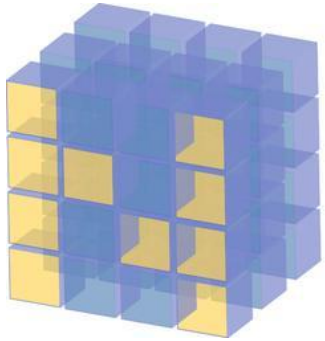
神经网络&深度学习 Google TENSORFLOW



05 NumPy科学计算库

西安科技大学 牟琦
muqi@xust.edu.cn

神经网络&深度学习 Google
TENSORFLOW

5.1 多维数组

中国大学MOOC

在**机器学习**中，需要把输入数据转变为**多维数组**的形式

数字

语文
score

85

score=85

一维数组

语文	数学	英语	历史	地理
score[0]	score[1]	score[2]	score[3]	score[4]
85	72	61	92	80

score=[85, 72, 61, 92, 80]

score[i]

score[0]=85

score[1]=72

.....

score[4]=80

二维数组

课程 \ 学生	语文 0	数学 1	英语 2	历史 3	地理 4
0	85	72	61	92	80
1	78	89	81	95	82
...	...	...	...	...	...
28	92	85	86	90	85
29	61	84	74	89	78

30行

score[i , j]

score[0,0]=85

score[0,1]=72

score[20,4]=78

score=[[85,72,61,92,80],
[78,89,81,95,82],
...,
[61,84,74,89,78]]

score=[[85,72,61,92,80],[78,89,81,95,82],...,[61,84,74,89,78]]

- 数组的**形状**（Shape）：描述数组的**维度**，以及各个维度内部的**元素个数**。

$$(n_0, n_1, \ldots, n_i, \ldots n_m)$$

一维数组

shape: (5,)

85	72	61	92	80

[85, 72, 61, 92, 80]

二维数组

shape: (30, 5)

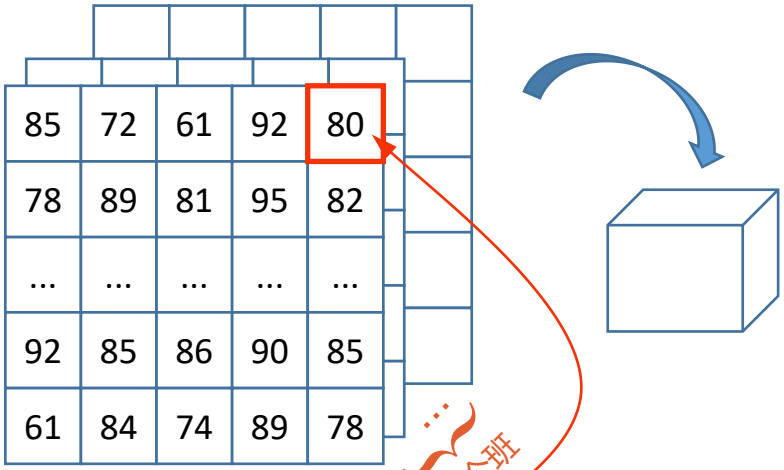
85	72	61	92	80
78	89	81	95	82
…	…	…	…	…
92	85	86	90	85
61	84	74	89	78

30行

score[i , j]

[[85,72,61,92,80],
[78,89,81,95,82],
…,
[61,84,74,89,78]]

三维数组

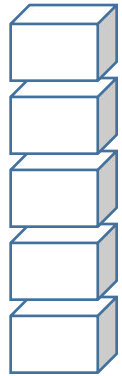


score[i , j, k]

score[0,0,4]= 80

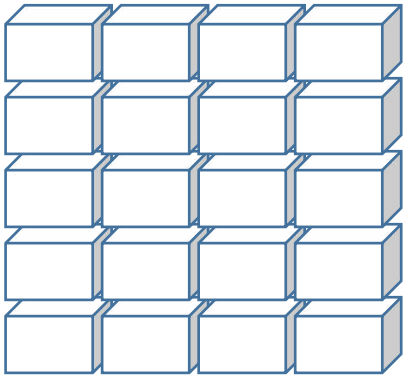
shape: (10,30, 5)

四维数组



shape: (5,10,30, 5)

五维数组



shape: (4,5,10,30, 5)

多维数组

$(n_0, n_1, \ldots, n_i, \ldots n_m)$

形状（Shape）：是一个**元组**，描述数组的**维度**，以及各个维度的**长度**。

长度（Length）：某个维度中的**元素个数**。

一维数组：

[0, 1, 2, 3]

shape=(4，)

二维数组

[[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11]]

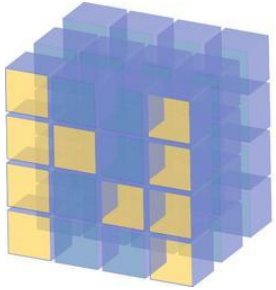
shape=（3, 4 ）

三维数组

[[[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11]],
[[12, 13, 14, 15],
[16, 17, 18, 19],
[20, 21, 22, 23]]]

shape=（2, 3, 4 ）

5.2 创建NumPy数组



NumPy ——Numeric Python

- 提供了**多维数组**、**矩阵**的常用操作和一些高效的**科学计算函数**。
- 底层运算通过C语言实现，处理**速度快**、**效率高**，适用于**大规模多维数组**。
- 可以直接完成**数组**和**矩阵**运算，**无需循环**。

安装NumPy库

- Anaconda：在Anaconda中，已经被安装了NumPy
- pip安装

```
pip install numpy
```

导入NumPy库

```
import numpy as np
```

在调用Numpy中的函数时，一定要加上前缀**np**

```
from numpy import *
```

在调用Numpy中的函数时，可以不加前缀

创建数组

array([列表]/(元组))

```
>>>a=np.array([0,1,2,3])

>>> a
array([0, 1, 2,3])

>>>print(a)
[0,1,2,3]

>>>type(a)
<class 'numpy.ndarray '>
```

输出指定的数组元素

```
>>>a[0]
0
>>>print(a[1],a[2],a[3])
1  2  3

>>>a[ 0:3 ]
array([ 0, 1, 2 ])
>>>a[ :3]
array([ 0, 1, 2 ])
>>>a[0: ]
array([ 0, 1, 2 ,3])
```

数组的属性

属　性	描　述
ndim	数组的维数
shape	数组的形状
size	数组元素的总个数
dtype	数组中元素的数据类型
itemsize	数组中每个元素的字节数

```
>>>a=np.array([0,1,2,3])
>>>a
array([0, 1, 2,3])

>>>a.ndim
1
>>>a.shape
(4,)
>>>a.size
4
```

二维数组

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

```
>>>b.ndim
2
>>>b.shape
(3, 4)
>>>b.size
12
```

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

```
>>>b[0].ndim
1
>>>b[0].shape
(4, )
>>>b[0].size
4
```

二维数组

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

```
>>>b[0][0]
0
>>>b[0][1]
1
```

```
>>>b[0,0]
0
>>>b[0,1]
1
```

三维数组

```
>>>t= np.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]]])
>>> t
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]]])
```

t[0]

t[1]

```
>>>t.ndim
3
>>>t.shape
(2, 3, 4)
>>>t.size
24
```

```
>>>t[0].ndim
2
>>>t[0].shape
(3,4)
>>>t[0].size
12
```

三维数组

```
>>>t= np.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]]])
>>> t
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]]])
```

```
>>>t[0][0]
array([0, 1, 2, 3])
>>>t[0][0].ndim
1
```

```
>>>t[0][0].shape
(4,)
>>>t[0][0].size
4
```

```
>>>t[0][0][0]
0
>>>t[0,0,0]
0
```

数组元素的数据类型

NumPy要求数组中所有元素的**数据类型**必须是**一致**的

- int8、uint8、int16、uint16、int32、uint32、int64、uint64
- float16、float32、float64、float128
- complex64、complex128、complex256
- bool、object、string_、unicode_

array([列表]/(元组), dtype=数据类型)

int64、'int64'、"np.int64"

```
>>>a= np.array( [0, 1,2, 3 ], dtype=np.int64)
>>>a
array([0,1, 2, 3], dtype=int64)
```

```
>>> a.dtype
dtype('int64')
>>>a.itemsize
8
```

每个元素所占的字节数

```
>>>c=np.array([1.2, 3.5, 5.1])
>>>c.dtype
dtype('float64')
```

在使用Python列表或元组、创建NumPy数组时，所创建的数组类型，**由原来的元素类型推导而来**。

创建特殊的数组

函　数	功能描述
np.arange()	创建数字序列数组
np.ones()	创建全1数组
np.zeros()	创建全0数组
np.eye()	创建单位矩阵
np.linspace()	创建等差数列
np.logspace()	创建等比数列

- arange()函数：创建一个由**数字序列**构成的数组。

np.arange(起始数字, 结束数字, 步长, dtype)

前闭后开：数字序列中**不包括结束数字**
起始数字省略时，默认从0开始
步长省略时，默认为1

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

```
>>>d = np.arange(0, 2, 0.3)
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

- ones()函数：创建一个**元素全部为1**的数组

np.ones(shape, dtype)

```
>>> np.ones((3,2),dtype=np.int16)
array([[1, 1],
       [1, 1],
       [1, 1]], dtype=int16)
```

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

当数据类型省略时，这里仍然有**2层括号**

- zeros()函数：创建一个**元素全部为0**的数组

```
np.zeros( shape, dtype )
```

```
>>> np.zeros((2,3))
array([[0., 0., 0.],
       [0., 0., 0.]])
```

- eye()函数：创建一个**单位矩阵**。

```
np.eye( shape, dtype )
```

```
>>> np.eye(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
>>> np.eye(2,3)
array([[1., 0., 0.],
       [0., 1., 0.]])
```

- linspace()函数：创建**等差数列**。

参数：起始数字、结束数字、元素个数、元素数据类型

```
np.linspace(start, stop, num=50,  dtype)
```

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

- logspace()函数：创建一个**等比数列**。

参数：起始指数、结束指数、元素个数、基、 、元素数据类型

```
np.logspace(start, stop, num=50, base=10,  dtype)
```

```
>>> np.logspace(1,5,5, base=2)
array([  2.,   4.,   8.,  16.,  32.])
```

- **asarray()函数：** 将**列表**或**元组**转化为数组对象。

```
import numpy as np

list1=[[1,1,1],[1,1,1],[1,1,1]]
arr1=np.array(list1)
arr2=np.asarray(list1)

list1[0][0]=3

print('list1:\n',list1)
print('arr1:\n',arr1)
print('arr2:\n',arr2)
```

运行结果：

```
list1:
[[3, 1, 1], [1, 1, 1], [1, 1, 1]]
arr1:
[[1 1 1]
 [1 1 1]
 [1 1 1]]
arr2:
[[1 1 1]
 [1 1 1]
 [1 1 1]]
```

当数据源本身是一个**ndarray对象**时，array()会**复制**出一个副本，占用新的内存，而asarray()则**不复制副本**，它直接引用**原数组**。

```
import numpy as np

arr1=np.ones((3,3))
arr2=np.array(arr1)
arr3=np.asarray(arr1)

arr1[0][0]=3

print('arr1:\n',arr1)
print('arr2:\n',arr2)
print('arr3:\n',arr3)
```

```
arr1:
[[ 3.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
arr2:
[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
arr3:
[[ 3.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
```


tensorflow / tensorflow
TensorFlow: Open sou
Clone this repo:
git clone https://tensor
Branches
master

5.3 数组运算

数组元素的切片—— 一维数组

可以使用切片来访问NumPy数组中的一部分，切片方法和Python序列数据结构的切片一样

```
>>>a=np.array([0,1,2,3])
>>>a[ 0:3 ]
array([ 0, 1, 2 ])

>>>a[ :3]
array([ 0, 1, 2 ])

>>>a[0: ]
array([ 0, 1, 2 ,3])
```

数组元素的切片——二维数组

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

```
>>>b[0]
array([0,1,2,3])

>>> b[0:2]
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])

>>> b[:2]
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

```
>>>b[0:2, 0:2]
array([[0, 1],
       [4, 5]])

>>>b[0:2,1:3]
array([[1, 2],
       [5, 6]])

>>>b[:,0]
array([0, 4, 8])
```

数组元素的切片——三维数组

```
>>>t= np.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]]])
>>> t
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]]])
```

```
>>>t[:, :, 0]
array([[ 0,  4,  8],
       [12, 16, 20]])
```

```
>>> t[:, :, 1]
array([[ 1,  5,  9],
       [13, 17, 21]])
```

改变数组的形状

函 数	功能描述
np.reshape(shape)	**不改变当前数组**，按照shape创建新的数组
np.resize(shape)	**改变当前数组**，按照shape创建数组

```
>>>b = np.arange(12)
>>>b
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ])
```

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

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

当改变形状时，应该考虑到数组中元素的个数，确保改变前后，**元素总个数相等**。

```
>>>b = np.arange(12)
>>>b
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ])
```

```
>>>b.reshape(2,5)
ValueError: cannot reshape array of size 12 into shape(2,5)
```

错误提示

创建数组并且改变数组形状

```
>>>b = np.arange(12)
>>>b.reshape(3,4)
```

```
>>>b = np.arange(12).reshape(3,4)
>>>b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
>>>t = np.arange(24).reshape(2, 3, 4)
>>>t
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]]])
```

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

参数-1：根据数组中元素总个数、以及其他维度的取值，来**自动计算**出这个维度的取值。

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

数组间的运算

```
>>> a = np.array([0,1,2,3])
>>> d = np.array([2,3,4,5])
>>> a+d
array([[ 2, 4, 6, 8]])
```

相加的2个数组的**形状**和**长度**应该一致，否则就会出现错误

```
>>> d = np.array([1,2])                    #创建一维数组
>>> a+d                                    #数组内的元素个数不同，无法相加
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-3-637545a26abd> in <module>()
----> 1 a+d

ValueError: operands could not be broadcast together with shapes (4,) (2,)
```

一维数组可以和**多维数组**相加，相加时会将一维数组**扩展**至多维。

```
>>>a = np.array([0,1,2,3])
>>>b = np.array([[0,1,2,3],[4,5,6,7],[8,9,10,11]])
>>> a+b
array([[ 0,  2,  4,  6],
       [ 4,  6,  8, 10],
       [ 8, 10, 12, 14]])
```

```
>>>b ** 2
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121]], dtype=int32)
```

- 数组之间的**减法**、**乘法**、**除法**运算，和加法运算规则相同。
- 当两个数组中元素的**数据类型不同**时，精度低的数据类型，会**自动转换**为精度更高的数据类型，然后再进行运算。

矩阵运算——矩阵乘法

乘号运算符：矩阵中对应的元素分别相乘

A B A*B

$$\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\times\begin{bmatrix}2 & 0\\ 3 & 4\end{bmatrix}=\begin{bmatrix}2 & 0\\ 0 & 4\end{bmatrix}$$

```
>>>A = np.array([[1,1],[0,1]])
>>>B = np.array([[2,0],[3,4]])
>>>C = A * B
>>>C
array([[2, 0],
       [0, 4]])
```

矩阵相乘：按照**矩阵相乘**的规则运算

A B 矩阵相乘

$$\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\times\begin{bmatrix}2 & 0\\ 3 & 4\end{bmatrix}=\begin{bmatrix}5 & 4\\ 3 & 4\end{bmatrix}$$

```
>>>np.matmul(A,B)
array([[5, 4],
       [3, 4]])
```

```
>>> np.dot(A,B)
array([[5, 4],
       [3, 4]])
```

矩阵运算——矩阵乘法

- @运算符：连续进行矩阵乘法

A B C 矩阵相乘

$$\begin{bmatrix}1 & 1\\0 & 1\end{bmatrix}\times\begin{bmatrix}2 & 0\\3 & 4\end{bmatrix}\times\begin{bmatrix}0 & 0\\1 & 1\end{bmatrix}=\begin{bmatrix}9 & 4\\7 & 4\end{bmatrix}$$

```
>>> A=np.array([[1,1],[0,1]])
>>> B=np.array([[2,0],[3,4]])
>>> C=np.array([[1,0],[1,1]])
>>> np.matmul(np.matmul(A,B),C)
array([[9, 4],
       [7, 4]])
>>> A@B@C
array([[9, 4],
       [7, 4]])
```

矩阵运算——转置和求逆

矩阵转置——np.transpose()

```
>>>np.transpose(A)
array([[1, 0],
       [1, 1]])

>>>np.transpose(B)
array([[2, 3],
       [0, 4]])
```

矩阵求逆——np.linalg.inv ()

```
>>>np.linalg.inv(A)
array([[ 1., -1.],
       [ 0.,  1.]])

>>>np.linalg.inv(B)
array([[ 0.5  ,  0.   ],
       [-0.375,  0.25 ]])
```

数组元素间的运算

函 数	功能描述
numpy.sum()	计算所有元素的和
numpy.prod()	计算所有元素的乘积
numpy.diff()	计算数组的相邻元素之间的差
np.sqrt()	计算各元素的平方根
np.exp()	计算各元素的指数值
np.abs()	取各元素的绝对值

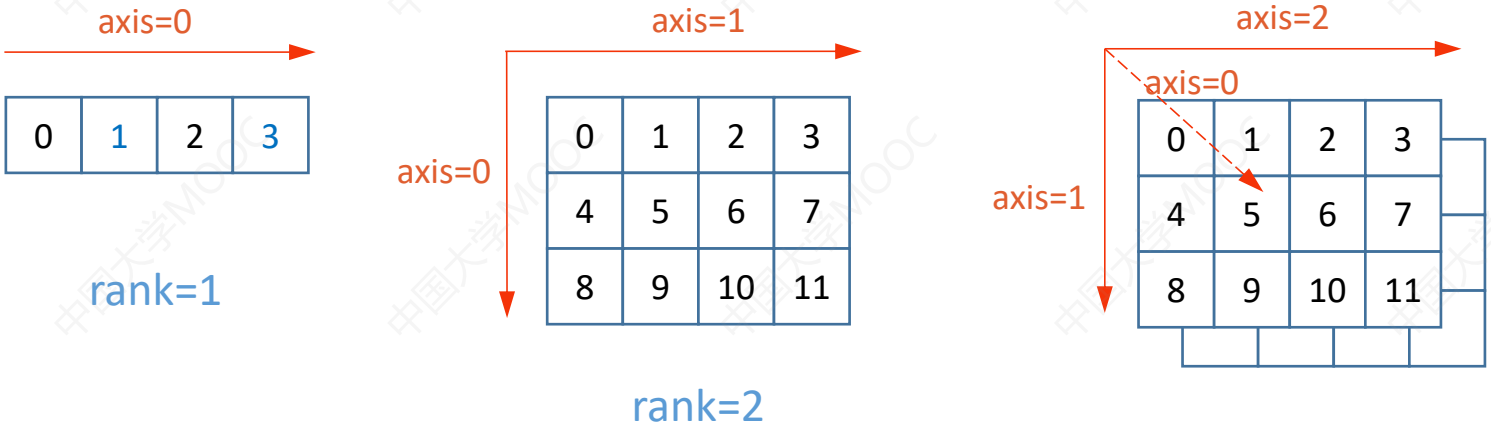
sum()——对数组中所有元素求和

```
>>>a=np.arange(4)
array([0, 1, 2, 3])

>>>np.sum(a)
6
```

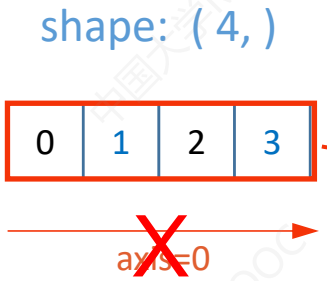
```
>>>b=np.arange(12).reshape(3,4)
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

>>>np.sum(b)
66
```

按行求和&按列求和

- 轴（axes）：数组中的每一个维度被称为一个轴
- 秩（rank）：轴的个数

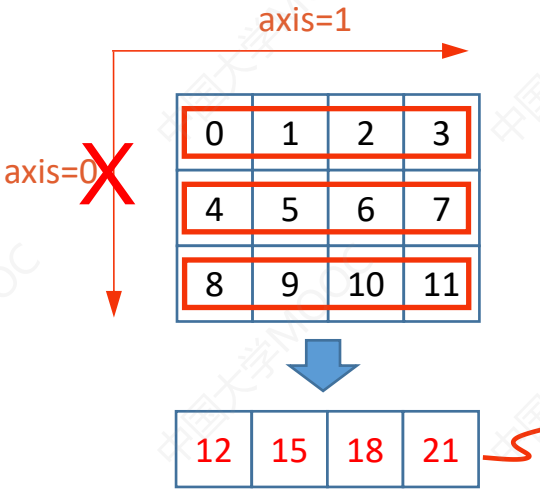


- 一维数组（rank=1）



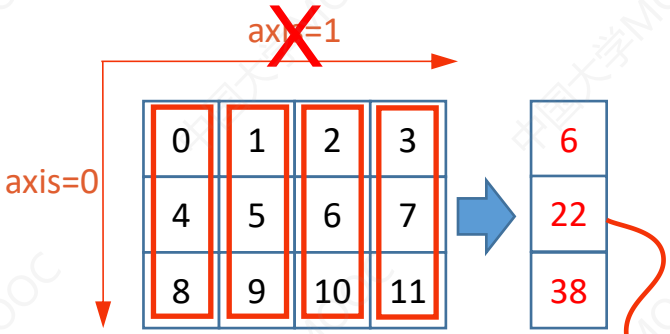

```
>>>a=np.arange(4)
>>>a
array([0, 1, 2, 3])

>>>np.sum(a)
6
```

二维数组（rank=2）




```
>>>b=np.arange(12).reshape(3,4)
>>>b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

>>>np.sum(b,axis=0)
array([12, 15, 18, 21])

>>>np.sum(b,axis=1)
array([ 6, 22, 38])
```

二维数组（rank=2）




```
>>>b=np.arange(12).reshape(3,4)
>>>b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

>>>np.sum(b,axis=0)
array([12, 15, 18, 21])

>>>np.sum(b,axis=1)
array([ 6, 22, 38])
```

- 三维数组（rank=3）

axis = 0，1，2

```
>>>t=np.arange(24).reshape(X,3,4)
>>>t
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]]])
```

```
>>>np.sum(t,axis=0)
array([[12, 14, 16, 18],
       [20, 22, 24, 26],
       [28, 30, 32, 34]])

>>>print(np.sum(t,axis=1)
array([[12, 15, 18, 21],
       [48, 51, 54, 57]])

>>>np.sum(t,axis=2)
array([[ 6, 22, 38],
       [54, 70, 86]])
```

三维数组（rank=3）

axis = 0, 1, 2

```
>>>t=np.arange(24).reshape(2,3,4)
>>>t
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]]])
```

12, 15, 18, 21

48, 51, 54, 57

```
>>>np.sum(t,axis=0)
array([[12, 14, 16, 18],
       [20, 22, 24, 26],
       [28, 30, 32, 34]])

>>>print(np.sum(t,axis=1)
array([[12, 15, 18, 21],
       [48, 51, 54, 57]])

>>>np.sum(t,axis=2)
array([[ 6, 22, 38],
       [54, 70, 86]])
```

- 三维数组（rank=3）

axis = 0, 1, 2

```
>>>t=np.arange(24).reshape(2,3,4)
>>>t
array([[[ 0,  1,  2,  3],      6
        [ 4,  5,  6,  7],      22
        [ 8,  9, 10, 11]],     38

       [[12, 13, 14, 15],      54
        [16, 17, 18, 19],      70
        [20, 21, 22, 23]]])    86
```

```
>>>np.sum(t,axis=0)
array([[12, 14, 16, 18],
       [20, 22, 24, 26],
       [28, 30, 32, 34]])

>>>print(np.sum(t,axis=1))
array([[12, 15, 18, 21],
       [48, 51, 54, 57]])

>>>np.sum(t,axis=2)
array([[ 6, 22, 38],
       [54, 70, 86]])
```

- 四维数组（rank=4）

axis = 0, 1, 2, 3

t.shape: (5,10,30, 5)

np.sum(t,axis=0) (10,30, 5)

np.sum(t,axis=1) (5,30, 5)

数组元素间的运算

函 数	功能描述
numpy.sum()	计算所有元素的和
numpy.prod()	计算所有元素的乘积
numpy.diff()	计算数组的相邻元素之间的差
np.sqrt()	计算各元素的平方根
np.exp()	计算各元素的指数值
np.abs()	取各元素的绝对值

sqrt()函数实例

```
>>> d = np.logspace(1,4,4,base=2)
>>> d
array([  2.,   4.,   8.,  16.])

>>> np.sqrt(d)
array([ 1.41421356,  2.        ,  2.82842712,  4.        ])
```

数组堆叠运算

np.stack((数组1, 数组2,...) , axis)

一维数组

x

1	2	3

y

4	5	6

shape=(3,)

axis=0

1	2	3
4	5	6

shape=(2, 3)

axis=1

1	4
2	5
3	6

shape=(3, 2)

一维数组堆叠运算

```
>>> x = np.array([1, 2, 3])          #创建一维数组x
>>> y = np.array([4, 5, 6])          #创建一维数组y

>>> np.stack((x, y), axis=0)         #在轴=0上堆叠
array([[1, 2, 3],
       [4, 5, 6]])

>>> np.stack((x, y), axis=1)         #在轴=1上堆叠
array([[1, 4],
       [2, 5],
       [3, 6]])
```

实例

```
area=[137.97,104.50,100.00,124.32,79.20,99.00,124.00,114.00,
      106.69,138.05,53.75,46.91,68.00,63.02,81.26,86.21]
room=[3,2,2,3,1,2,3,2,2,3,1,1,1,1,2,2]
b0 = np.ones(16)

np.stack((area,room,b0), axis = 1)
```

执行NumPy函数前，首先**自动把Python列表转换为NumPy数组，和数据类型转换**，然后再进行堆叠

a：Python列表，数据元素为浮点数
room：Python列表，数据元素为整数
b0：NumPy数组，全部元素为1

实例

```
area=[137.97,104.50,100.00,124.32,79.20,99.00,124.00,114.00,
      106.69,138.05,53.75,46.91,68.00,63.02,81.26,86.21]
room=[3,2,2,3,1,2,3,2,2,3,1,1,1,1,2,2]
b0 = np.ones(16)

np.stack((area,room,b0), axis = 1)
```

axis=1 堆叠 → (16,3)

```
array([[137.97,   3.  ,   1.  ],
       [104.5 ,   2.  ,   1.  ],
       [100.  ,   2.  ,   1.  ],
       [124.32,   3.  ,   1.  ],
       [ 79.2 ,   1.  ,   1.  ],
       [ 99.  ,   2.  ,   1.  ],
       [124.  ,   3.  ,   1.  ],
       [114.  ,   2.  ,   1.  ],
       [106.69,   2.  ,   1.  ],
       [138.05,   3.  ,   1.  ],
       [ 53.75,   1.  ,   1.  ],
       [ 46.91,   1.  ,   1.  ],
       [ 68.  ,   1.  ,   1.  ],
       [ 63.02,   1.  ,   1.  ],
       [ 81.26,   2.  ,   1.  ],
       [ 86.21,   2.  ,   1.  ]])
```

浮点数数组

二维数组堆叠

```
>>> m = np.array([[1, 2, 3], [4,5, 6], [7, 8, 9]])
>>> n = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])
>>> m.shape
(3, 3)
>>> n.shape
(3, 3)

>>> np.stack((m, n), axis=0).shape
(2, 3, 3)
>>> np.stack((m, n), axis=1).shape
(3, 2, 3)
>>> np.stack((m, n), axis=2).shape
(3, 3, 2)
```

神经网络&深度学习
TENSORFLOW
Google

5.4 矩阵和随机数

中国大学MOOC

矩阵——numpy.matrix

matrix (字符串/列表/元组/数组)

mat (字符串/列表/元组/数组)

参数为字符串

```
>>>a = np.mat(' 1  2  3 ;  4  5  6 ')
>>>a
matrix([[1, 2, 3],
        [4, 5, 6]])
```

参数为Python列表

```
>>>b = np.mat([[1, 2, 3],[4, 5, 6]])
>>>b
matrix([[1, 2, 3],
        [4, 5, 6]])
```

使用NumPy二维数组创建矩阵

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

>>>m = np.mat(a)
>>>m
matrix([[1, 2, 3],
        [4, 5, 6]])

>>>type(a)
<class 'numpy.ndarray'>
>>>type(m)
<class 'numpy.matrix'>
```

参数为NumPy数组

```
>>>m.ndim
2

>>>m.shape
(2, 3)

>>>m.size
6

>>>m.dtype
dtype( 'int32' )
```

矩阵对象的属性

属性	说 明
.ndim	矩阵的维数
.shape	矩阵的形状
.size	矩阵的元素个数
.dtype	元素的数据类型

矩阵运算——矩阵相乘

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 8 & 2 \end{bmatrix}$$

```
>>>a1=np.mat([[0,1],[2,3]])
>>>a2=np.mat([[1,1],[2,0]])
>>>a3=a1*a2
>>>a3
matrix([[ 2,  0],
        [ 8,  2]])
```

矩阵运算——转置、求逆

- 矩阵转置： .T
- 矩阵求逆： .I

```
>>>n=np.mat([[1,2],[-1,-3]])
>>>n
matrix([[ 1,  2],
        [-1, -3]])

>>>n.T
matrix([[ 1,  -1],
        [ 2,  -3]])

>>>n.I
matrix([[ 3,  2],
        [-1, -1]])

>>>n*n.I
matrix([[ 1,  0],
        [ 0,  1]])
```

一个矩阵**乘以它的逆**等于**单位阵**

非方阵的转置、求逆运算

```
>>>a = np.array([[1,2,3],[4,5,6]])
>>>m = np.mat(a)
>>>m
matrix([[1, 2, 3],
        [4, 5, 6]])

>>> m.T
matrix([[ 1,  4],
        [ 2,  5],
        [ 3,  6]])

>>> m.I
matrix([[-0.94444444,  0.44444444],
        [-0.11111111,  0.11111111],
        [ 0.72222222, -0.22222222]])
```

矩阵 VS **二维数组**

- 运算符号简单

  A*B

- 能够表示**高维数组**
- 数组更加灵活，速度更快

随机数模块——numpy.random

函　数	功能描述	返回值
np.random.rand(d0,d1,…,dn)	元素在[0,1)区间均匀分布的数组	浮点数
np.random.uniform(low,hige, size)	元素在[low,hige)区间均匀分布的数组	浮点数
numpy.random.randint(low,hige, size)	元素在[low,hige)区间均匀分布的数组	整　数
np.random.randn(d0,d1,…,dn)	产生标准正态分布的数组	浮点数
np.random.normal(loc, scale, size)	产生正态分布的数组	浮点数

产生随机数

```
#创建2*3的随机数组，取值是在[0,1)之间均匀分布的浮点数
>>> np.random.rand(2,3)
array([[ 0.84543047,  0.76450077,  0.87173984],
       [ 0.01204357,  0.1531674 ,  0.67527698]])

#参数为空，返回一个数字
>>> np.random.rand()
0.289648133266711
```

```
#创建3*2的随机数组，取值是在1至5之间均匀分布的浮点数
>>> np.random.uniform(1,5,(3,2))
array([[ 2.6034975 ,  1.12862773],
       [ 3.24128894,  4.97185042],
       [ 2.77097208,  1.48494326]])
```

```
#创建3*2的随机数组，取值是在1至5之间均匀分布的整数
>>> np.random.randint(1,5,(3,2))
array([[2, 2],
       [4, 1],
       [4, 2]])
```

```
#创建2*3的随机数组，符合标准正态分布
>>> np.random.randn(2,3)
array([[-0.94594743, -1.10163142, -0.40212785],
       [-1.04332498, -1.35958875,  1.5320874 ]])
```

```
#创建3*2的随机数组，符合正态分布，均值为0，方差为1
>>> np.random.normal(0,1,(3,2))
array([[ 1.48764022, -1.52437091],
       [ 0.73473077,  1.51170983],
       [-0.99776822, -0.89273968]])
```

```
#创建2*3的随机数组，符合标准正态分布
>>> np.random.randn(2,3)
array([[-0.94594743, -1.10163142, -0.40212785],
       [-1.04332498, -1.35958875,  1.5320874 ]])
```

```
#创建3*2的随机数组，符合正态分布，均值为0，方差为1
>>> np.random.normal(0,1,(3,2))
array([[ 1.48764022, -1.52437091],
       [ 0.73473077,  1.51170983],
       [-0.99776822, -0.89273968]])
```

因为是随机数，使用同样的语句，所得到的结果也是不同的。

```
>>>np.random.rand(2,3)
array([[0.19151945, 0.62210877, 0.43772774],
       [0.78535858, 0.77997581, 0.27259261]])

>>>np.random.rand(2,3)
array([[0.68346294, 0.71270203, 0.37025075],
       [0.56119619, 0.50308317, 0.01376845]])
```

- **伪随机数：** 由**随机种子**，根据一定的算法生成的。
- **随机种子：** 指定随机数生成时所用算法**开始**的整数值。
  - 如果使用相同的seed( )值，则每次生成的随即数都相同。
  - 如果不设置这个值，则系统根据时间来自己选择这个值，此时每次生成的随机数因时间差异而不同。
  - 设置的seed()值仅一次有效。
  - 随机数产生的算法，和系统有关。

seed()函数——设置随机种子

```
>>>np. random.seed(612)
>>>np.random.rand(2,3)
array([[0.14347163, 0.49589878, 0.95454587],
       [0.13751674, 0.85456667, 0.42853136]])

>>>np. random.seed(612)
>>>np.random.rand(2,3)
array([[0.14347163, 0.49589878, 0.95454587],
       [0.13751674, 0.85456667, 0.42853136]])

>>>np.random.rand(2,3)
array([[0.27646426, 0.80187218, 0.95813935],
       [0.87593263, 0.35781727, 0.50099513]])
```

采用seed()函数设置随机种子，产生了相同的随机数

采用seed()函数设置随机种子，产生了相同的随机数

没有设置随机种子，产生了不一样的结果。

shuffle()——打乱顺序函数

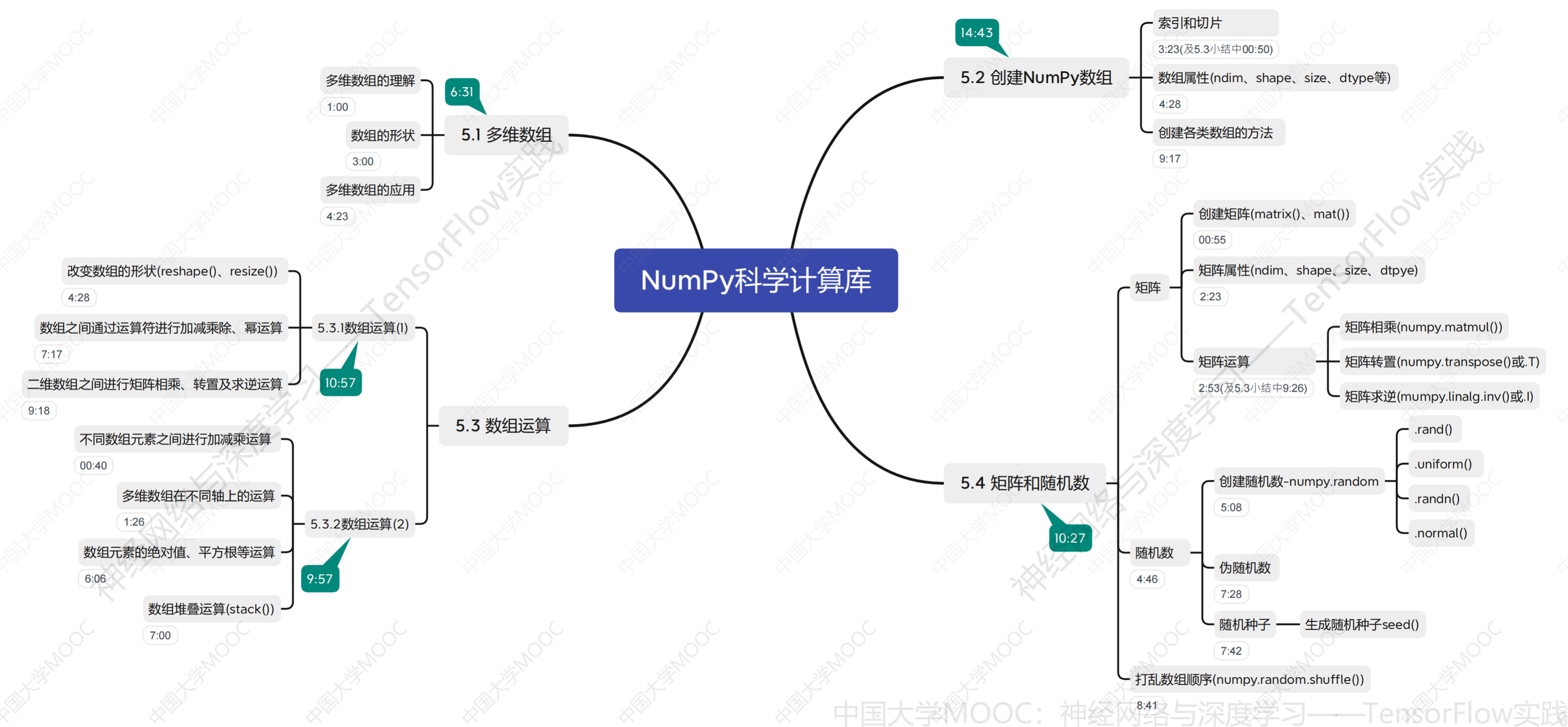
np.random.shuffle(序列)

Python列表、umPy数组等

```
>>> arr = np.arange(10)
>>>print(arr)
[0 1 2 3 4 5 6 7 8 9]

>>> np.random.shuffle(arr)
>>>print( arr)
[1 7 5 2 9 4 3 6 0 8]
```

对于多维数组，使用shuffle()函数只打乱第一维元素

```
>>>b=np.arange(12).reshape(4,3)
>>>b
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])

>>>np.random.shuffle(b)
>>>b
array([[ 3,  4,  5],
       [ 9, 10, 11],
       [ 6,  7,  8],
       [ 0,  1,  2]])
```



题目：生成数组并按要求输出。(20 分)

(1) 生成一个 4×4 的**整数**数组，数值为 1-100 的随机数，输出该数组的所有元素。(4 分)

(2) 分别输出数组中的前两列和后两列元素；(4 分)

(3) 分别输出数组中的第一列（列索引为 0）和最后一列元素，逐元素求和并输出；(4 分)

(4) 输入数字 n,输出行索引为 n 的一行元素；(4 分)

(5) 输入数字 m,n，输出行索引从 m 到 n 的元素。(4 分)