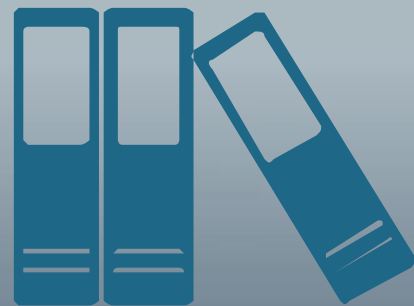
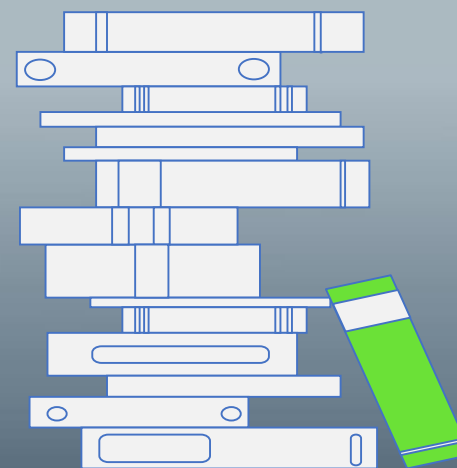


NumPy进阶





基础部分补充内容





矢量化运算

In the context of high-level languages like Python, Matlab, and R, the term **vectorization** describes the use of optimized, pre-compiled code written in a low-level language (e.g. C) to **perform mathematical operations over a sequence of data**. This is done in place of an explicit iteration written in the native language code (e.g. a “for-loop” written in Python).

在Python中，矢量化是指使用高度优化过的、提前编译好的、以C等低级语言编写的代码，来进行对数据序列的数学运算，以代替在Python中直接写循环来完成这类计算。

```
my_arr = np.arange(1000000)
my_list = list(range(1000000))

%time for _ in range(10): my_arr2 = my_arr * 2
%time for _ in range(10): my_list2 = [x * 2 for x in my_list]

CPU times: total: 0 ns
Wall time: 9.58 ms
CPU times: total: 281 ms
Wall time: 509 ms
```



矢量化运算

一元函数

一个输出元素依赖于一个输入元素

Unary Function: $f(x)$	NumPy Function
$ x $	<code>np.absolute</code>
\sqrt{x}	<code>np.sqrt</code>
Trigonometric Functions	
$\sin x$	<code>np.sin</code>
$\cos x$	<code>np.cos</code>
$\tan x$	<code>np.tan</code>
Logarithmic Functions	
$\ln x$	<code>np.log</code>
$\log_{10} x$	<code>np.log10</code>
$\log_2 x$	<code>np.log2</code>
Exponential Functions	
e^x	<code>np.exp</code>

二元函数

一个输出元素依赖于两个输入元素

Binary Function: $f(x, y)$	NumPy Function	Python operator
$x \cdot y$	<code>np.multiply</code>	<code>*</code>
$x \div y$	<code>np.divide</code>	<code>/</code>
$x + y$	<code>np.add</code>	<code>+</code>
$x - y$	<code>np.subtract</code>	<code>-</code>
x^y	<code>np.power</code>	<code>**</code>
$x \% y$	<code>np.mod</code>	<code>%</code>
$\max(x, y)$	<code>np.maximum</code>	
$\min(x, y)$	<code>np.minimum</code>	

```
arr1 = np.array([[3.7, -1.2, 2.6], [1, 3, 4]])
arr2 = np.array([[1.2, 0, 1.0], [2, 1, 0]])

np.maximum(arr1, arr2)

array([[3.7, 0. , 2.6],
       [2. , 3. , 4. ]])
```



矢量化运算

一个输出元素依赖于一个序列的输入元素

Sequential Function: $f(\{x_i\}_{i=0}^{n-1})$	NumPy Function	
Mean of $\{x_i\}_{i=0}^{n-1}$	<code>np.mean</code>	
Median of $\{x_i\}_{i=0}^{n-1}$	<code>np.median</code>	
Variance of $\{x_i\}_{i=0}^{n-1}$	<code>np.var</code>	<pre>arr1 = np.array([[3.7, -1.2, 2.6], [1, 3, 4]]) arr1.mean() 2.183333333333333 arr1.mean(axis=0) array([2.35, 0.9 , 3.3]) arr1.argmax(axis=0) array([0, 1, 1], dtype=int64)</pre>
Standard Deviation of $\{x_i\}_{i=0}^{n-1}$	<code>np.std</code>	
Maximum Value of $\{x_i\}_{i=0}^{n-1}$	<code>np.max</code>	
Minimum Value of $\{x_i\}_{i=0}^{n-1}$	<code>np.min</code>	
Index of the Maximum Value of $\{x_i\}_{i=0}^{n-1}$	<code>np.argmax</code>	
Index of the Minimum Value of $\{x_i\}_{i=0}^{n-1}$	<code>np.argmin</code>	
Sum of $\{x_i\}_{i=0}^{n-1}$	<code>np.sum</code>	



矢量化运算

一元函数（更完整的表格）

abs、fabs	计算整数、浮点数或复数的绝对值。对于非复数值，可以使用更快的fabs
sqrt	计算各元素的平方根。相当于 <code>arr ** 0.5</code>
square	计算各元素的平方。相当于 <code>arr ** 2</code>
exp	计算各元素的指数 e^x
log、log10、log2、log1p	分别为自然对数（底数为e）、底数为10的log、底数为2的log、 $\log(1 + x)$
sign	计算各元素的正负号：1（正数）、0（零）、-1（负数）
ceil	计算各元素的ceiling值，即大于等于该值的最小整数
floor	计算各元素的floor值，即小于等于该值的最大整数
arccos、arccosh、arcsin、arcsinh、arctan、arctanh	反三角函数
logical_not	计算各元素not x的真值。相当于 <code>-arr</code>



矢量化运算

二元函数（更完整的表格）

add	将数组中对应的元素相加
subtract	从第一个数组中减去第二个数组中的元素
multiply	数组元素相乘
divide、floor_divide	除法或向下圆整除法（丢弃余数）
power	对第一个数组中的元素A，根据第二个数组中的相应元素B，计算 A^B
maximum、fmax	元素级的最大值计算。fmax将忽略NaN
minimum、fmin	元素级的最小值计算。fmin将忽略NaN
mod	元素级的求模计算（除法的余数）
copysign	将第二个数组中的值的符号复制给第一个数组中的值
greater、greater_equal、less、less_equal、equal、not_equal	执行元素级的比较运算，最终产生布尔型数组。相当于中缀运算符>、>=、<、<=、==、!=
logical_and、logical_or、logical_xor	执行元素级的真值逻辑运算。相当于中缀运算符&、 、^



NumPy的数据类型

类型	类型代码	说明
int8、uint8	i1、u1	有符号和无符号的8位（1个字节）整型
int16、uint16	i2、u2	有符号和无符号的16位（2个字节）整型
int32、uint32	i4、u4	有符号和无符号的32位（4个字节）整型
int64、uint64	i8、u8	有符号和无符号的64位（8个字节）整型
float16	f2	半精度浮点数
float32	f4或f	标准的单精度浮点数。与C的float兼容
float64	f8或d	标准的双精度浮点数。与C的double和Python的float对象兼容
float128	f16或g	扩展精度浮点数
complex64、complex128、complex256	c8、c16、c32	分别用两个32位、64位或128位浮点数表示的复数
bool	?	存储True和False值的布尔类型
object	O	Python对象类型
string_	S	固定长度的字符串类型（每个字符1个字节）。例如，要创建一个长度为10的字符串，应使用S10
unicode_	U	固定长度的unicode类型（字节数由平台决定）。跟字符串的定义方式一样（如U10）

来源: <https://wizzardforcel.gitbooks.io/pyda-2e/content/4.html>

```
arr = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
arr
arr.astype(np.int32)
```

```
array([ 3, -1, -2,  0, 12, 10])
```

```
arr.astype('int32')
```

```
array([ 3, -1, -2,  0, 12, 10])
```

```
arr.astype('i4')
```

```
array([ 3, -1, -2,  0, 12, 10])
```

```
arr.astype('?')
```

```
array([ True,  True,  True,  True,  True,  True])
```

```
numeric_strings = np.array(['1.25', '-9.6', '42'],
                             dtype=np.string_)
```

```
numeric_strings
```

```
array([b'1.25', b'-9.6', b'42'], dtype='<S4')  
对于字符型数组，不恰当的指定字符串长度会导致字符被截断。
```

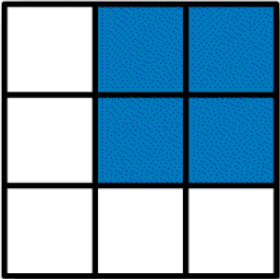
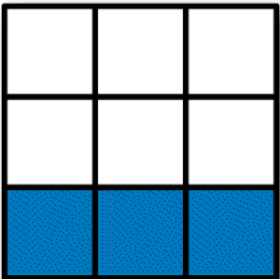
```
numeric_strings = np.array(['1.25', '-9.6', '42'], dtype='<S2')
```

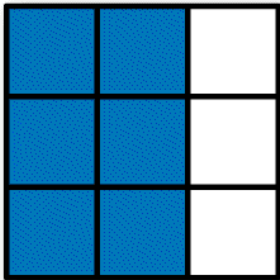
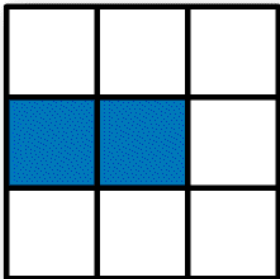
```
numeric_strings
```

```
array([b'1.', b'-9', b'42'], dtype='<S2')
```




切片索引

	Expression	Shape
	<code>arr[:2, 1:]</code>	<code>(2, 2)</code>
	<code>arr[2]</code>	<code>(3,)</code>
	<code>arr[2, :]</code>	<code>(3,)</code>
	<code>arr[2:, :]</code>	<code>(1, 3)</code>

	<code>arr[:, :2]</code>	<code>(3, 2)</code>
	<code>arr[1, :2]</code>	<code>(2,)</code>
	<code>arr[1:2, :2]</code>	<code>(1, 2)</code>



布尔型索引

```
names = np.array(['Bob', 'Joe', 'Will',  
                  'Bob', 'Will', 'Joe', 'Joe'])  
data = np.random.randn(7, 4)
```

```
array([[ 0.0009,  1.3438, -0.7135, -0.8312],  
       [-2.3702, -1.8608, -0.8608,  0.5601],  
       [-1.2659,  0.1198, -1.0635,  0.3329],  
       [-2.3594, -0.1995, -1.542 , -0.9707],  
       [-1.307 ,  0.2863,  0.378 , -0.7539],  
       [ 0.3313,  1.3497,  0.0699,  0.2467],  
       [-0.0119,  1.0048,  1.3272, -0.9193]])
```

data[names == 'Bob']

```
array([[ 0.0009,  1.3438, -0.7135, -0.8312],  
       [-2.3594, -0.1995, -1.542 , -0.9707]])
```

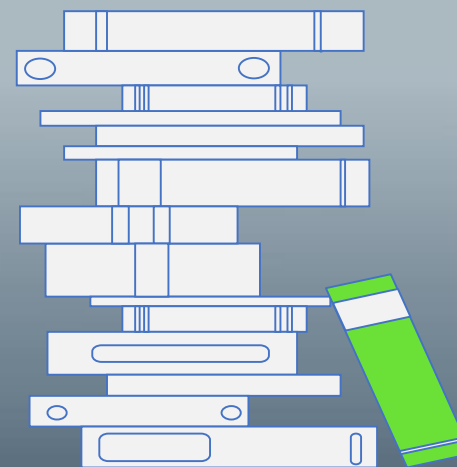
names == 'Bob'

Bob	True
Joe	False
Will	False
Bob	True
Will	False
Joe	False
Joe	False

data[names == 'Bob']



广播(Broadcasting)





广播的定义

Array Broadcasting is a mechanism used by NumPy to permit vectorized mathematical operations between arrays of **unequal, but compatible shapes**. Specifically, an array will be **treated** as if its contents have been replicated along the appropriate dimensions, such that the shape of this new, higher-dimensional array suits the mathematical operation being performed.

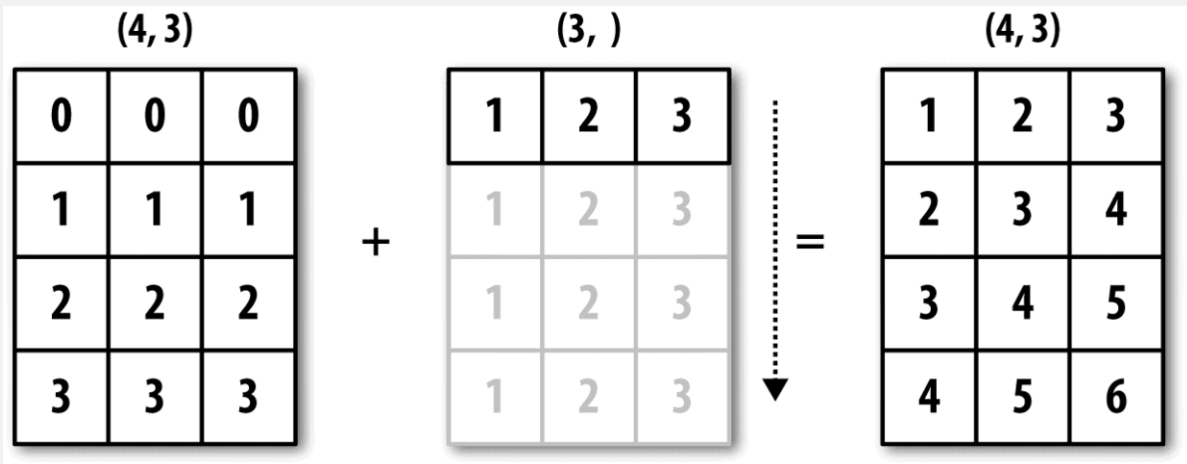
NumPy中，数组的广播机制允许矢量化运算被应用到原本形状不同、但是兼容的数组上。具体来说，一个数组可以被认为在合适的维度上重复其内容以创建一个新的数组，使得这个新的数组适用于矢量运算。

$$\begin{pmatrix} -0.0 & -0.1 & -0.2 & -0.3 \\ -0.4 & -0.5 & -0.6 & -0.7 \\ -0.8 & -0.9 & -1.0 & -1.1 \end{pmatrix} \cdot (1 \ 2 \ 3 \ 4) \rightarrow \begin{pmatrix} -0.0 & -0.1 & -0.2 & -0.3 \\ -0.4 & -0.5 & -0.6 & -0.7 \\ -0.8 & -0.9 & -1.0 & -1.1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$



广播的原则

如果两个数组的后缘维度（trailing dimension，即从末尾开始算起的维度）的轴长度相符或其中一方的长度为1，则认为它们是广播兼容的。广播会在缺失和（或）长度为1的维度上进行。



```
arr = np.random.randn(4, 3)
arr

array([[ 1.5777,  0.1611, -1.5929],
       [-0.2517, -0.4474,  1.2534],
       [-0.6305,  0.7346, -1.3853],
       [ 0.6166,  0.4716,  0.1801]])
```

```
column_mean = arr.mean(axis=0)
column_mean
```

```
array([ 0.328 ,  0.23  , -0.3862])
```

```
column_mean.shape
```

```
(3,)
```

```
demeaned = arr - arr.mean(axis=0)
demeaned
```

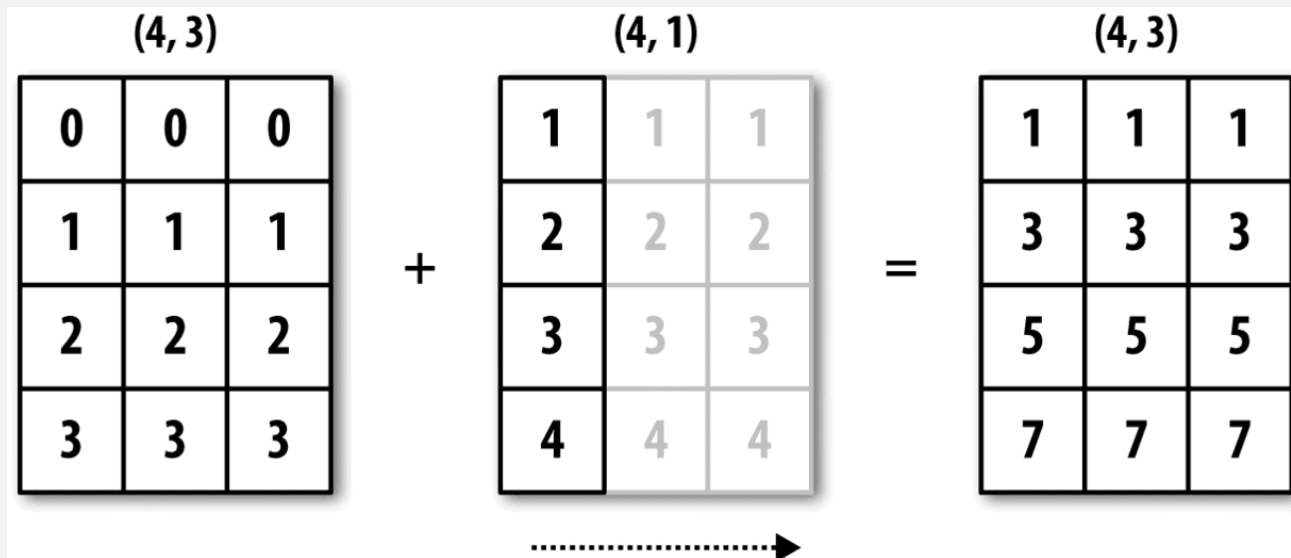
```
array([[ 1.2497, -0.0689, -1.2067],
       [-0.5797, -0.6774,  1.6396],
       [-0.9585,  0.5046, -0.9991],
       [ 0.2886,  0.2416,  0.5663]])
```

```
demeaned.mean(axis=0)
```



沿其它轴向广播

如果两个数组的后缘维度（trailing dimension，即从末尾开始算起的维度）的轴长度相符或其中一方的长度为1，则认为它们是广播兼容的。广播会在缺失和（或）长度为1的维度上进行。



```
arr
```

```
array([[ 1.5777,  0.1611, -1.5929],  
       [-0.2517, -0.4474,  1.2534],  
       [-0.6305,  0.7346, -1.3853],  
       [ 0.6166,  0.4716,  0.1801]])
```

```
row_mean = arr.mean(axis=1)  
row_mean.shape
```

```
(4,)
```

```
row_mean.reshape((4, 1))
```

```
array([[ 0.0486],  
       [ 0.1847],  
       [-0.4271],  
       [ 0.4227]])
```

```
demeaned = arr - row_mean.reshape((4, 1))  
demeaned
```

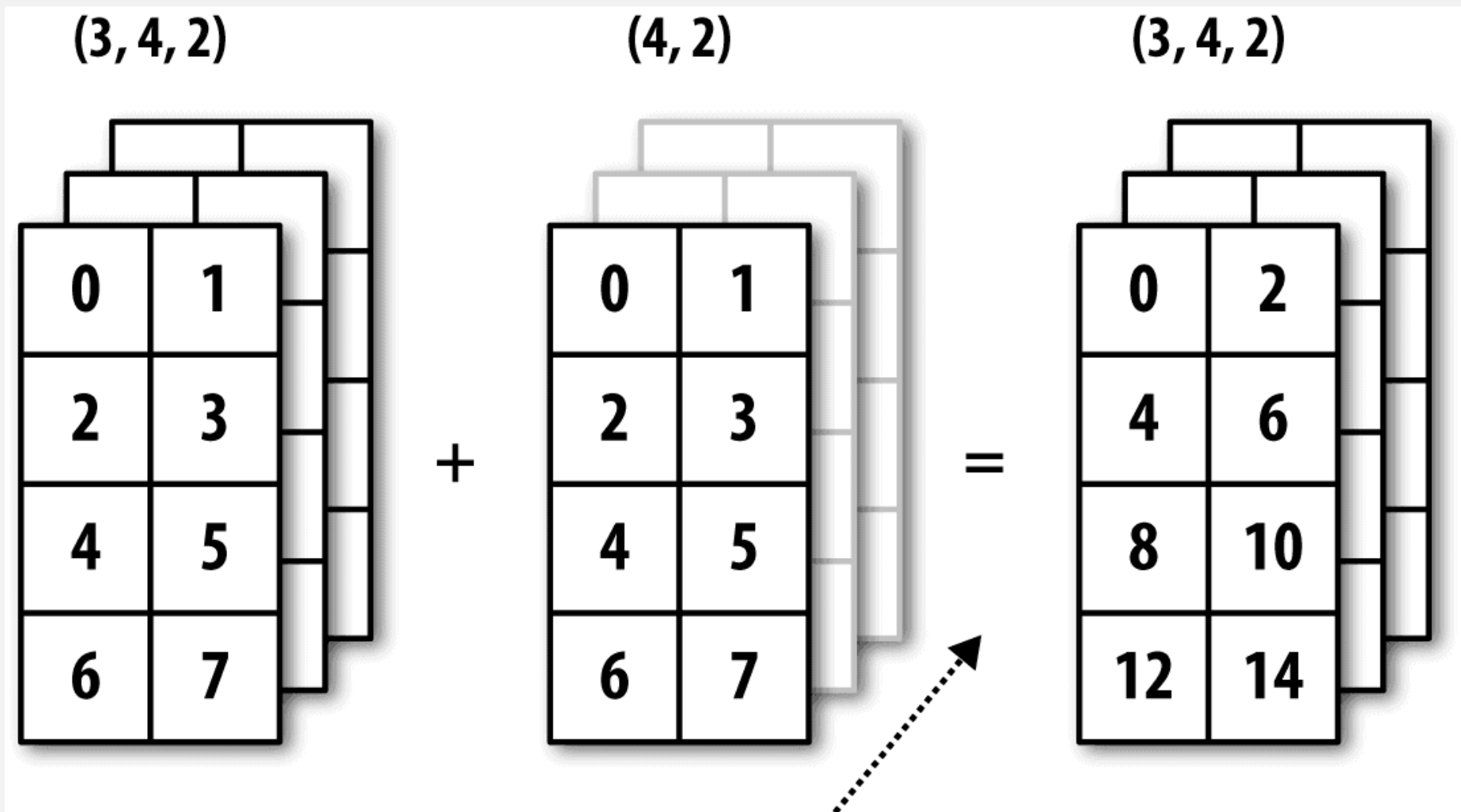
```
array([[ 1.5291,  0.1125, -1.6415],  
       [-0.4365, -0.6321,  1.0686],  
       [-0.2034,  1.1617, -0.9582],  
       [ 0.1938,  0.0488, -0.2427]])
```

```
demeaned.mean(axis=1)
```

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



更高维度的广播





通过索引机制插入轴

```
arr_1d = np.random.normal(size=3)
arr_1d
```

```
array([-0.7426,  0.8544,  1.0839])
```

```
arr_1d[:, np.newaxis].shape
```

```
(3, 1)
```

```
arr_1d[:, np.newaxis]
```

```
array([[ -0.7426],
       [  0.8544],
       [  1.0839]])
```

```
arr_1d[np.newaxis, :].shape
```

```
(1, 3)
```

```
arr_1d[np.newaxis, :]
```

```
array([[ -0.7426,  0.8544,  1.0839]])
```

```
arr = np.zeros((4, 4))
arr
```

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

```
arr_3d = arr[:, np.newaxis, :]
arr_3d
```

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

```
arr_3d.shape
```

```
(4, 1, 4)
```

```
arr = np.random.randn(3, 4, 5)
arr
```

```
array([[[[-1.3181,  1.8289,  1.6739,  0.8665,  0.6979],
        [-0.3272,  2.11   ,  0.3175, -0.6971,  0.7475],
        [-1.385   ,  1.1389,  0.2016,  0.6744,  1.6366],
        [ 1.3743,  0.337   ,  0.4853, -0.7615, -0.4414]],

       [[ 1.1291, -0.4877,  0.51   ,  1.282   , -0.7154],
        [-0.5768, -0.734   ,  2.0851, -1.2594,  0.2649],
        [-2.1169,  0.3475,  0.3156,  2.8779,  1.0385],
        [-0.992   ,  1.7024,  0.2332, -0.7327,  0.4347]],

       [[-1.2868,  1.3091, -1.5402,  0.0593, -0.6716],
        [-0.1131, -0.258   ,  1.1906,  0.8892, -1.4934],
        [ 0.5092, -0.4886,  0.2226,  1.0761,  0.709   ],
        [-0.4053, -0.5407, -1.8353,  2.3576,  0.4572]]])
```

```
depth_means = arr.mean(axis=2)
depth_means
```

```
array([[ 0.7498,  0.4301,  0.4533,  0.1987],
       [ 0.3436, -0.044   ,  0.4925,  0.1291],
       [-0.4261,  0.0431,  0.4057,  0.0067]])
```

```
depth_means.shape
```

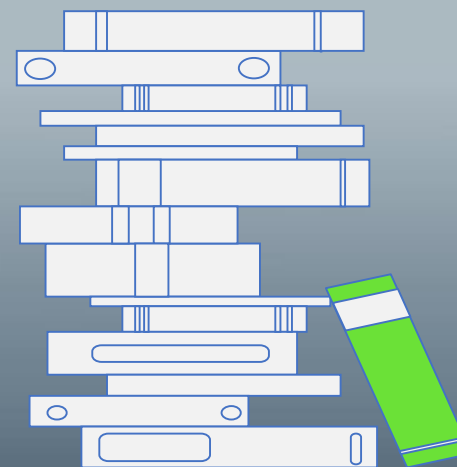
```
(3, 4)
```

```
demeaned = arr - depth_means[:, :, np.newaxis]
demeaned.mean(2)
```

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




花式索引 (Fancy Indexing)





利用整数数组进行索引

```
arr = np.empty((8, 4))
for i in range(8):
    arr[i] = i
arr
```

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

```
arr[[4, 3, 0, 6]]
```

```
array([[4., 4., 4., 4.],
       [3., 3., 3., 3.],
       [0., 0., 0., 0.],
       [6., 6., 6., 6.]])
```

```
arr[[-3, -5, -7]]
```

```
array([[5., 5., 5., 5.],
       [3., 3., 3., 3.],
       [1., 1., 1., 1.]])
```



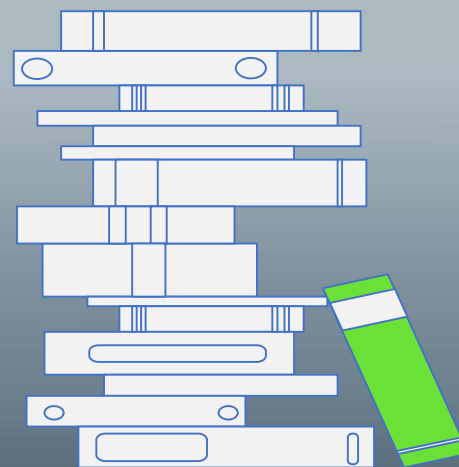
利用整数数组进行索引

```
arr = np.arange(32).reshape((8, 4))  
arr  
  
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]])
```

```
arr[[1, 5, 7, 2], [0, 3, 1, 2]]  
  
array([ 4, 23, 29, 10])  
  
arr[[1, 5, 7, 2]][:, [0, 3, 1, 2]]  
  
array([[ 4,  7,  5,  6],  
       [20, 23, 21, 22],  
       [28, 31, 29, 30],  
       [ 8, 11,  9, 10]])
```



条件逻辑





将条件逻辑表述为数组运算

True	1.1	2.1
False	1.2	2.2
True	1.3	2.3
True	1.4	2.4
False	1.5	2.5

```
xarr = np.array([1.1, 1.2, 1.3, 1.4, 1.5])  
yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])  
cond = np.array([True, False, True, True, False])
```

```
result = [(x if c else y)  
          for x, y, c in zip(xarr, yarr, cond)]  
result
```

```
[1.1, 2.2, 1.3, 1.4, 2.5]
```

```
result = np.where(cond, xarr, yarr)  
result
```

```
array([1.1, 2.2, 1.3, 1.4, 2.5])
```