

機械学習のための PythonマスターコースDay2



<https://to-kei.net>

全人類がわかる統計学 Presents



目次

1. Numpy紹介
2. 実行前必要なもの
3. 配列の作成
4. 配列の要素指定
5. 配列の計算
6. 他によく使われる関数

1. Numpy紹介

1.1 数学計算用のモジュール

1.2 他によく使われるpythonのライブラリー



Pythonにおいて数値計算を効率的に行うための拡張モジュール

- 多次元配列を扱える
- 高速な数値と行列計算ができる
- 大量のデータ処理にかかる時間を短縮できる



他によく使われるpythonのライブラリー（orモジュール）

- Pandas：データ処理
- BeautifulSoup4：ウェブスクレイピング
- Scipy：数値計算
- Matplotlib：可視化
- Seaborn：可視化
- Scikit-learn：機械学習
- TensorFlow：深層学習
- ...
- ...

2. 実行前必要なもの

2.1 Anacondaのインストール

2.2 Anacondaにプリーインストールされているライブラリー



Anacondaのインストール

➤ダウンロードサイト：<https://www.anaconda.com/download/>

The screenshot shows the Anaconda website's download page. At the top, there's a navigation bar with links for Documentation, Blog, Contact, and a search icon. Below this, the Anaconda logo is on the left, and a menu with links for 'What is Anaconda?', 'Products', 'Support', 'Resources', 'About', and a highlighted 'Downloads' button is on the right. The main section has a green background with the text 'Download Anaconda Distribution' and 'Version 5.3 | Release Date: September 28, 2018'. Below this, it says 'Download For:' followed by icons for Windows, Apple, and Linux. At the bottom, there are three columns of text describing the distribution's features: 'High-Performance Distribution' (Easily install 1,400+ [data science packages](#)), 'Package Management' (Manage packages, dependencies and environments with [conda](#)), and 'Portal to Data Science' (Uncover insights in your data and create interactive visualizations). A footer bar at the bottom says 'Download for Your Preferred Platform'.




ANACONDA.

Documentation Blog Contact

What is Anaconda? Products Support Resources About Downloads

Download Anaconda Distribution

Version 5.3 | Release Date: September 28, 2018

Download For:   

High-Performance Distribution

Easily install 1,400+ [data science packages](#)

Package Management

Manage packages, dependencies and environments with [conda](#)

Portal to Data Science

Uncover insights in your data and create interactive visualizations

Download for Your Preferred Platform



Anacondaにプリインストールされているライブラリー

- Numpy
- Pandas
- Seaborn
- Matplotlib
- Scikit-Learn
- beautifulsoup4
- TensorFlow
- ...
- (Anacondaにある全てモジュール：<https://docs.anaconda.com/anaconda/packages/old-pkg-lists/4.3.1/py35/>)

3. 配列の作成

3.1 ライブラリーのインポート

3.2 現在Numpyのバージョンを確認

3.3 1次元の配列を作成

3.4 データの次元数と形状確認

3.5 2次元の配列を作成

3.6 3次元の配列を作成

3.7 arange関数

3.8 練習



Numpyの読み込み

1. ライブラリーのインポート

import numpy as np

(「as np」は省略のため)

2. 現在Numpyのバージョンを確認

print(np.__version__)

アップデート方法

>>> **conda update numpy**

In [1]:

```
import numpy as np
```

In [2]:

```
print(np.__version__)
```

1.14.0



配列の作成

3. 1次元の配列を作成

`np.array([1,2,3])`

In [3]:

```
np.array([1,2,3])
```

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

4. データの次元数と形状確認

`np.array([1,2,3]).shape`

In [6]:

```
np.array([1,2,3]).shape
```

```
(3,)
```



配列の作成

5. 2次元の配列を作成

`np.array([[1, 2, 3], [4, 5, 6]])`

Shape (2,3)

3列

2行

1	2	3
4	5	6

In [5]:

```
np.array([[1, 2, 3], [4, 5, 6]])
```

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

In [7]:

```
np.array([[1, 2, 3], [4, 5, 6]]).shape
```

```
(2, 3)
```



配列の作成

6. 3次元の配列を作成

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

In [9]:

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

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

```
[[ 7, 8, 9],  
 [10, 11, 12]])
```

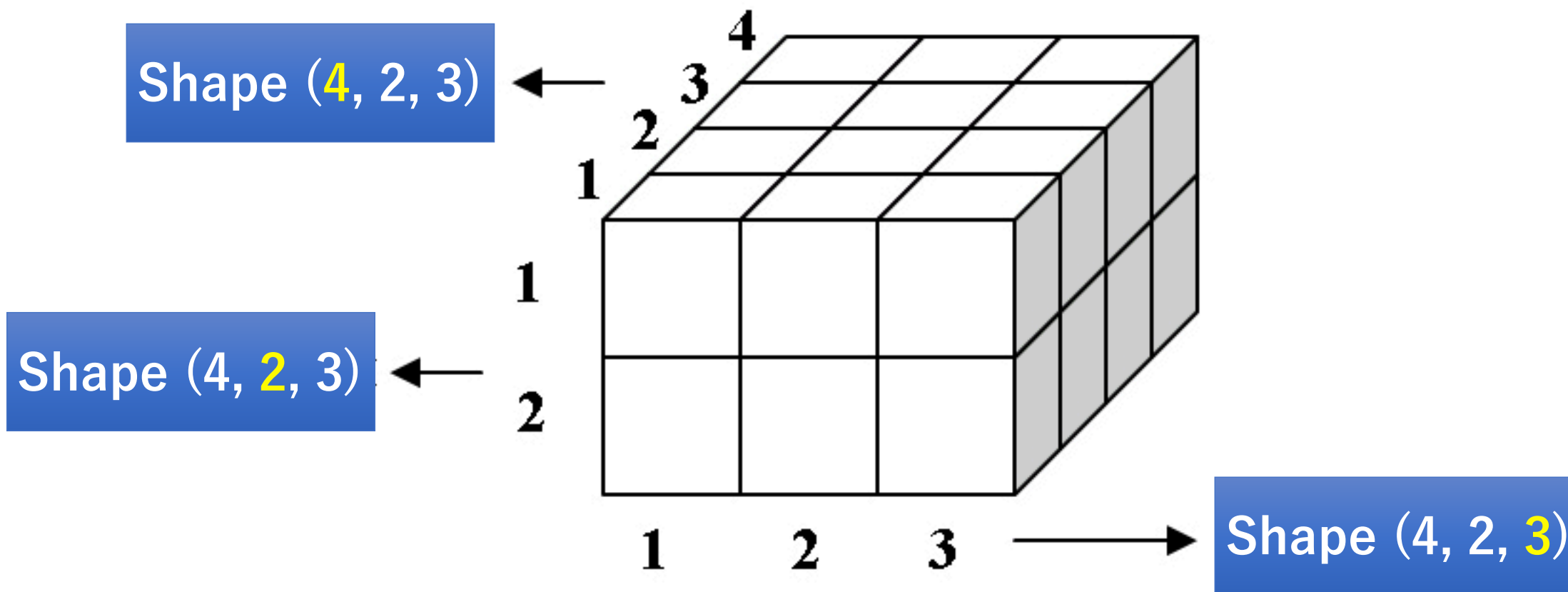
In [10]:

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

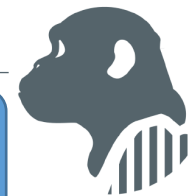
```
(2, 2, 3)
```



3次元配列の説明



3次元配列の表示順番は 「層」 ⇒ 「行」 ⇒ 「列」



arange関数 1

7. arange

np.arange(5)

```
In [40]:
```

```
np.arange(5)
```

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

np.arange(5.0)

```
In [41]:
```

```
np.arange(5.0)
```

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



arange関数 2

np.arange(5, 50)

5 から49まで、1次元
の整数昇順列を作成

In [42]:

```
np.arange(5, 50)
```

```
array([ 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])
```

np.arange(5, 50, 5)

5 から49まで5単位間
隔で、1次元の整数昇順
列を作成

In [43]:

```
np.arange(5, 50, 5)
```

```
array([ 5, 10, 15, 20, 25, 30, 35, 40, 45])
```




arange関数

- まとめ -

引数が 1つの場合：`np.arange(#始点=0、終点、#間隔=1)`

引数が 2つの場合：`np.arange(始点、終点、#間隔=1)`

引数が 3つの場合：`np.arange(始点、終点、間隔)`

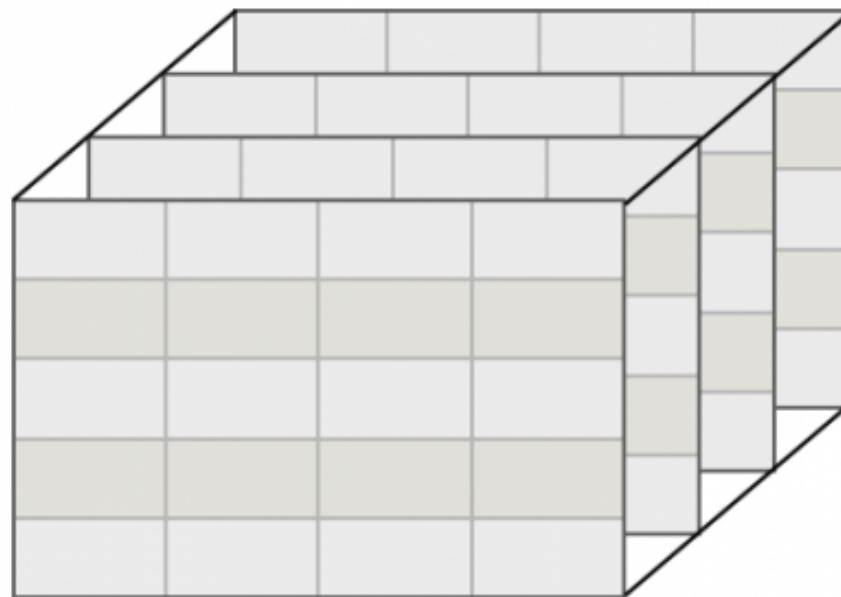


練習

1. numpyで下記の等差数列を作成してください
[2, 5, 8, 11, 14, 17, 20]

2. 上記の等差数列は他の作成方法がありますか？

3. 右のような3次元配列を作成してください
(shapeが同じであれば)





練習解答

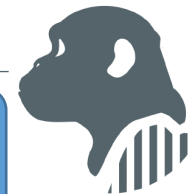
1. numpyでこのような配列を作成してください。

[2, 5, 8, 11, 14, 17, 20]

```
In [51]:
```

```
np.array([ 2, 5, 8, 11, 14, 17, 20])
```

```
array([ 2, 5, 8, 11, 14, 17, 20])
```



練習解答

[2, 5, 8, 11, 14, 17, 20]

2. 上記の等差数列は他の作成方法がありますか？

右の3種類

In [52]:

```
np.arange(2,21,3)
```

```
array([ 2,  5,  8, 11, 14, 17, 20])
```

In [53]:

```
np.arange(2,22,3)
```

```
array([ 2,  5,  8, 11, 14, 17, 20])
```

In [54]:

```
np.arange(2,23,3)
```

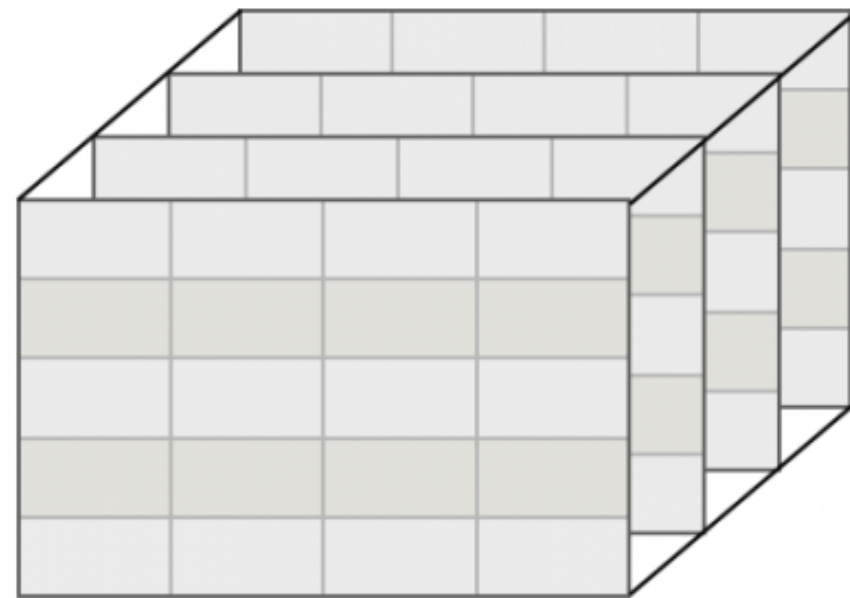
```
array([ 2,  5,  8, 11, 14, 17, 20])
```



練習解答

3. 右のような3次元配列を作成してください。

```
b = np.array([  
    [[1,1,1,1],[1,1,1,1],[1,1,1,1],[1,1,1,1],[1,1,1,1]],  
    [[1,1,1,1],[1,1,1,1],[1,1,1,1],[1,1,1,1],[1,1,1,1]],  
    [[1,1,1,1],[1,1,1,1],[1,1,1,1],[1,1,1,1],[1,1,1,1]]  
])  
b.shape
```



4. 配列の要素指定

4.1 1次元配列の要素指定

4.2 1次元配列の要素数え方

4.3 要素の変換

4.4 2次元配列の要素指定

4.5 3次元配列の要素指定

4.6 練習



1次元配列の要素指定

まず、0~5まで、1単位を感覚で整数昇順1次元配列を作成

c = np.arange(6)

In [63]:

```
c = np.arange(6)
```

```
c
```

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

配列Cの第1位の要素を抽出する

c[0]

In [64]:

```
c[0]
```

```
0
```

配列Cの第4位の要素を抽出する

c[3]

In [67]:

```
c[3]
```

```
3
```



1次元配列の要素数え方

0

1

2

3

4

["Steak", "Pizza", "Noodle", "Fruit", "Diet"]

-5

-4

-3

-2

-1



1次元配列の要素指定

`c[-1]`

```
In [68]:
```

```
c[-1]
```

5

`c[10]`

```
In [69]:
```

```
c[10]
```

IndexError

Traceback (most recent call last)

<ipython-input-69-ea0c20b3b68b> in <module>()

----> 1 c[10]

IndexError: index 10 is out of bounds for axis 0 with size 6



要素の変換

配列Cを作る

c = np.arange(6)

In [27]:

```
c = np.arange(6)
```

```
c
```

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

配列Cの第1位の要素を
99に変更する。

c[0] = 99

In [28]:

```
c[0] = 99
```

```
c
```

```
array([99, 1, 2, 3, 4, 5])
```

配列Cの第3位から全ての
要素を66に変更する。

c[2:] = 66

In [30]:

```
c[2:] = 66
```

```
c
```

```
array([99, 1, 66, 66, 66, 66])
```



2次元配列の要素指定

まず、2次元の配列を作ります

d = np.array([np.arange(0,5),np.arange(5,10),np.arange(10,15)])

```
In [75]: d = np.array([np.arange(0,5),np.arange(5,10),np.arange(10,15)])  
d  
  
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14]])
```

d[0]

```
In [77]: d[0]  
  
array([0, 1, 2, 3, 4])
```



2次元配列の要素指定

`d[1][4]`

```
In [78]:
```

```
d[1][4]
```

9

`d[2][2:5]`

```
In [79]:
```

```
d[2][2:5]
```

```
array([12, 13, 14])
```



要素範囲を指定

2:5は第3要素から第5要素まで



2次元配列の要素指定

	0列	1列	2列	3列	4列
0行	1	2	3	4	5
1行	6	7	8	9	10
2行	11	12	13	14	15

d [2] [2:5] ← 列を指定
↑
行を指定



3次元配列の要素指定

```
e = np.array([  
    [np.arange(0,5),np.arange(5,10),np.arange(10,15)],  
    [np.arange(15,20),np.arange(25,30),np.arange(35,40)],  
    [np.arange(45,50),np.arange(55,60),np.arange(65,70)],  
    [np.arange(75,80),np.arange(85,90),np.arange(95,100)]  
])
```

In [80]:

```
e = np.array([  
    [np.arange(0,5),np.arange(5,10),np.arange(10,15)],  
    [np.arange(15,20),np.arange(25,30),np.arange(35,40)],  
    [np.arange(45,50),np.arange(55,60),np.arange(65,70)],  
    [np.arange(75,80),np.arange(85,90),np.arange(95,100)]  
])
```



3次元配列の要素指定

第1層を抽出する
e[0]

```
In [85]:
```

```
e.shape
```

```
(4, 3, 5)
```

```
In [83]:
```

```
e[0]
```

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

練習

上記の「e」を練習サンプルにし、下記の配列を抽出してください。

1. [27, 28, 29]

2. 88

(Hit:配列の要素が3つから2つになりました。なぜでしょうか?)

In [86]:

e

```
array([[[ 0, 1, 2, 3, 4],  
        [ 5, 6, 7, 8, 9],  
        [10, 11, 12, 13, 14]],  
  
       [[15, 16, 17, 18, 19],  
        [25, 26, 27, 28, 29],  
        [35, 36, 37, 38, 39]],  
  
       [[45, 46, 47, 48, 49],  
        [55, 56, 57, 58, 59],  
        [65, 66, 67, 68, 69]],  
  
       [[75, 76, 77, 78, 79],  
        [85, 86, 87, 88, 89],  
        [95, 96, 97, 98, 99]]])
```




練習解答

1. [27, 28, 29]

e[1][1]

In [94]:

e[1][1][2:]

array([27, 28, 29])

2. 88

e[3][1][3]

In [90]:

e[3][1][3]

88

5. 配列の計算

5.1 配列ごとの四則演算

5.2 配列同士の四則演算

5.3 他の数学計算

5.4 ベクトルの内積

5.5 行列の積 `np.dot`

5.6 練習



配列ごとの四則演算

`f = np.array([[1,2,3,4], [8,9,10,11]])`

```
In [3]: f = np.array([[1,2,3,4], [8,9,10,11]])  
f
```

```
array([[ 1,  2,  3,  4],  
       [ 8,  9, 10, 11]])
```

`f - 1`

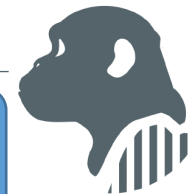
```
In [4]: f - 1
```

```
array([[ 0,  1,  2,  3],  
       [ 7,  8,  9, 10]])
```

`f * 3`

```
In [5]: f * 3
```

```
array([[ 3,  6,  9, 12],  
       [24, 27, 30, 33]])
```



配列ごとの四則演算

$1 / f$

In [6]:

$1 / f$

```
array([[1.    , 0.5    , 0.33333333, 0.25   ],  
       [0.125  , 0.11111111, 0.1    , 0.09090909]])
```

** は2乗

$f ** 2$

In [7]:

$f ** 2$

```
array([[ 1,  4,  9, 16],  
       [64, 81, 100, 121]])
```



配列同士の四則演算

$f - f$

In [8]:

$f - f$

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

$f * f$

In [9]:

$f * f$

```
array([[ 1,  4,  9, 16],  
       [64, 81, 100, 121]])
```



他の数学計算

max : 最大値

```
In [50]: np.max(e)
```

99

min : 最小値

```
In [51]: np.min(e)
```

0

sum : 合算値

```
In [52]: np.sum(e)
```

2670

mean : 平均値

```
In [53]: np.mean(e)
```

44.5

median : 中央値

```
In [56]: np.median(e)
```

42.0



ベクトルの内積

ベクトル a , b を設定した時、内積は $a \cdot b$ と表すことができます。

$$\vec{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}, \vec{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$\vec{a} \cdot \vec{b} = a_1 * b_1 + a_2 * b_2 + \cdots + a_n * b_n$$



行列の積 np.dot

数学（特に線形代数）、統計学などでよく使われます。

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$A \cdot B = \begin{pmatrix} a_{11} * b_{11} + a_{12} * b_{21} & a_{11} * b_{12} + a_{12} * b_{22} \\ a_{21} * b_{11} + a_{22} * b_{21} & a_{21} * b_{12} + a_{22} * b_{22} \end{pmatrix}$$



行列の積 np.dot

np.dotで先ほどの行列の積やベクトルの内積などが簡単に計算できます。

g = np.arange(3)

In [32]:

```
g = np.arange(3)
```

```
g
```

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

h = np.arange(3,6)

In [35]:

```
h = np.arange(3,6)
```

```
h
```

```
array([3, 4, 5])
```

np.dot(g,h)

In [36]:

```
np.dot(g,h)
```

14



練習

1. $A = \text{np.array}([5, 10, 15])$ をベースに下記の配列を作成してください

a). $[35, 50, 65]$

b). $[30, 160]$

例 : $A + [5, 10, 15] = [10, 20, 30]$

Hit : A中に要素が3つあるにも関わらず、bの中に要素が2つしかありません。それはなぜでしょうか？

練習解答

1. $A = \text{np.array}([5, 10, 15])$ をベースに下記の配列を作成してください

a). $[35, 50, 65]$

$A * 3 + 20$

b). $[30, 160]$

$B = \text{np.array}([[1, 1, 1], [4, 5, 6]])$

$\text{np.dot}(B, A)$

```
In [70]: A = np.array([5, 10, 15])  
A
```

```
array([ 5, 10, 15])
```

```
In [80]: A * 3 + 20
```

```
array([35, 50, 65])
```

```
In [77]: B = np.array([[1, 1, 1], [4, 5, 6]])  
B
```

```
array([[1, 1, 1],  
       [4, 5, 6]])
```

```
In [78]: np.dot(B, A)
```

```
array([ 30, 160])
```

6. 他によく使われる関数

6.1 reshape

6.2 transpose

6.3 random.rand

6.4 linspace

6.5 append

6.6 flatten

6.7 all

6.8 any

6.9 where



1. reshape

reshape

配列の形状を変換する

```
In [67]:
```

```
i = np.arange(9)  
i
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
In [72]:
```

```
# 配列の形状を変換する  
i.reshape(3,3)
```

```
array([[0, 1, 2],  
       [3, 4, 5],  
       [6, 7, 8]])
```



2. transpose

transpose

縦軸と横軸を変換する

```
In [70]:
```

```
# 縦軸と横軸を変換する
```

```
i.reshape(3,3).transpose()
```

```
array([[0, 3, 6],  
       [1, 4, 7],  
       [2, 5, 8]])
```



3. random.rand

random.rand

0~1 の中で乱数を作成する

```
In [71]:
```

```
# 0~1 で乱数を作成する
```

```
j = np.random.rand()
```

```
j
```

```
0.49717082814166713
```



4. linspace

Linspace0

決まった空間をX均等分にする

```
In [75]:
```



```
# 決まった空間をX均等分にする
```

```
k = np.linspace(0, 20, 5)
```

```
k
```

```
array([ 0.,  5., 10., 15., 20.])
```




5. append

append

配列の末尾で要素を追加する

```
In [76]: # 配列の末尾で要素を追加する  
         l = np.append(k, [25, 30])  
         l
```

```
array([ 0.,  5., 10., 15., 20., 25., 30.])
```



6. flatten

flatten

高次元配列を1次元に変換する

In [78]:

```
# 高次元配列を1次元に変換する  
m = np.arange(12).reshape(3,4)  
m
```

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

In [80]:

```
m.shape
```

```
(3, 4)
```

In [81]:

```
m.flatten()
```

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



7. all

all

全て要素が条件満たす時「True」

それ以外は「False」

```
In [82]: np.all(l > 20)
```

False

```
In [83]: np.all(l < 50)
```

True



8. any

any

一つ要素だけ条件を満たす時

「True」、それ以外は「False」

```
In [84]:
```

```
np.any(l == 5)
```

```
True
```



9. where

where

条件を満たす部分を「True」の値
を取る、それ以外は「False」の
値

```
In [85]:
```

```
np.where(l<20, -11, 22)
```

```
array([-11, -11, -11, -11, 22, 22, 22])
```

7. 総合演習

7.1 乱数10個の配列を作成し、その中の最大値と最小値の差を計算してください

7.2 下記のコードの実行結果を考えてみてください

7.3 下記のコードの実行結果を考えてみてください



練習

1. 乱数10個の配列を作成し、その中の最大値と最小値の差を計算してください

2. 下記のコードの実行結果を考えてみてください

```
>>> C = np.arange(40).reshape(4,5,2)
```

```
>>> C[2][2:] = 14
```

```
>>> C
```

3. 下記のコードの実行結果を考えてみてください

```
>>> D = np.where(np.linspace(-1, 1, 40).reshape(4,5,2) >= 0, 1, -1)
```

```
>>> D
```



練習解答 1

1. 乱数10個の配列を作成し、その中の最大値と最小値の差を計算してください

```
In [161]:
```

```
E = np.random.rand(10)  
E
```

```
array([0.00939707, 0.91384791, 0.38847324, 0.42054054, 0.01074836,  
       0.92085177, 0.78691038, 0.65631273, 0.32096928, 0.91117594])
```

```
In [163]:
```

```
np.max(E)-np.min(E)
```

```
0.9114547006388224
```




練習解答 2

2. 下記のコードの実行結果を考えてみてください

```
>>> C = np.arange(40).reshape(4,5,2)
```

0~39の1次元配列を作成し、それを4「層」x 5「行」x 2「列」の3次元配列に変換する

```
>>> C[2][2:] = 14
```

変換後の配列の第3層の第3行から全て列の値を14に変える

```
>>> C
```

C配列を表示する

練習解答 2

```
C = np.arange(40).reshape(4, 5, 2)
```

```
C[2][2:] = 14
```

```
C
```

```
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],  
       [14, 14],  
       [14, 14],  
       [14, 14]],
```

```
      [[30, 31],  
       [32, 33],  
       [34, 35],  
       [36, 37],  
       [38, 39]]])
```





練習解答 3

3. 下記のコードの実行結果を考えてみてください

```
>>> D = np.where(np.linspace(-1, 1, 40).reshape(4,5,2) >= 0, 1, -1)
```

np.linspace(-1, 1, 40)は-1から1まで、40区間分に切って、結果を1次元配列に入れる

reshape(4,5,2)は元の配列を 4「層」x 5「行」x 2「列」の3次元配列に変換する

np.where(____ >= 0.5, 1, D) は0.5より大きい数値を1に、それ以外をそのまま残すように変換する

```
>>> D
```

結果を表示する

練習解答 3

```
D = np.where(np.linspace(-1, 1, 40).reshape(4, 5, 2) >= 0, 1, -1)
D
```

```
array([[[[-1, -1],
          [-1, -1],
          [-1, -1],
          [-1, -1]],
```

```
        [[-1, -1],
          [-1, -1],
          [-1, -1],
          [-1, -1],
          [-1, -1]],
```

```
        [[ 1, 1],
          [ 1, 1],
          [ 1, 1],
          [ 1, 1],
          [ 1, 1]],
```

```
        [[ 1, 1],
          [ 1, 1],
          [ 1, 1],
          [ 1, 1],
          [ 1, 1]]])
```

