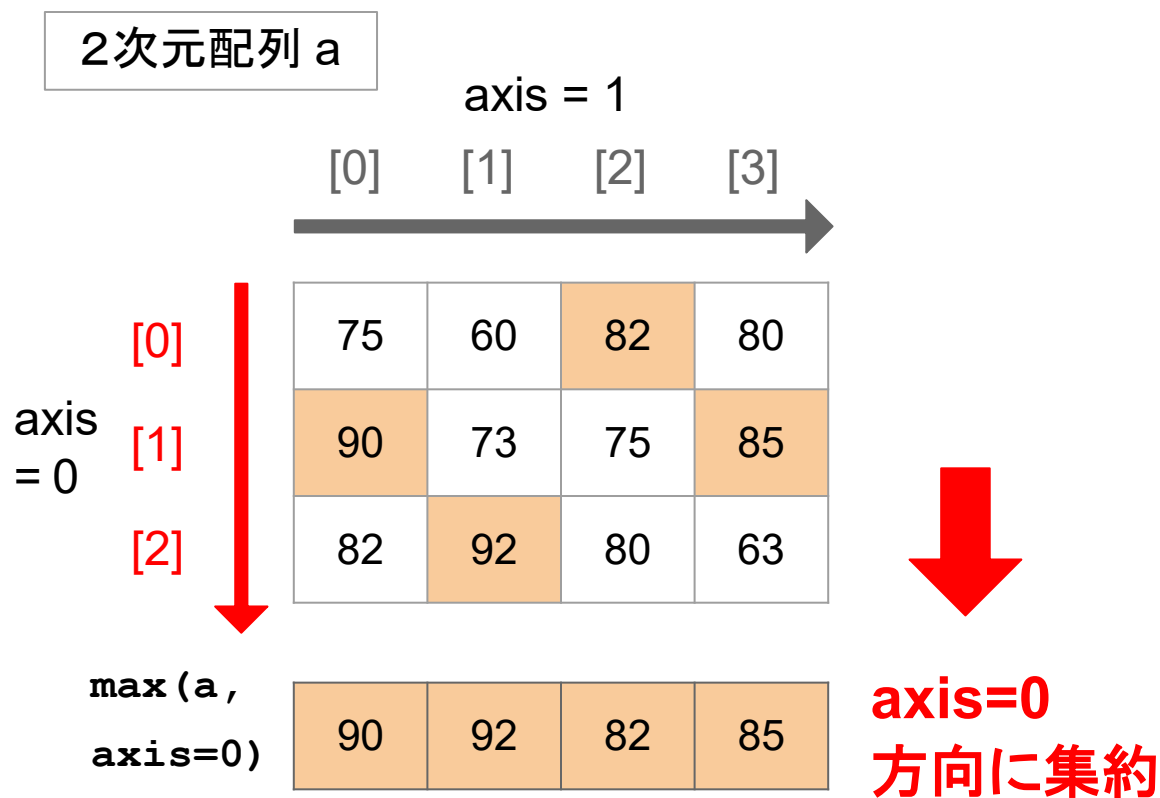
input

`np.max(a)`

output

92

`np.max(ndarray, axis=0)`のように軸指定して、`axis=0`方向の最大値を得る



input `np.max(a)`

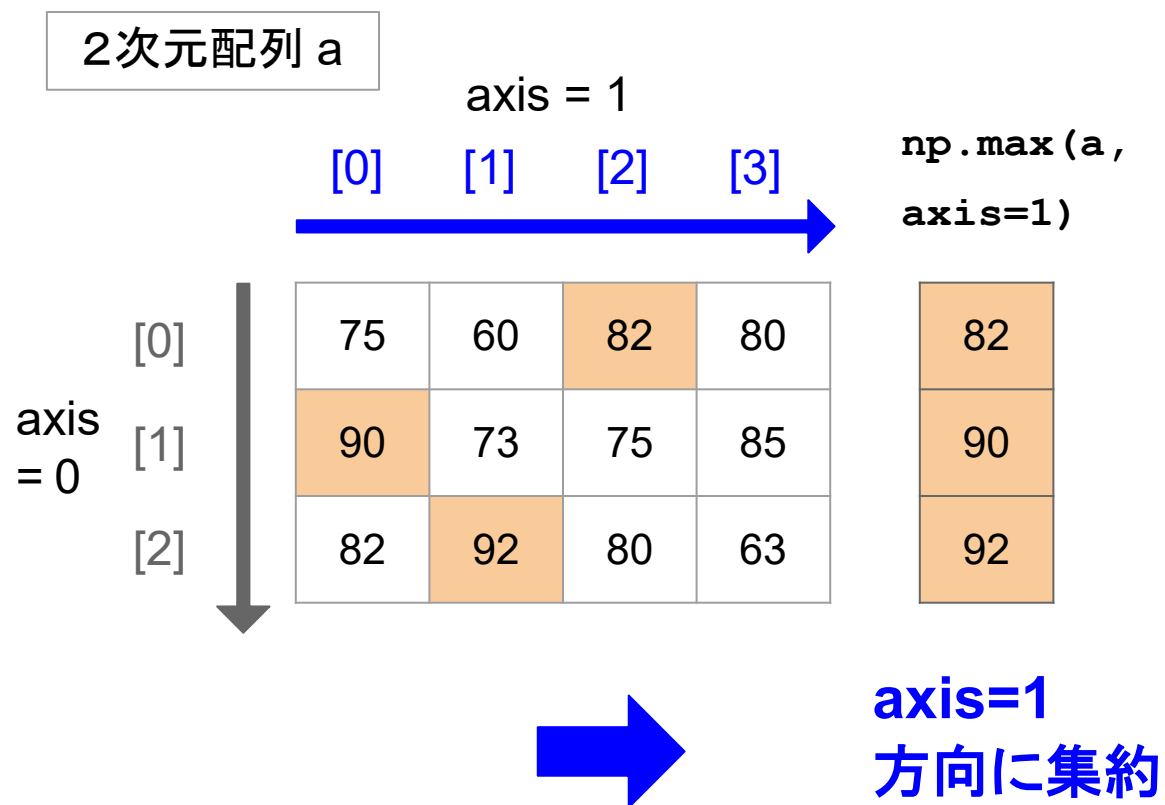
output 92

input `np.max(a, axis=0)`

output `array([90, 92, 82, 85])`



`np.max(ndarray, axis=1)`のように軸指定して、`axis=1`方向の最大値を得る



input `np.max(a)`

output 92

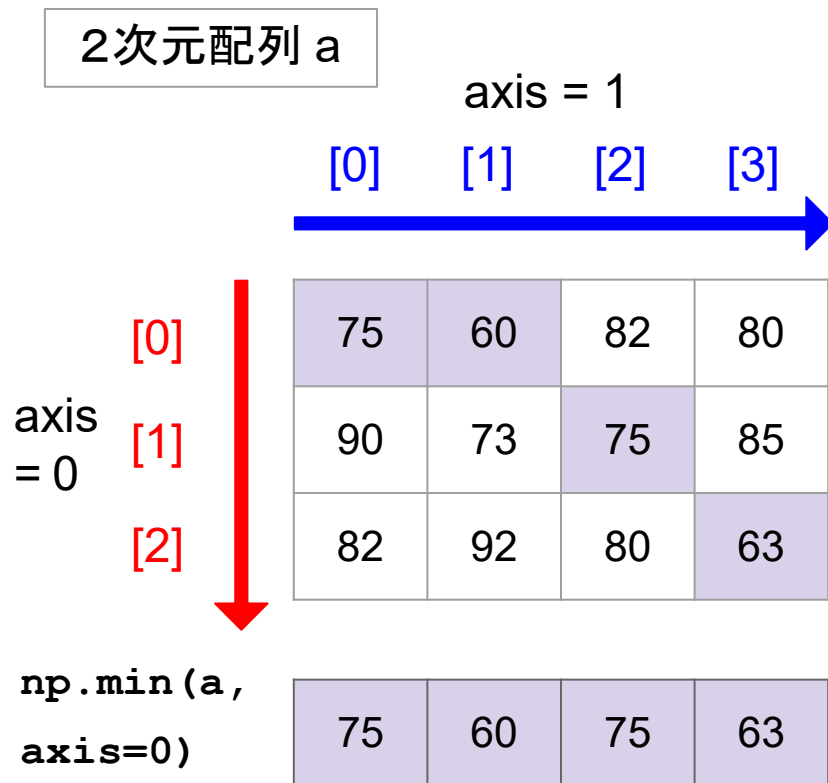
input `np.max(a, axis=0)`

output `array([90, 92, 82, 85])`

input `np.max(a, axis=1)`

output `array([82, 90, 92])`

np.max()以外の集約関数 np.min()、np.sum()、np.mean()、np.std()



axis=0方向に集約

最小値

input

`np.min(a, axis=0)`

output

`array([75, 60, 75, 63])`

合計値

input

`np.sum(a, axis=0)`

平均値

input

`np.mean(a, axis=0)`

標準偏差

input

`np.std(a, axis=0)`

いま説明したのが赤枠部分

(本日の講義でできるようになること)

Numpyと呼ばれるライブラリ (ツールのようなもの) の基本的な使用ができるようになる

### 1. Numpyとは何か、その特徴

### 2. 1次元配列

2-1. 計算の基本 (ユニバーサル関数・ブロードキャスト・集約関数)

2-2. 中身のデータ参照方法 (インデクシング)

### 3. 2次元配列

3-1. 計算の基本 (ユニバーサル関数・ブロードキャスト)

3-2. 縦軸・横軸の概念 (axis・集約関数)

3-3. 中身のデータ参照方法 (インデクシング)

## インデクシング(2)

## 冒頭クイズ インデクシング(2)

様々なインデックス指定方法で、2次元配列 `a` から任意の要素を抽出してみましょう。

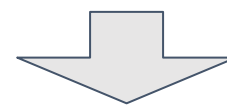
2次元配列 `a`

	Aさん [0]	Bさん [1]	Cさん [2]	Dさん [3]
国語 [0]	75	60	82	80
数学 [1]	90	73	75	85
英語 [2]	82	92	80	63

axis = 0 (vertical arrow)  
axis = 1 (horizontal arrow)

Q1. Dさんの数学の成績は？

Q2. A, B, Cさんの国語と英語の成績は？



インデクシングによって  
取り出す

# インデクシング(2) N次元配列の基本的なインデックス指定

2次元配列から、axis=0方向の要素を取り出す

Q1

2次元配列a

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

欲しい出力

[0]	[1]	[2]	[3]
1	2	3	4

input

`a[0]`

output

`array([1, 2, 3, 4])`

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

# インデクシング(2) N次元配列の基本的なインデックス指定

低い次元からインデックスを指定して要素を取り出す(① axis=0 ② axis=1)

Q2

2次元配列a

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

欲しい出力

[0]
2

①axis=0 ②axis=1

input

`a[0, 1]`

output

2

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

## インデクシング(2) N次元配列の基本的なインデックス指定

連続している要素はスライスで指定する

Q3

2次元配列a

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

欲しい出力

[0]	[1]	[2]
2	6	10

↓ インデックスが連続しているので、スライスで指定できる

input

`a[:, 1]`

output

`array([2, 6, 10])`

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

## インデクシング(2) N次元配列の基本的なインデックス指定

連続している要素はスライスで指定する

Q4

2次元配列a

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

欲しい出力

		[0]	[1]	[2]
axis=0	[0]	2	3	4
	[1]	6	7	8

↓                      ↓インデックスが連続しているので、スライスで指定できる

input            `a[: 2, 1:]`

output           `array([[2, 3, 4], [6, 7, 8]])`

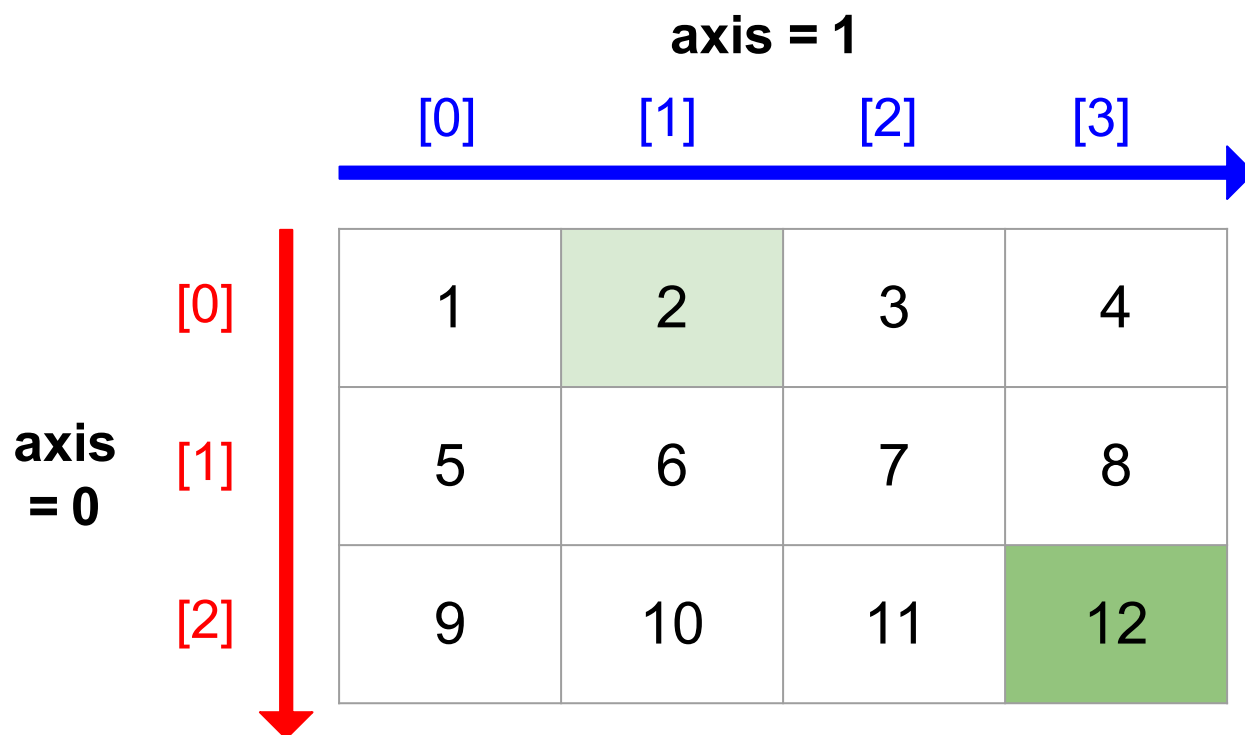
		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12



# インデクシング(2) N次元配列の高度なインデックス指定

2次元配列から、離れた位置にある任意の要素を抽出する

離れた位置にある要素を任意の並び方で取り出すにはどうすれば良いのでしょうか？



欲しい出力

①

2	12
---	----

②

2
12

### 高度なインデックス指定のルール

離れている要素を任意の並び方で取得する方法には、以下のルールがあります。

(1) `a[[axis=0のインデックス], [axis=1のインデックス]]` のようにaxis毎に指定

(2) 出力されるndarrayの「形状」や「要素の位置」はインデックス指定に従う

(3) ブロードキャスト機能を利用して、記述を省略できる

## インデクシング(2) N次元配列の高度なインデックス指定

(1)  $a[[\text{axis}=0\text{のインデックス}], [\text{axis}=1\text{のインデックス}]$  のようにaxis毎に指定

Q5

2次元配列a

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

欲しい出力

[0]	[1]
2	12

axis=0のindex      axis=1のindex

$a[ \quad \boxed{0 \quad 2} \quad , \quad \boxed{1 \quad 3} \quad ]$

input

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

output

`array([2, 12])`

## インデクシング(2) N次元配列の高度なインデックス指定

(2) 出力されるndarrayの「形状」や「要素の位置」はインデックス指定に従う

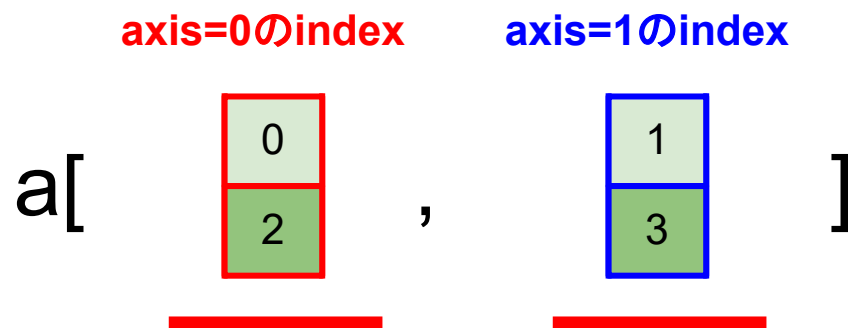
Q6

2次元配列a

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

欲しい出力

	[0]
[0]	2
[1]	12



input

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

output

array([[2], [12]])

※ axis=1のサイズが1の2次元配列

## インデクシング(2) N次元配列の高度なインデックス指定

(2) 出力されるndarrayの「形状」や「要素の位置」はインデックス指定に従う

Q6

2次元配列a

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

欲しい出力

	[0]
[0]	2
[1]	12

axis=0のindex      axis=1のindex

a[ 

0
2

 , 

1
3

 ]

同じ形状

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

array([[2], [12]])

※ axis=1のサイズが1の2次元配列

## (3) ブロードキャスト機能を利用して、記述を省略できる①

Q7

2次元配列a

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

欲しい出力

	[0]	[1]
	2	4

ブロードキャスト: 自動的に形状を合わせて演算を可能にする機能

$a[$ 
0 0
 $,$ 
1 3
 $]$ 
→
省略
 $a[$ 
0
 $,$ 
1 3
 $]$

input `a[[0, 0], [1, 3]]`

output `array([2, 4])`

input `a[0, [1, 3]]`

output `array([2, 4])`

## (3) ブロードキャスト機能を利用して、記述を省略できる②

Q8

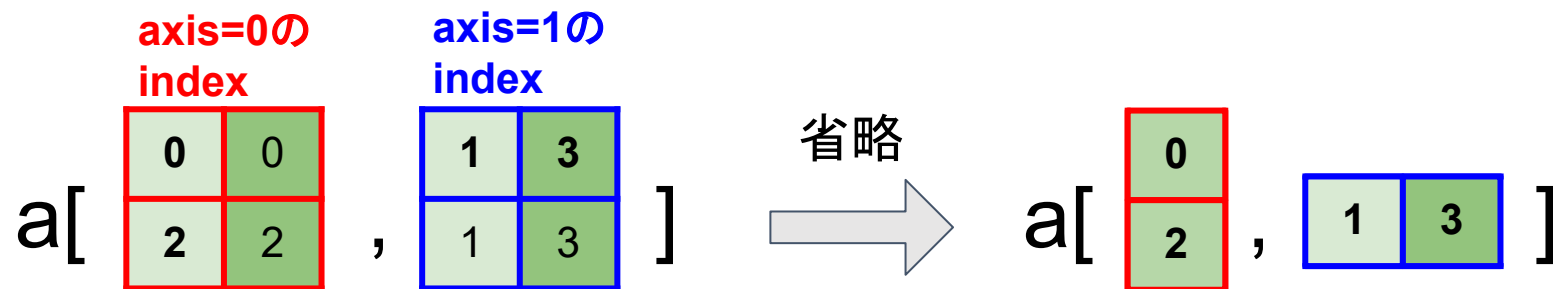
2次元配列a

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

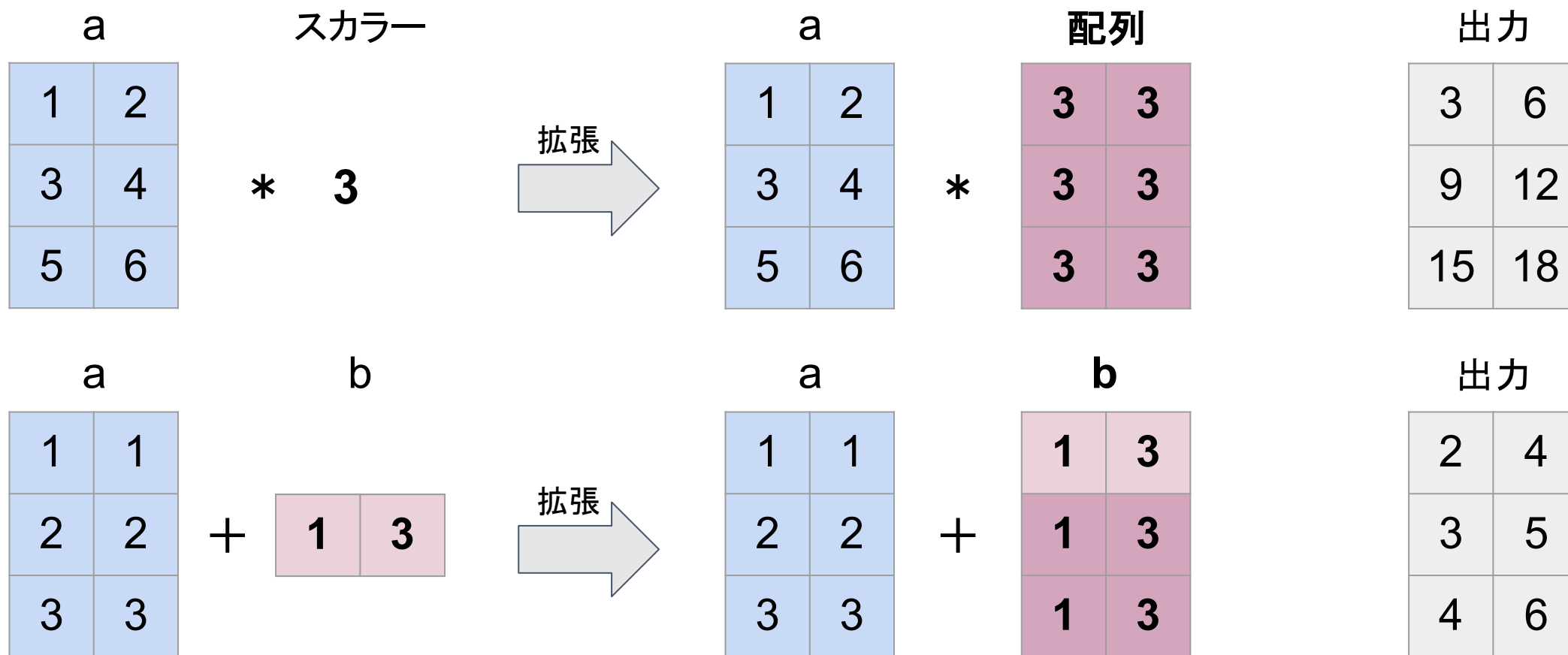
欲しい出力

	[0]	[1]
[0]	2	4
[1]	10	12

ブロードキャスト: 自動的に形状を合わせて演算を可能にする機能

input `a[[[0, 0], [2, 2]], [[1, 3], [1, 3]]]`output `array([[2, 4], [10, 12]])`input `a[[[0], [2]], [1, 3]]`output `array([[2, 4], [10, 12]])`

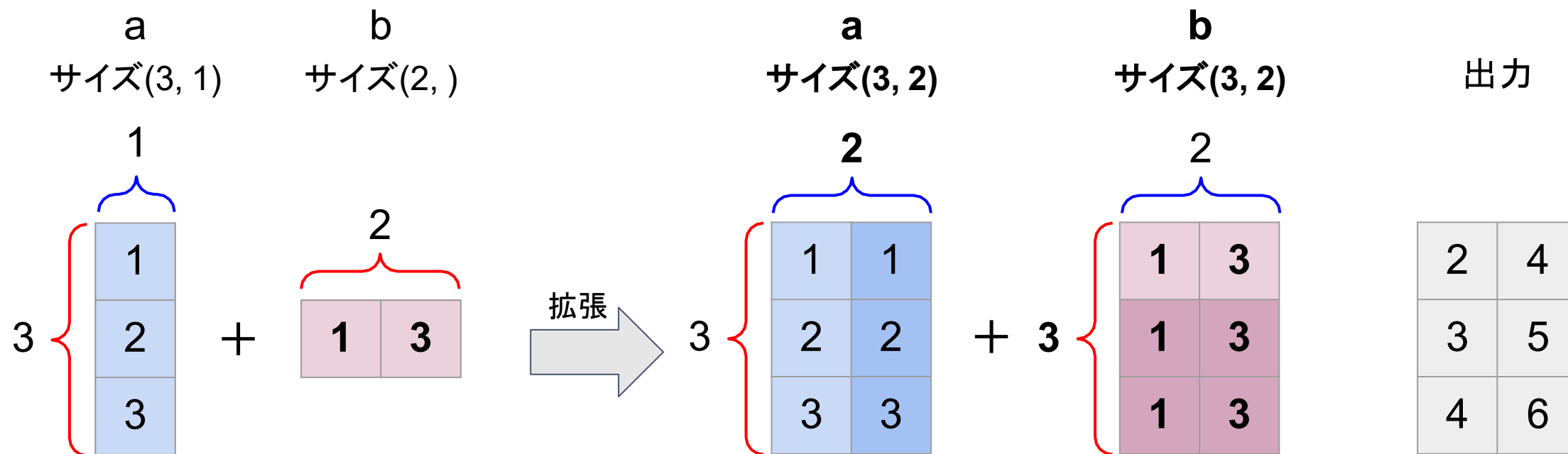
サイズが1の次元が、相手の配列の同じ位置の次元のサイズに拡張されます。





(再掲)ブロードキャスト:自動的に形状を合わせて演算を可能にする機能

サイズが1の次元が、相手の配列の同じ位置の次元のサイズに拡張されます。



## インデクシング(2) N次元配列の高度なインデックス指定

## (3) ブロードキャスト機能を利用して、記述を省略できる②

Q8

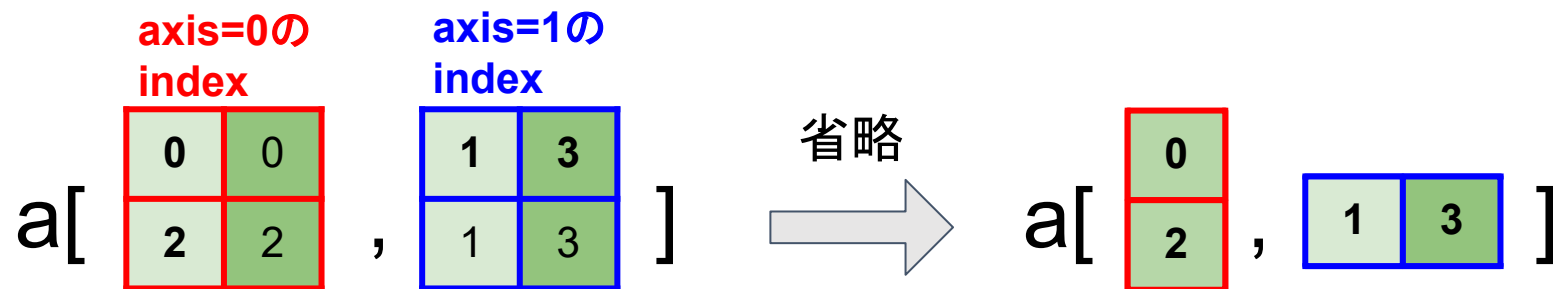
2次元配列a

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

欲しい出力

	[0]	[1]
[0]	2	4
[1]	10	12

ブロードキャスト: 自動的に形状を合わせて演算を可能にする機能

input `a[[[0, 0], [2, 2]], [[1, 3], [1, 3]]]`output `array([[2, 4], [10, 12]])`input `a[[[0], [2]], [1, 3]]`output `array([[2, 4], [10, 12]])`

# インデクシング(2) 基本的なインデックス指定と高度なインデックス指定の組み合わせ

基本的なインデックス指定と高度なインデックス指定を組み合わせることも可能

Q9

2次元配列a

		axis=1			
		[0]	[1]	[2]	[3]
axis=0	[0]	1	2	3	4
	[1]	5	6	7	8
	[2]	9	10	11	12

欲しい出力

		[0]	[1]	[2]
[0]		2	3	4
		10	11	12

↓ 高度な指定   ↓ 基本的な指定

input

`a[[0, 2], 1:]`

output

`array([[2, 3, 4], [10, 11, 12]])`

## インデクシング(2)

## 冒頭のクイズの答え インデクシング(2)

2次元配列 a

	Aさん	Bさん	Cさん	Dさん	
	[0]	[1]	[2]	[3]	axis = 1
国語 [0]	75	60	82	80	
数学 [1]	90	73	75	85	
英語 [2]	82	92	80	63	
axis = 0					

Q1. Dさんの数学の成績は？

# インデクシング(2)

## 冒頭のクイズの答え インデクシング(2)

2次元配列 a

	Aさん	Bさん	Cさん	Dさん
	[0]	[1]	[2]	[3]
国語 [0]	75	60	82	80
数学 [1]	90	73	75	85
英語 [2]	82	92	80	63

axis = 0 (vertical arrow)

axis = 1 (horizontal arrow)

Q1. Dさんの数学の成績は？

input a[1, 3]

output 85

# インデクシング(2)

## 冒頭のクイズの答え インデクシング(2)

2次元配列 a

	Aさん	Bさん	Cさん	Dさん
	[0]	[1]	[2]	[3]
国語 [0]	75	60	82	80
数学 [1]	90	73	75	85
英語 [2]	82	92	80	63

axis = 0 (vertical arrow)

axis = 1 (horizontal arrow)

Q2. A, B, Cさんの国語と英語の成績は？

## インデクシング(2)

## 冒頭のクイズの答え インデクシング(2)

2次元配列 a

	Aさん	Bさん	Cさん	Dさん
	[0]	[1]	[2]	[3]
国語 [0]	75	60	82	80
数学 [1]	90	73	75	85
英語 [2]	82	92	80	63

axis = 0 (vertical arrow)  
axis = 1 (horizontal arrow)

Q2. A, B, Cさんの国語と英語の成績は？

input

`a[[0, 2], : 3]`

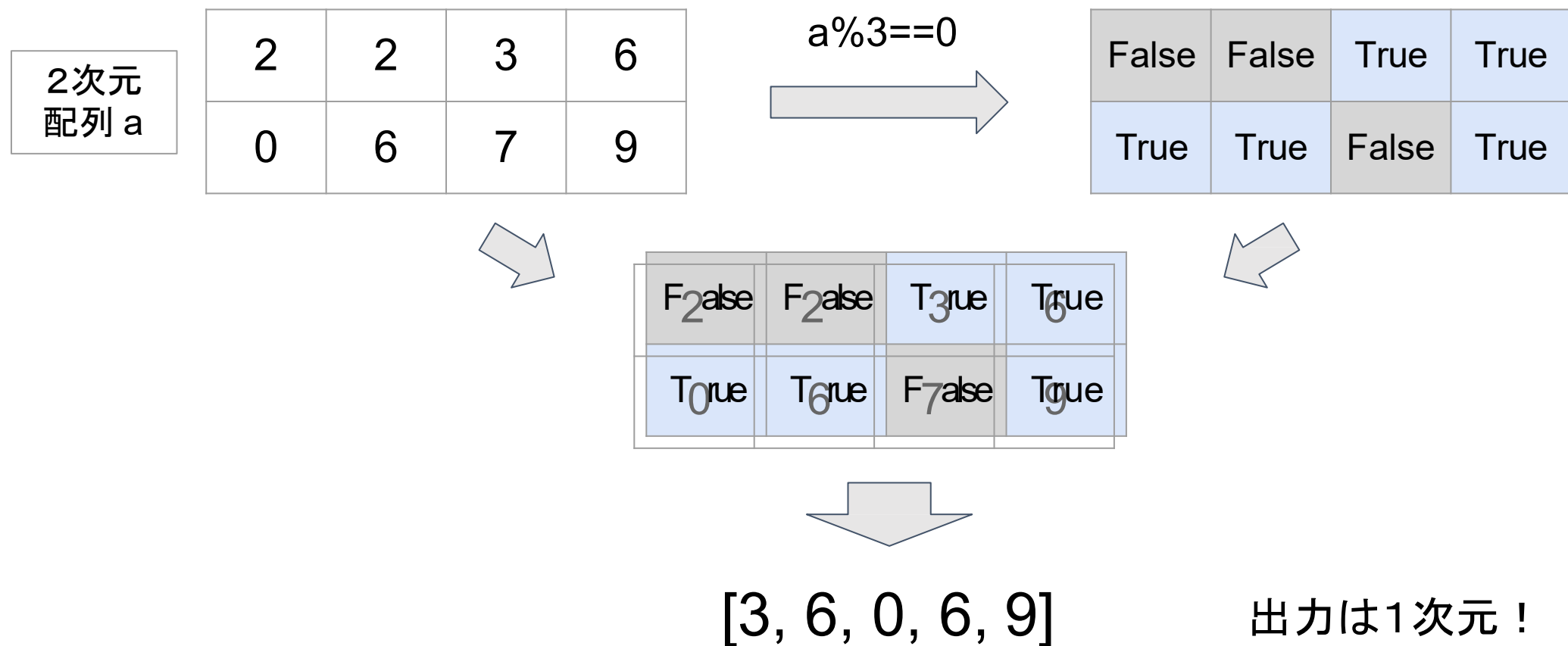
output

`array([[75, 60, 82],  
 [82, 92, 80]])`

## インデクシング(2)

### ブール値によるインデックス指定

ブール値によって取得したい値を指定することもできます。





いま説明したのが赤枠部分

(本日の講義でできるようになること)

Numpyと呼ばれるライブラリ (ツールのようなもの) の基本的な使用ができるようになる

### 1. Numpyとは何か、その特徴

### 2. 1次元配列

2-1. 計算の基本 (ユニバーサル関数・ブロードキャスト・集約関数)

2-2. 中身のデータ参照方法 (インデクシング)

### 3. 2次元配列

3-1. 計算の基本 (ユニバーサル関数・ブロードキャスト)

3-2. 縦軸・横軸の概念 (axis・集約関数)

3-3. 中身のデータ参照方法 (インデクシング)

## 第3回 Pythonによる科学計算 (Numpy)

# まとめ・本日でできるようになったこと



Numpy: Pythonで複雑な科学計算を行うためのライブラリ

1. Numpyの特徴を理解できる
2. ユニバーサル関数によって、for文なしで要素ごとの計算ができる
3. ブロードキャストによって、サイズの違う配列同士で演算できる
4. ndarrayのインデックスやaxis(軸)の概念が分かる
5. インデクシングによって、配列から任意の要素や配列を取り出せる
6. 集約関数によって、axisごとの統計量を算出できる

## 第3回 Pythonによる科学計算 (Numpy)

# 最後にも強調してお伝えしたいこと



★都度調べればよい

まず, Numpyの機能についてどこまで覚える必要があるのか？ということですが, これに関しては自分から覚えるべきことは少ないと思っています。

特に, 今回の講義だと, 1, 2次元配列の基本的な扱い方を覚えるだけでよく, 他の機能については, **その都度調べればよい**のです。

そうすることで, 自然とよく使うものは覚える一方, 滅多に使わないようなものをたくさん覚える必要もなくなります。実際, プロのエンジニアでも自分が普段扱わない機能については知らないことはたくさんあり, その都度調べています。

講座受講の目的を考えて、適切な生成AIの利用を！

## 受講の手引き > はじめに > 生成AI(ChatGPT等)の使用について

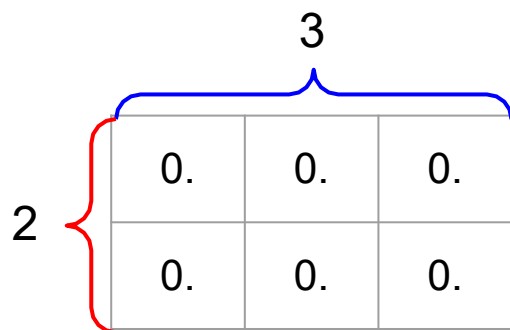
### ▼ 生成AI（ChatGPT等）の使用について

- 本講義での基本スタンスとして、ChatGPT含む生成AIを用いて作成したコードをそのまま利用することはお控えください。生成AIの回答がすべて正しいとは限らないため利用者自身で正しいかどうか見極めることも必要です。正しいかどうか判断できるような知識をつけるためにも本講義では一度ご自身で演習などを参考に実装し、理解を深めることを推奨いたします。
- GCI内におけるChatGPTを始めとしたLLMサービス（以下、「サービス」と記載）の利用に関するポリシー
  1. 「サービス」を利用した際の出力の責任は、使用者に帰属する。
    - a. 「サービス」による出力が正しいとは限らないため、利用者自身で正しいかどうか見極める必要がある。そのため、「サービス」を使用する際の入力と出力は使用者自身で責任をもって管理/使用/評価すること。
  2. 講座受講の目的は、あくまでも学習である。この目的から外れる使い方は避ける。
    - a. 「サービス」の出力をそのまま宿題の回答とする行為などは、学びの目的から大きく逸脱する。学びにつながらない使用は避けること。



# 休憩

【コラム】ndarrayを作る方法色々 0のみ、1のみの配列を作ることができる



A 2x3 grid of cells, each containing the value 0. A red curly brace on the left indicates the row dimension (2), and a blue curly brace on top indicates the column dimension (3).

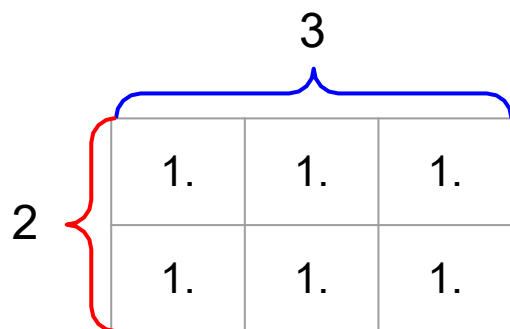
0.	0.	0.
0.	0.	0.

input

```
np.zeros((2, 3))
```

output

```
array([[0., 0., 0.],  
       [0., 0., 0.]])
```



A 2x3 grid of cells, each containing the value 1. A red curly brace on the left indicates the row dimension (2), and a blue curly brace on top indicates the column dimension (3).

1.	1.	1.
1.	1.	1.

input

```
np.ones((2, 3))
```

output

```
array([[1., 1., 1.],  
       [1., 1., 1.]])
```

※ 0. や 1. のように「.」が付くのは、float(浮動小数点数)型であることを表しているためです

## 補助資料 `np.arange()`

`np.arange(start, stop, step)`: start から stop までのndarrayをstepの間隔で生成

stopのみを指定して、startとstepは省略することができます。

input `np.arange(8)`

output `array([0, 1, 2, 3, 4, 5, 6, 7])`

input `np.arange(2, 8)`

output `array([2, 3, 4, 5, 6, 7])`

input `np.arange(2, 8, 2)`

output `array([2, 4, 6])`

`np.reshape(array, new_shape)`: ndarrayの形状を変更するための関数

2次元配列a

4

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

3

input

`a.shape`

output

`(3, 4)`

2次元配列b

`a.reshape(2, 6)`

6

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

2

`ndarray.reshape( axis=0の  
サイズ , axis=1の  
サイズ )`

のように指定する

input

`b = a.reshape(2, 6)``b.shape`

output

`(2, 6)`



補助資料 `np.reshape()`

要素数と他の次元のサイズが既知の場合、-1を指定すると自動でサイズ調整される

2次元配列a

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

input

`a.shape`

output

`(3, 4)`

2次元配列b

`a.reshape(2, -1)`

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

(2, -1)を指定すると(2, 6)に自動変換される

input

`b = a.reshape(2, -1)``b.shape`

output

`(2, 6)`

## 補助資料 属性

属性: ndarrayに関連付けられた情報

2次元配列a

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

ndarray.shape: 配列の形状を示す

input

**a.shape**

output

**(3, 4)**

ndarray.ndim: 配列の次元数を示す

input

**a.ndim**

output

**2**

ndarray.dtype: 配列のデータタイプを示す

input

**a.dtype**

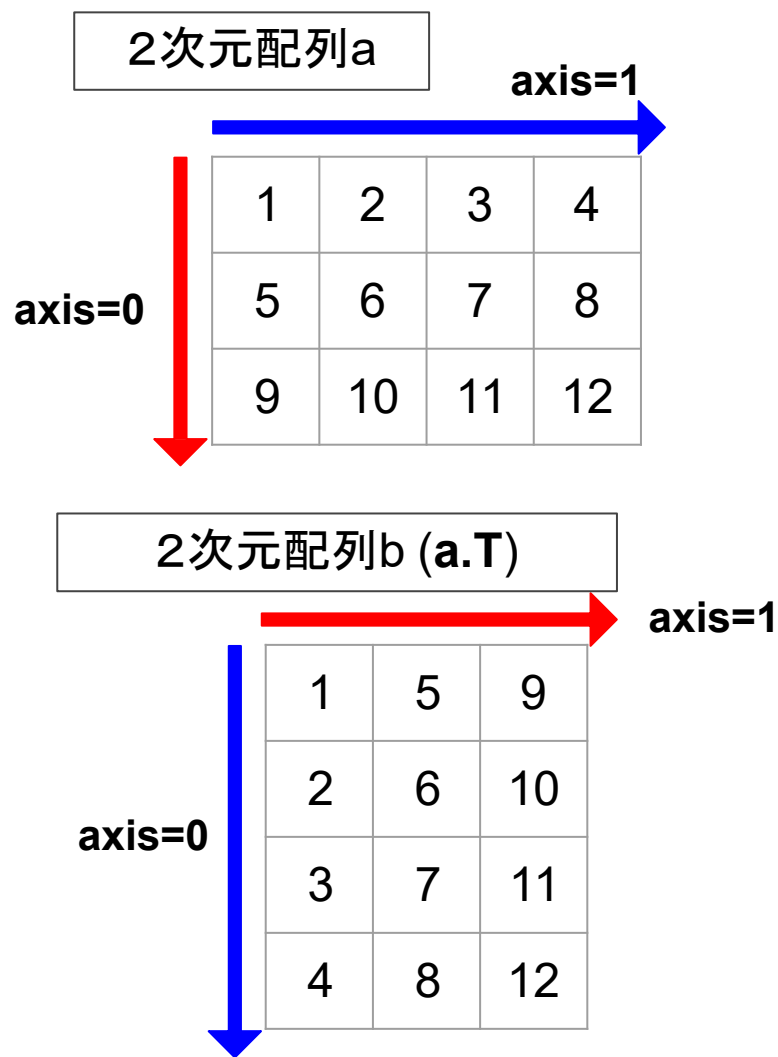
output

**dtype('int64')**

※ int64: 64bit整数型

## 補助資料 ndarray.T 転置属性

ndarray.T: 転置。配列の行と列を入れ替える操作



ndarray.T: 転置。配列の行と列を入れ替える操作

input

```
b = a.T
```

```
b
```

output

```
array([[ 1,  5,  9],  
       [ 2,  6, 10],  
       [ 3,  7, 11],  
       [ 4,  8, 12]])
```

input

```
b.shape
```

output

```
(4, 3)
```