

# 【Python】numpy——数组array

```
import numpy as np
```

Numpy 的主要对象是同质的多维数组array

为了可视化切片过程，我们把二维数组的垂直方向定义为 axis 0 轴，水平方向为 axis 1 轴。

		axis 1		
		0	1	2
axis 0	0	0.0	1.0	2.0
	1	3.0	4.0	5.0
	2	6.0	7.0	8.0

## 一、创建数组

创建Numpy数组一般有三种方法：

- (1) 通过传入可迭代对象创建，我将之称为基本方法np.array()
- (2) 使用Numpy内部功能函数，内部方法
- (3) 使用特殊的 库函数 ，特殊方法

### (1)np.array()

```
>>> import numpy as np
>>> np.array([1, 2, 3, 4, 5])          # 把列表转换为数组
array([1, 2, 3, 4, 5])
>>> np.array((1, 2, 3, 4, 5))         # 把元组转换成数组
array([1, 2, 3, 4, 5])
>>> np.array(range(5))                # 把range对象转换成数组
array([0, 1, 2, 3, 4])
>>> np.array([[1, 2, 3], [4, 5, 6]])  # 二维数组
array([[1, 2, 3],
       [4, 5, 6]])
```

在创建的时候，可以显式地指定数据的类型：

```
c = np.array([ [1,2], [3,4] ], dtype=complex )
c
```

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

CSDN @fftx\_00

### (2)np.ones()、np.zeros()、np.eye()、np.empty()

函数zeros创建一个都是0的数组，

函数ones创建一个都是1的数组，

函数empty创建一个初始内容是0或者垃圾值的数组，这取决于内存当时的状态。

默认情况下，创建的数组的数据类型为float64

```
1 | np.zeros(3)
2 | np.zeros( (3,4) )
3 | np.ones( (2,3,4), dtype=np.int16 )
```

```

4 | 5 |
6 | np.empty( (2,3) ) # 根据当前内存状态的不同，可能会返回未初始化的垃圾数值，不安全。
7 | np.full((3,4), 2.2) # 创建一个全部由2.2组成的数组

```

### (3)特殊方法:

#### 1.numpy.arange()

numpy.arange(start, stop, step, dtype)

start: 范围的起始值，默认为0  
stop: 范围的终止值（不包含）  
step: 两个值的间隔，默认为1  
dtype: 返回ndarray的数据类型，如果没有提供，则会使用输入数据的类型。

```

>>> np.arange(8) # 类似于内置函数range()
array([0, 1, 2, 3, 4, 5, 6, 7])
>>> np.arange(1, 10, 2)
array([1, 3, 5, 7, 9])

```

CSDN @fftx\_00

#### 2.numpy.linspace()

numpy.linspace(start, stop, num, endpoint, retstep, dtype)

start: 序列的起始值  
stop: 序列的终止值，如果 endpoint 为 True，则终止值包含于序列中  
num: 要生成的等间隔样例数量，默认为 50  
endpoint: 序列中是否包含stop值，默认为 True  
retstep: 如果为 True，返回样例以及连续数字之间的步长  
dtype: 输出 ndarray 的数据类型

CSDN @fftx\_00

注意

1.endpoint默认为True:

**np.linspace()默认是包含终点的 [start,end]**

2.dtype

如果未给出数据类型，则从开始和停止推断数据类型。

**推断出的数据类型永远不会是整数**；即使参数将生成整数数组，也会选择float。

```

>>> np.linspace(0, 10, 11) # 等差数组，包含11个数
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
>>> np.linspace(0, 10, 11, endpoint=False) # 不包含终点
array([ 0.,  0.90909091, 1.81818182, 2.72727273, 3.63636364,
        4.54545455, 5.45454545, 6.36363636, 7.27272727, 8.18181818,
        9.09090909])

```

CSDN

#### (3)np.logspace()

numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)

参数设置类似于np.linspace()

1.endpoint默认为True:

**np.logspace()默认是包含终点的 [start,end]**

2.dtype

如果未给出数据类型，则从开始和停止推断数据类型。

**推断出的数据类型永远不会是整数**；即使参数将生成整数数组，也会选择float。

3.base默认为10:

**相当于10\*\*np.linspace()**

```
>>> np.logspace(0, 100, 10) # 相当于10**np.linspace(0,100,10)
array([1.00000000e+000, 1.29154967e+011, 1.66810054e+022,
       2.15443469e+033, 2.78255940e+044, 3.59381366e+055,
       4.64158883e+066, 5.99484250e+077, 7.74263683e+088,
       1.00000000e+100])
```

```
>>> np.logspace(1,6,5, base=2) # 相当于2 ** np.linspace(1,6,5)
array([ 2., 4.75682846, 11.3137085 , 26.90868529, 64.])
```

(4)np.identity()

```
>>> np.identity(3) # 单位矩阵, 3行3列
array([[ 1., 0., 0.],
       [ 0., 1., 0.],
       [ 0., 0., 1.]])
```

CSDN @fftx\_00

(5)np.random——.randint() .rand() .standard\_normal()

```
>>> np.random.randint(0, 50, 5) # 随机数组, 5个0到50之间的数字
array([13, 47, 31, 26, 9])
```

```
>>> np.random.randint(0, 50, (3,5)) # 3行5列, 共15个随机数, 都介于[0,50)
array([[44, 34, 35, 28, 18],
       [24, 24, 26, 4, 21],
       [30, 40, 1, 24, 17]])
```

```
>>> np.random.rand(10) # 10个介于[0,1)的随机数
array([ 0.58193552, 0.11106142, 0.13848858, 0.61148304, 0.72031
       0.12807841, 0.49999167, 0.24124012, 0.15236595, 0.54566
```

CSDN

```
np.random.standard_normal(5) # 从标准正态分布中随机采样5个数字
```

```
array([-1.04281101, -1.47809369, -0.95554033, -0.91126987, 0.05362179])
```

```
x = np.random.standard_normal(size=(3, 4, 2))
x
```

```
array([[[[-0.49556201, 0.72253654],
         [ 0.93004285, 0.43248735],
         [-1.2498637 , 0.41603869],
         [ 0.79871704, 0.27869685]],
```

```
[[[ 1.16892215, -0.87627052],
    [-0.1288569 , -0.8376619 ],
    [-0.54616519, 0.9932925],
    [-0.62385535, -1.12536919]],
```

```
[[[ 1.65261525, 0.16940175],
    [-0.96708337, 0.71569427],
    [ 0.22475224, 1.72108558],
    [-1.02616823, 1.76735533]])])
```

CSDN @fftx\_00

(6)np.diag()

```
np.diag([1,2,3,4]) # 对角矩阵
```

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

CSDN @fftx\_00

## 二、测试两数组是否相同(相近)——np.isclose(),np.allclose()

isclose(),allclose() 函数来测试两个数组中对应位置上的元素在允许的范围内是否相等。并可以接收绝对误差参数和相对误差参数。

- allclose():返回单个True或False

- `isclose()`:返回若干True/False的列表

```
import numpy as np
x = np.array([1, 2, 3, 4.001, 5])
y = np.array([1, 1.999, 3, 4.01, 5.1])
```

```
print(np.isclose(x, y))
```

```
[ True False  True False False]
```

```
print(np.allclose(x, y, rtol=0.2)) # 设置相对误差参数
print(np.allclose(x, y, atol=0.2)) # 设置绝对误差参数
```

```
True
True
```

```
print(np.isclose(x, y))
print(np.isclose(x, y, atol=0.2))
```

```
[ True False  True False False]
[ True  True  True  True  True]
```

CSDN @fftx\_00

## 三、修改数组

### 1.numpy.append()

返回新数组，不影响原来的数组

注意一维数组和多维数组追加的区别：

n维数组要保持维度，只能追加n维数组，不然报错；

多维数组 追加一维数组后，会变成一维数组（axis=None,先展平）

```
>>> x = np.arange(8)
```

```
>>> x
```

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

```
>>> np.append(x, 8)
```

# 返回新数组，在尾部追加一个元素

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

```
>>> np.append(x, [9,10])
```

# 返回新数组，在尾部追加多个元素

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

```
>>> x
```

# 不影响原来的数组

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

CSDN @fftx\_00

```
a = np.array([[1,2,3],[4,5,6]])
np.append(a, [7,8,9]) # 附加后，变成了一维的
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])CSDN @fftx_00
```

ndarray没有这个方法，而是numpy下的：

```
a.append([10,11,12]) # ndarray没有这个方法
```

多维数组的行追加、列追加和多维追加（axis=0,axis=1……）：

```
np.append(a, [[7,8,9]],axis = 0) # 注意参数格式
```

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

```
np.append(a, [[7],[8],[9]],axis = 1) # 注意参数格式
```

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

CSDN @fftx\_00

## 2.np.insert()

返回新数组，不影响原来的数组

注意一维数组和多维数组插入的区别：

axis=None时，会先将数组展平

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

```
>>> a
```

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

```
>>> np.insert(a, 1, 5)
```

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

```
>>> np.insert(a, 1, 5, axis=1)
```

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

CSDN @fftx\_00

## 3.切片

```
>>> x
```

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

```
>>> x[3] = 8
```

```
>>> x
```

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

# 使用下标的形式原地修改元素值  
# 原来的数组被修改了

CSDN @fftx\_00

```

>>> x = np.array([[1,2,3], [4,5,6], [7,8,9]])
>>> x[0, 2] = 4 # 修改第0行第2列的元素值
>>> x[1:, 1:] = 1 # 切片, 把行下标大于等于1,
# 且列下标也大于等于1的元素值都

>>> x
array([[1, 2, 4],
       [4, 1, 1],
       [7, 1, 1]])
>>> x[1:, 1:] = [1,2] # 同时修改多个元素值
>>> x
array([[1, 2, 4],
       [4, 1, 2],
       [7, 1, 2]])
>>> x[1:, 1:] = [[1,2],[3,4]] # 同时修改多个元素值
>>> x
array([[1, 2, 4],
       [4, 1, 2],
       [7, 3, 4]])

```

CSDN

## 四、数组的运算

### 1.数组与标量

```

>>> x = np.array((1, 2, 3, 4, 5)) # 创建数组对象
>>> x
array([1, 2, 3, 4, 5])
>>> x * 2 # 数组与数值相乘, 返回新数组
array([ 2, 4, 6, 8, 10])
>>> x / 2 # 数组与数值相除
array([ 0.5, 1. , 1.5, 2. , 2.5])
>>> x // 2 # 数组与数值整除
array([0, 1, 1, 2, 2], dtype=int32)
>>> x ** 3 # 幂运算
array([1, 8, 27, 64, 125], dtype=int32)
>>> x + 2 # 数组与数值相加
array([3, 4, 5, 6, 7])
>>> x % 3 # 余数
array([1, 2, 0, 1, 2], dtype=int32)
>>> 2 ** x # 分别计算2**1、2**2、2**3、2**4、2**5
array([2, 4, 8, 16, 32], dtype=int32)
>>> 2 / x
array([2. , 1. , 0.66666667, 0.5, 0.4])
>>> 63 // x
array([63, 31, 21, 15, 12], dtype=int32)

```

CSDN @fftx\_00

### 2.数组与数组

等长数组: 对应元素相加  
不等长数组: 广播

```
>>> np.array([1, 2, 3, 4]) + np.array([4, 3, 2, 1])
# 等长数组相加，对应元素相加，返回信
array([5, 5, 5, 5])
>>> np.array([1, 2, 3, 4]) + np.array([4])
# 数组中每个元素加4，数组不等长，广播
array([5, 6, 7, 8])
>>> a = np.array((1, 2, 3))
>>> a + a
# 等长数组之间的加法运算，对应元素相
array([2, 4, 6])
>>> a * a
# 等长数组之间的乘法运算，对应元素相
array([1, 4, 9])
>>> a - a
# 等长数组之间的减法运算，对应元素相
array([0, 0, 0])
>>> a / a
# 等长数组之间的除法运算，对应元素相
array([ 1.,  1.,  1.])
>>> a ** a
# 等长数组之间的幂运算，对应元素乘方
array([ 1,  4, 27], dtype=int32)
```

CSDN

```
>>> a = np.array((1, 2, 3))
>>> b = np.array([1, 2, 3], [4, 5, 6], [7, 8, 9])
>>> c = a * b
# 不同维度的数组与数组相乘，广播
>>> c
# a中的每个元素乘以b中对应列的元
# a中下标0的元素乘以b中列下标0的
# a中下标1的元素乘以b中列下标1的
# a中下标2的元素乘以b中列下标2的
```

```
array([[ 1,  4,  9],
       [ 4, 10, 18],
       [ 7, 16, 27]])
>>> a + b
# a中每个元素加b中的对应列元素
array([[ 2,  4,  6],
       [ 5,  7,  9],
       [ 8, 10, 12]])
```

CSDN

### 3.数组的内积

```
import numpy as np

x = np.array((1, 2, 3))
y = np.array((4, 5, 6))
print(np.dot(x, y))      # 输出结果都是32
print(x.dot(y))
print(sum(x*y))
```

$$x \cdot y = \sum_{i=1}^n x_i y_i$$

CSDN@fftx\_00

### 4.数学函数

```
1 | x = np.array([1, 2, 3], [4, 5, 6], [7, 8, 9])
2 | print(x)
```

3   <code>print(np.sin(x))</code>	# 一维数组中所有元素求正弦值	4   <code>print(np.cos(x))</code>	# 二维数组中所有元素求余弦值
5   <code>print(np.round(np.cos(x)))</code>	# 四舍五入		
6   <code>print(np.ceil(x/2))</code>	# 向上取整		

## 5.布尔运算 > < == &| np.all() np.any() np.sum()

```
>>> import numpy as np
>>> x = np.random.rand(10) # 包含10个随机数的数组
>>> x
array([0.56707504, 0.07527513, 0.0149213, 0.49157657, 0.75404095,
       0.40330683, 0.90158037, 0.36465894, 0.37620859, 0.62250594])
>>> x > 0.5 # 比较数组中每个元素值是否大于0.5
array([ True, False, False, False,  True, False,  True, False, False,  True]
      dtype=bool)
>>> x[x>0.5] # 获取数组中大于0.5的元素
array([ 0.56707504,  0.75404095,  0.90158037,  0.62250594])
>>> x < 0.5 # 数组元素每个元素是否小于0.5
array([False,  True,  True,  True, False,  True, False,  True,  True, False]
      dtype=bool)
>>> sum((x>0.4) & (x<0.6)) # 值大于0.4且小于0.6的元素数量, True表示1, False表示0
3
>>> np.all(x<1) # 测试是否全部元素都小于1
True
>>> np.any(x>0.8) # 是否存在大于0.8的元素
True
```

CSDN

## 五、数组排序

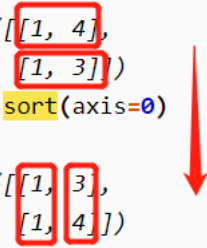
### 1.ndarray.sort(),numpy.sort()

```
ndarray.sort(axis=-1, kind=None, order=None)
```

```
numpy.sort(a, axis=-1, kind=None, order=None)
```

axis=0: 列的元素改变位置, 垂直方向计算  
axis=1: 行的元素改变位置, 水平方向计算

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



CSDN @fftx\_00



使用order关键字指定在对结构化数组进行排序时要使用的字段:

```
>>> a = np.array([('a', 2), ('c', 1)], dtype=[('x', 'S1'), ('y', int)])
>>> a.sort(order='y')
>>> a
array([(b'c', 1), (b'a', 2)],
      dtype=[('x', 'S1'), ('y', '<i8')])
```

CSDN @fftx\_00

## 2. np.argsort(), np.argmax(), np.argmin()

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

# 返回排序后元素的原下标  
# 原数组中下标1的元素最小  
# 下标2的元素次之  
# 下标0的元素最大  
# 使用数组做下标, 获取对应位置的元素

CSDN

```
>>> x[_]
array([1, 2, 3])
```

# \_表示上一条语句

```
>>> x = np.array([3, 1, 2, 4])
>>> x.argmax(), x.argmin()
(3, 1)
```

# 最大值和最小值的下标

CSDN @fftx\_00

## 六、赋值、深复制、浅复制(视图)

赋值: 对同一对象新建了一个引用

浅复制: 共享数据, 保存在base数组中 (改变数据会互相影响)

深复制: 新建完全不同的两个对象, 除了初始数据相同

### (1) 赋值

当对numpy数组进行赋值时，只是对同一个对象新建了一个引用，并不是建立新的对象，所以赋值前后的变量完全是同一对象，对其中一个引用修改时，所有引用都

```
a = np.arange(12) b = a # 赋值 a is b
```

```
b.shape = (3, 4) b a
```

### (2) 视图 (切片、浅复制)

numpy中允许不同数组间共享数据，这种机制在numpy中称为视图，对numpy数组的切片和浅复制都是通过视图实现的。如果数组B是数组A的视图(view)，则称A为B非A也是视图)。视图数组中的数据实际上保存在base数组中。

```
a = np.arange(12)
b = a.view() # 使用视图
a is b
```

```
b
```

```
b.shape = (3, 4) # 改变b的形状
a
```

```
b
```

```
b[0] = 0
a
```

```
b
```

从上面代码中我们可以发现，a和b是不同的两个数组，改变b的形状对a不会有影响，但是改变b的数据之后，a的数据也会发现改变，说明a与b是共享数据的。 CSDN

再来探索一些切片：

```
a = np.arange(12)
b = a[:] # 切片
a is b
```

```
b.shape = (3, 4)
a
```

```
b
```

```
b[0] = 0
a
```

```
b
```

果然，切片效果与视图一致。

### (3) 深复制

深复制通过数组自带的copy()方法实现，深复制产生的数组与原数组时完全不同的两个数组对象，完全享有独立的内存空间，所有操作都不会

```
a = np.arange(12)
b = a.copy()
a is b
```

```
b.shape = (3, 4)
b[0] = 0
a
```

```
b
```

CSDN

numpy中的array只有赋值受到改变形状的影响；  
赋值、切片、视图均受到改变值的影响

```

>>> import numpy as np
>>> a = np.array(range(12))
>>> b = a
>>> c = a.view()
>>> d = a[:]
>>> a[5]=233
>>> a
array([ 0,  1,  2,  3,  4, 233,  6,  7,  8,  9, 10, 11])
>>> b
array([ 0,  1,  2,  3,  4, 233,  6,  7,  8,  9, 10, 11])
>>> c
array([ 0,  1,  2,  3,  4, 233,  6,  7,  8,  9, 10, 11])
>>> d
array([ 0,  1,  2,  3,  4, 233,  6,  7,  8,  9, 10, 11])
>>>
>>>
>>> a.shape = (3, 4)
>>> a
array([[ 0,  1,  2,  3],
       [ 4, 233,  6,  7],
       [ 8,  9, 10, 11]])
>>> b
array([[ 0,  1,  2,  3],
       [ 4, 233,  6,  7],
       [ 8,  9, 10, 11]])
>>> c
array([ 0,  1,  2,  3,  4, 233,  6,  7,  8,  9, 10, 11])
>>> d
array([ 0,  1,  2,  3,  4, 233,  6,  7,  8,  9, 10, 11])

```

在python标准库中,

对list对象——

赋值相当于**新建引用**(始终为同一变量),

```

>>> a = [1, 2, 3]
>>> b = a
>>> a[1]=1
>>> a is b
True
>>> a
[1, 1, 3]
>>> b
[1, 1, 3]

```

切片相当于浅拷贝(只拷贝外层元素, **内部嵌套元素通过引用**, 而不是独立分配内存)

```

>>> a = [1, 2, 3, 4, 5]
>>> b = a[:]
>>> a is b
False

```

但**切片不会受到改变值的影响**

```

>>> a = [1, 2, 3, 4, 5]
>>> b = a[:]
>>> id(a[2]) == id(b[2])
True
>>> a[2]=233
>>> id(a[2]) == id(b[2])
False
>>> a
[1, 2, 233, 4, 5]
>>> b
[1, 2, 3, 4, 5]

```

跟 int对象一样

```

>>> a = 1
>>> b = a
>>> a is b
True
>>> a = 2
>>> a is b
False
>>> a
2
>>> b
1

```

## 七、访问数组

### 1.下标、切片

```
>>> import numpy as np
>>> b = np.array([1,2,3],[4,5,6],[7,8,9])
>>> b
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> b[0] # 第0行所有元素
array([1, 2, 3])
>>> b[0][0] # 第0行第0列的元素
1
>>> b[0,2] # 第0行第2列的元素，等价于b[0][2]的形式
3
>>> b[[0,1]] # 第0行和第1行的所有元素，不指定列下标，表示
array([[1, 2, 3],
       [4, 5, 6]])
>>> b[[0,2,1],[2,1,0]] # 第0行第2列、第2行第1列、第1行第0列的元素
# 第一个列表表示行下标，第二个列表表示列下标
array([3, 8, 4])
```

CSDN

```
1 | a[x1][y1]
2 | a[x1,y1] # 输出a[x1][y1]
3 | a[[x1,x2,x3],[y1,y2,y3]] # 输出a[x1][y1],a[x2][y2],a[x3][y3]
4 | a[x1:x3,y1:y3] # 输出a[x1][y1:y3],a[x2][y1:y3]
5 | # 输出a[x1][y1],a[x1][y2],a[x2][y1],a[x2][y2]
6 |
7 | a[x1] # 输出a[x1][:]
8 | a[[x1,x2,x3]] # 输出a[x1][:],a[x2][:],a[x3][:]
9 | a[:,[y1,y2,y3]] # 输出a[:,y1],a[:,y2],a[:,y3]
```

```
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[::-1] # 反向切片
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
>>> a[::2] # 隔一个取一个元素
array([0, 2, 4, 6, 8])
>>> a[:5] # 前5个元素
array([0, 1, 2, 3, 4])
```

CSDN @fftx\_00

```
>>> c[:, [2,4]] # 第2列和第4列所有元素，对行下标进行切片，冒号表示所有行
array([[ 2,  4],
       [ 7,  9],
       [12, 14],
       [17, 19],
       [22, 24]])
>>> c[[1,3]][:, [2,4]] # 第1、3行的2、4列元素
array([[ 7,  9],
       [17, 19]])
```

CSDN @fftx\_00

CSDN @fftx\_01

```
>>> a = np.array(range(24))
>>> a.shape=(4,6)
>>> a
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]])
>>> a[[1,3]][:, [2,4]]
array([[ 8, 10],
       [20, 22]])
```

CSDN @fftx\_00

## 八、改变形状 x.shape x.reshape() x.resize()

### 1. x.shape x.size

x.shape: 返回形状的元组  
x.size: 元素总个数

```
>>> import numpy as np
>>> x = np.arange(1, 11, 1)
>>> x
array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>> x.shape          # 查看数组的形状
(10,)
>>> x.size           # 数组中元素的数量
10
>>> x.shape = 2, 5   # 改为2行5列
>>> x
array([[1, 2, 3, 4, 5],
       [6, 7, 8, 9, 10]])
>>> x.shape
(2, 5)
>>> x.shape = 5, -1  # -1表示自动计算
>>> x
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8],
       [9, 10]])
```

CSDN @fftx\_00

### 2. x.reshape()

```
>>> x = x.reshape(2,5)          # reshape()方法返回新数组
>>> x
array([[1, 2, 3, 4, 5],
       [6, 7, 8, 9, 10]])
```

CSDN @fftx\_00

### 3. x.resize() np.resize()

```
>>> import numpy as np
>>> a = np.array(range(30))
>>> a.resize((1, 10))
>>> a
array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

注意: array不能引用或被引用

ValueError: cannot resize an array that references or is referenced ...

np.resize()则没有这个要求

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

## 九、查找、过滤数组元素 np.where() np.piecewise()

```
>>> a
array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
>>> np.where(a>5)
(array([0, 0, 0, 0], dtype=int64), array([6, 7, 8, 9], dtype=int64))
>>> np.where(a>5, 0, 1)
array([[1, 1, 1, 1, 1, 1, 0, 0, 0, 0]])
```

CSDN @fftx\_00

```

>>> import numpy as np
>>> x = np.random.randint(0, 10, size=(1,10))
>>> x
array([[0, 4, 3, 3, 8, 4, 7, 3, 1, 7]])
>>> np.where(x<5, 0, 1)          # 小于5的元素值对应0, 其他对应1
array([[0, 0, 0, 0, 1, 0, 1, 0, 0, 1]])
>>> x.resize((2, 5))
>>> x
array([[0, 4, 3, 3, 8],
       [4, 7, 3, 1, 7]])
>>> np.piecewise(x, [x<4, x>7], [lambda x:x*2, lambda x:x*3])
                                # 小于4的元素乘以2, 大于7的元素乘以3, 其他元素变为0
array([[ 0,  0,  6,  6, 24],
       [ 0,  0,  6,  2,  0]])
>>> np.piecewise(x, [x<3, (3<x)&(x<5), x>7], [-1, 1, lambda x:x*4])
                                # 小于3的元素变为-1, 大于3小于5的元素变为1, 大于7的元
                                # 条件没有覆盖到的其他元素变为0
array([[ -1,  1,  0,  0, 32],
       [ 1,  0,  0, -1,  0]])

```