NumPy

`numpy`库是计算速度非常快的一个数学库，用于对n维数组进行数学计算。

一、基本的数据类型：

名称	描述
`bool_`	布尔型数据类型（`True` 或者 `False`）
`int_`	默认的整数类型（类似于C 语言中的 `long`，`int32` 或 `int64`）
`intc`	与 C 的 `int` 类型一样，一般是 `int32` 或 `int 64`
`intp`	用于索引的整数类型（类似于 C 的 `ssize_t`，一般情况下仍然是 `int32` 或 `int64`）
`int8`	字节（`-128 ~ 127`）
`int16`	整数（`-32768 ~ 32767`）
`int32`	整数（`-2147483648 ~ 2147483647`）
`int64`	整数（`-9223372036854775808 ~ 9223372036854775807`）
`uint8`	无符号整数（`0 ~ 255`）
`uint16`	无符号整数（`0 ~ 65535`）
`uint32`	无符号整数（`0 ~ 4294967295`）
`uint64`	无符号整数（`0 ~ 18446744073709551615`）
`float_`	`float64` 类型的简写
`float16`	半精度浮点数，包括：`1` 个符号位，`5` 个指数位，`10` 个尾数位
`float32`	单精度浮点数，包括：`1` 个符号位，`8` 个指数位，`23` 个尾数位
`float64`	双精度浮点数，包括：`1` 个符号位，`11` 个指数位，`52` 个尾数位
`complex_`	`complex128` 类型的简写，即 `128` 位复数
`complex64`	复数，表示双 `32` 位浮点数（实数部分和虚数部分）
`complex128`	复数，表示双 `64` 位浮点数（实数部分和虚数部分）

`int8`，`int16`，`int32`，`int64` 四种数据类型可以使用字符串 `i1`，`i2`，`i4`，`i8` 代替

字节顺序是通过对数据类型预先设定`<`或`>`来决定的。`<`意味着小端法(最小值存储在最小的地址，即低位组放在最前面)。`>`意味着大端法(最重要的字节存储在最小的地址，即高位组放在最前面)。

```
小端法：
地址     内存值
0x00:  4E        # 最低有效字节
0x01:  61
0x02:  BC
0x03:  00        # 最高有效字节
```

```
import numpy as np
# 字节顺序标注
dt = np.dtype('<i4')
print(dt)    # 输出：int32
```

结构化类型：

可以指定某一列的数据类型，第一个参数表示数据类型的名称，第二个参数表示具体的数据类型，这样做的好处是可以将不同的数据类型联合到一个数组中，可以处理更为复杂的数据。

```
import numpy as np
student = np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
a = np.array([('abc', 21, 50),('xyz', 18, 75)], dtype = student)
print(a)  # a可以直接当作数据类型进行使用
'''
输出：
[('abc', 21, 50.0), ('xyz', 18, 75.0)]
'''
```

二、数组的属性

属性	说明
`ndarray.ndim`	秩，即轴的数量或维度的数量
`ndarray.shape`	数组的维度，对于矩阵，n 行 m 列
`ndarray.size`	数组元素的总个数，相当于 .shape 中 n*m 的值
`ndarray.dtype`	ndarray 对象的元素类型
`ndarray.itemsize`	ndarray 对象中每个元素的大小，以字节为单位
`ndarray.flags`	ndarray 对象的内存信息

属性	说明
`ndarray.real`	ndarray元素的实部
`ndarray.imag`	ndarray 元素的虚部
`ndarray.data`	包含实际数组元素的缓冲区，由于一般通过数组的索引获取元素，所以通常不需要使用这个属性。

三、数组的创建

`numpy.empty`

用于创建空数组

```
numpy.empty(shape, dtype = float, order = 'C')
```

参数说明：

参数	描述
`shape`	数组形状
`dtype`	数据类型，可选
`order`	有"C"和"F"两个选项,分别代表，行优先和列优先，在计算机内存中的存储元素的顺序。

`numpy.zeros`

创建元素为0的数组

```
numpy.zeros(shape, dtype = float, order = 'C')
```

参数说明：

参数	描述
`shape`	数组形状
`dtype`	数据类型，可选
`order`	'C' 用于 C 的行数组，或者 'F' 用于 FORTRAN 的列数组

numpy.ones

创建元素为1的数组

```
numpy.ones(shape, dtype = None, order = 'C')
```

参数说明:

参数	描述
shape	数组形状
dtype	数据类型，可选
order	'C' 用于 C 的行数组，或者 'F' 用于 FORTRAN 的列数组

numpy.zeros_like

创建元素为0且与其它数组大小相同的数组

```
numpy.zeros_like(a, dtype=None, order='K', subok=True, shape=None)
```

参数说明:

参数	描述
a	给定要创建相同形状的数组
dtype	创建的数组的数据类型
order	数组在内存中的存储顺序，可选值为 'C'（按行优先）或 'F'（按列优先），默认为 'K'（保留输入数组的存储顺序）
subok	是否允许返回子类，如果为 True，则返回一个子类对象，否则返回一个与 a 数组具有相同数据类型和存储顺序的数组
shape	创建的数组的形状，如果不指定，则默认为 a 数组的形状。

numpy.ones_like

创建元素为1且与其它数组大小相同的数组

```
numpy.ones_like(a, dtype=None, order='K', subok=True, shape=None)
```

参数说明:

参数	描述
`a`	给定要创建相同形状的数组
`dtype`	创建的数组的数据类型
`order`	数组在内存中的存储顺序，可选值为`'C'`（按行优先）或`'F'`（按列优先），默认为`'K'`（保留输入数组的存储顺序）
`subok`	是否允许返回子类，如果为`True`，则返回一个子类对象，否则返回一个与`a`数组具有相同数据类型和存储顺序的数组
`shape`	创建的数组的形状，如果不指定，则默认为`a`数组的形状。

四、转换数组

`numpy.asarray`

可以直接将其它类型的数据转换为数组类型

```
numpy.asarray(a, dtype = None, order = None)
```

参数说明：

参数	描述
`a`	任意形式的输入参数，可以是，列表, 列表的元组, 元组, 元组的元组, 元组的列表，多维数组
`dtype`	数据类型，可选
`order`	可选，有`'C'`和`'F'`两个选项,分别代表，行优先和列优先，在计算机内存中的存储元素的顺序。

`numpy.frombuffer`

可以将数据以流的形式存储到数组中，相当于是动态数组

```
numpy.frombuffer(buffer, dtype = float, count = -1, offset = 0)
```

注意：`buffer`是字符串的时候，`Python3`默认`str`是`Unicode`类型，所以要转成`bytestring`在原`str`前加上`b`。

参数说明：

参数	描述
`buffer`	可以是任意对象，会以流的形式读入。
`dtype`	返回数组的数据类型，可选
`count`	读取的数据数量，默认为`-1`，读取所有数据。
`offset`	读取的起始位置，默认为`0`。

`numpy.fromiter`

`numpy.fromiter` 方法从可迭代对象中建立`ndarray` 对象，返回一维数组。

```
numpy.fromiter(iterable, dtype, count=-1)
```

参数说明：

参数	描述
`iterable`	可迭代对象
`dtype`	返回数组的数据类型
`count`	读取的数据数量，默认为`-1`，读取所有数据

```
import numpy as np

# 使用 range 函数创建列表对象
list=range(5)
it=iter(list)

# 使用迭代器创建 ndarray
x=np.fromiter(it, dtype=float)
print(x)
'''
输出：
[0. 1. 2. 3. 4.]
'''
```

五、创建指定范围的数组

`numpy.arange`

*创建数值范围并**返回** `ndarray` 对象*

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

根据`start`与 `stop` 指定的范围以及 `step`设定的步长，生成一个`ndarray`。

参数说明：

参数	描述
`start`	起始值，默认为`0`
`stop`	终止值（不包含）
`step`	步长，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。

`numpy.linspace`

`numpy.linspace`函数用于创建一个**一维**数组，数组是一个等差数列构成的

```
np.linspace(start, stop, num=50, endpoint=True, retstep=False,
dtype=None)
```

参数说明：

参数	描述
`start`	序列的起始值
`stop`	序列的终止值，如果`endpoint`为`True`，该值包含于数列中
`num`	要生成的等步长的样本数量，默认为`50`
`endpoint`	该值为 `True` 时，数列中包含`stop`值，反之不包含，默认是`True`。
`retstep`	如果为 `True` 时，生成的数组中会显示间距，反之不显示。
`dtype`	`ndarray` 的数据类型

numpy.logspace

numpy.logspace 函数用于创建一个于等比数列。

```
np.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)
```

base 参数意思是取对数的时候 log 的下标。

参数说明：

参数	描述
start	序列的起始值为：$base^{start}$
stop	序列的终止值为：$base^{stop}$。如果 endpoint 为 True，该值包含于数列中
num	要生成的等步长的样本数量，默认为 50
endpoint	该值为 True 时，数列中中包含 stop 值，反之不包含，默认是 True。
base	对数 log 的底数。
dtype	ndarray 的数据类型

```
import numpy as np
# 默认底数是 10
a = np.logspace(1.0,  2.0, num =  10)
print (a)
'''
输出：
[ 10.          12.91549665     16.68100537      21.5443469
27.82559402
  35.93813664   46.41588834     59.94842503      77.42636827    100.
]
'''
```

六、切片操作

使用内置函数：slice()

```
import numpy as np

a = np.arange(10)
s = slice(2,7,2)   # 从索引 2 开始到索引 7 停止，间隔为2
print (a[s])
'''
```

```
输出：
[2  4  6]
'''
```

直接进行切片

可以直接使用冒号`:`进行切片操作，其含义为start： stop： step

```
import numpy as np

a = np.arange(10)
b = a[2:7:2]   # 从索引 2 开始到索引 7 停止，间隔为 2
print(b)
'''
输出：
[2  4  6]
'''
```

多维数组同理：

1. 如果只有一个数字`2`，表示定位到改维度的第`2`个
2. 如果是`2:`，表示定位到该维度从第`2`个开始到最后一个
3. 如果是`:2`表示从该维度的开始到第`2`个结束
4. 如果是`:`或`不写`表示该维度的所有数据。

```
import numpy as np

a = np.array([[1,2,3], [4,5,6],[7,8,9]])
b = a[1:3, 1:3]
c = a[1:3,[1,2]]
d = a[...,1:]
print(b)
print(c)
print(d)
'''
输出：
[[5 6]
 [8 9]]
[[5 6]
 [8 9]]
[[2 3]
 [5 6]
 [8 9]]

'''
```

省略号`...`，表示该维度的所有元素

```
import numpy as np

a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print (a[...,1])   # 第2列元素
print (a[1,...])   # 第2行元素
print (a[...,1:])  # 第2列及剩下的所有元素

'''
输出：
[2 4 5]
[3 4 5]
[[2 3]
 [4 5]
 [5 6]]
'''
```

七、索引

坐标索引

分别传入每一个维度指定位置的坐标，(可以结合`:`使用)，然后一 一对应进行索引。

```
import numpy as np

x = np.array([[1,  2],  [3,  4],  [5,  6]])
y = x[[0,1,2],  [0,1,0]]      # 表示组合，第一个列表中的值表示的是第一维坐标，第二个列表中的值表示的是第二维坐标，然后对应组合即可
print (y)
'''
输出：
[1  4  5]
'''
```

布尔索引

用于计算符合条件的元素

```
import numpy as np

x = np.array([[  0,  1,  2],[  3,  4,  5],[  6,  7,  8],[  9,  10,  11]])
print ('我们的数组是：')
print (x)
print ('\n')
# 现在我们会打印出大于 5 的元素
```

```
print  ('大于 5 的元素是：')
print (x[x >  5])
'''
输出：
我们的数组是：
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]


大于 5 的元素是：
[ 6  7  8  9 10 11]
'''
```

使用取反的运算符`~`来判断满足非空的元素的值

```
import numpy as np

a = np.array([np.nan,  1,2,np.nan,3,4,5])
print (a[~np.isnan(a)])    #
'''
输出：
[ 1.   2.   3.   4.   5.]
'''
```

八、广播

如果两个数组的维度不同，且在相同的维度上其个数相等，并且不同维度中维度数少的维度为`1`，则可以将所有维度为`1`的数组进行重复计算，重复次数与另一个数组的维度对齐。

如：`a = [1, 1, 1]` `b = [[1, 2, 3], [1, 2, 3]]` 那么`a + b = [[1 (+ 1), 2(+ 1), 3(+ 1)], [1(+ 1), 2(+ 1), 3(+ 1)]] = [[2, 3, 4],[2, 3, 4]]`，这样数组`a`在第`1`维度上重复了`2`次。

```
import numpy as np

a = np.array([[ 0, 0, 0],
           [10,10,10],
           [20,20,20],
           [30,30,30]])
b = np.array([0,1,2])
print(a + b)
'''
输出：
[[ 0  1  2]
```

```
 [10 11 12]
 [20 21 22]
 [30 31 32]]
'''
```

九、数组的迭代

一般迭代

可以使用`nditer`对创建的n维数组进行迭代，其中参数`order`用于控制遍历顺序，`order='C'`表示按行进行遍历，`order='F'`表示按列进行遍历。

```
import numpy as np

a = np.arange(0,60,5)
a = a.reshape(3,4)
print ('原始数组是：')
print (a)
print ('\n')
print ('原始数组的转置是：')
b = a.T
print (b)
print ('\n')
print ('以 C 风格顺序排序：')
c = b.copy(order='C')
print (c)
for x in np.nditer(c):
    print (x, end=", " )
print  ('\n')
print  ('以 F 风格顺序排序：')
c = b.copy(order='F')
print (c)
for x in np.nditer(c):
    print (x, end=", " )


'''
输出：
原始数组是：
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]


原始数组的转置是：
[[ 0 20 40]
 [ 5 25 45]
```

```
 [10 30 50]
 [15 35 55]]


以 C 风格顺序排序:
[[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]
0, 20, 40, 5, 25, 45, 10, 30, 50, 15, 35, 55,

以 F 风格顺序排序:
[[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]
0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55,
'''
```

默认情况下，遍历的时候为只读状态，如果需要在遍历的时候进行修改，需要设置参数`op_flags`为`readwrite`或者`writeonly`的模式。

```
import numpy as np

a = np.arange(0,60,5)
a = a.reshape(3,4)
print ('原始数组是: ')
print (a)
print ('\n')
for x in np.nditer(a, op_flags=['readwrite']):
    x[...]=2*x
print ('修改后的数组是: ')
print (a)

'''
输出:
原始数组是:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]


修改后的数组是:
[[  0  10  20  30]
 [ 40  50  60  70]
 [ 80  90 100 110]]
'''
```

广播迭代

如果需要将两个不同维度的数组同时进行迭代，可以使用广播迭代的方式，其原理与广播的机制相同，都是将较低维度的数组进行重复较高维度的数组的高维次。

```
import numpy as np

a = np.arange(0,60,5)
a = a.reshape(3,4)
print  ('第一个数组为：')
print (a)
print  ('\n')
print ('第二个数组为：')
b = np.array([1,  2,  3,  4], dtype =  int)
print (b)
print ('\n')
print ('修改后的数组为：')
for x,y in np.nditer([a,b]):
    print ("%d:%d"  %  (x,y), end=", " )

'''
输出：
第一个数组为：
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]


第二个数组为：
[1 2 3 4]


修改后的数组为：
0:1, 5:2, 10:3, 15:4, 20:1, 25:2, 30:3, 35:4, 40:1, 45:2, 50:3, 55:4,
'''
```

十、数组操作

修改数组形状

`numpy.reshape`

原数据不发生改变

```
numpy.reshape(arr, newshape, order='C')
```

> `arr`：要修改形状的数组

> `newshape`：整数或者整数数组，新的形状应当兼容原有形状
>
> `order`：'C'： 按行，'F'： 按列，'A'： 原顺序，'K'：元素在内存中的出现顺序。

`numpy.ndarray.flat`

`numpy.ndarray.flat` 是一个数组元素迭代器，通常情况下，直接遍历一次多维数组，遍历的是他的第一维度，如果需要得到每一个元素，需要进行多次遍历。

```
import numpy as np

a = np.arange(9).reshape(3,3)
print ('原始数组：')
for row in a:
    print (row)

#对数组中每个元素都进行处理，可以使用flat属性，该属性是一个数组元素迭代器：
print ('迭代后的数组：')
for element in a.flat:
    print (element)

'''
输出：
原始数组：
[0 1 2]
[3 4 5]
[6 7 8]
迭代后的数组：
0
1
2
3
4
5
6
7
8
'''
```

`numpy.ndarray.flatten`

`numpy.ndarray.flatten`，数组的展平处理，返回一份数组拷贝，对拷贝所做的修改不会影响原始数组。

```
ndarray.flatten(order='C')
```

> 参数说明：`order`：'C'： 按行，'F'： 按列，'A'： 原顺序，'K'： 元素在内存中的出现顺序。

`numpy.ravel`

`numpy.ravel()`展平的数组元素，顺序通常是"C风格"，返回的是数组视图（`view`，有点类似 C/C++引用`reference`的意味），修改**会影响原始数组。**

```
numpy.ravel(a, order='C')
```

> 参数说明：`order`：'C'：按行，'F'：按列，'A'：原顺序，'K'：元素在内存中的出现顺序。

翻转数组

`numpy.transpose`

`numpy.transpose` 函数用于对换数组的维度，格式如下：

```
numpy.transpose(arr, axes)
```

> 参数说明:
>
> - `arr`：要操作的数组
> - `axes`：整数列表，对应维度，通常所有维度都会对换。

`numpy.ndarray.T`类似 `numpy.transpose`

`numpy.rollaxis`

`numpy.rollaxis` 函数向后滚动特定的轴到一个特定位置，格式如下：

```
numpy.rollaxis(arr, axis, start)
```

> 参数说明:
>
> - `arr`：数组
> - `axis`：要向后滚动的轴，其它轴的相对位置不会改变
> - `start`：默认为零，表示完整的滚动。会滚动到特定位置

`numpy.swapaxes`

`numpy.swapaxes` 函数用于交换数组的两个轴，格式如下：

```
numpy.swapaxes(arr, axis1, axis2)
```

> 参数说明：
>
> - `arr`：输入的数组
> - `axis1`：对应第一个轴的整数
> - `axis2`：对应第二个轴的整数

修改数组维度

`numpy.broadcast_to`

`numpy.broadcast_to` 函数将数组广播到新形状。它在原始数组上返回只读视图。 它通常不连续。 如果新形状不符合`Numpy` 的广播规则，该函数可能会抛出`ValueError`。

```
numpy.broadcast_to(array, shape, subok)
```

```
import numpy as np

a = np.arange(4).reshape(1,4)

print ('原数组：')
print (a)
print ('\n')

print ('调用 broadcast_to 函数之后：')
print (np.broadcast_to(a,(4,4)))

'''
输出：
原数组：
[[0 1 2 3]]


调用 broadcast_to 函数之后：
[[0 1 2 3]
 [0 1 2 3]
 [0 1 2 3]
 [0 1 2 3]]
'''
```

`numpy.expand_dims`

`numpy.expand_dims` 函数通过在指定位置插入新的轴来扩展数组形状，函数格式如下:

```
 numpy.expand_dims(arr, axis)
```

> 参数说明：

> - `arr`：输入数组
> - `axis`：新轴插入的位置

```
array_20 = np.array([[1, 2], [3, 4]])
print('array_20:\n', array_20)
print('array_20.shape:',array_20.shape)
array_21 = np.expand_dims(array_20, 0)
print('array_21:\n', array_21)
print('array_21.shape:',array_21.shape)

'''
输出：
array_20:
 [[1 2]
 [3 4]]
array_20.shape: (2, 2)
array_21:
 [[[1 2]
  [3 4]]]
array_21.shape: (1, 2, 2)
'''
```

`numpy.squeeze`

`numpy.squeeze` 函数从给定数组的形状中删除一维的轴，相当于压缩。

```
numpy.squeeze(arr, axis)
```

> 参数说明：
>
> - `arr`：输入数组
> - `axis`：整数或整数元组，用于选择形状中一维条目的子集

```
array_22 = np.arange(16).reshape(1, 4, 4)
print(array_22.shape)
array_23 = array_22.squeeze()
print(array_23.shape)

'''
输出：
(1, 4, 4)
(4, 4)
'''
```

连接数组

numpy.concatenate

numpy.concatenate 函数用于沿指定轴连接相同形状的两个或多个数组

```
numpy.concatenate((a1, a2, ...), axis)
```

> 参数说明：
>
> - a1, a2, ...：相同类型的数组
> - axis：沿着它连接数组的轴，默认为 0

```
array_24 = np.arange(12).reshape(4, 3)
array_25 = np.arange(8, 20).reshape(4, 3)
print('array_24:\n', array_24)
print('array_25:\n', array_25)
array_concatenate = np.concatenate((array_24, array_25), axis = 1)  # 沿着轴1进行连接
print('array_concatenate:\n', array_concatenate)

'''
输出：
array_24:
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
array_25:
 [[ 8  9 10]
 [11 12 13]
 [14 15 16]
 [17 18 19]]
array_concatenate:
 [[ 0  1  2  8  9 10]
 [ 3  4  5 11 12 13]
 [ 6  7  8 14 15 16]
 [ 9 10 11 17 18 19]]
'''
```

numpy.stack

numpy.stack 函数用于沿新轴连接数组序列

```
numpy.stack(arrays, axis)
```

> 参数说明：

- `arrays`：**相同形状**的数组序列
- `axis`：返回数组中的轴，输入数组沿着它来**堆叠**

```
import numpy as np

a = np.array([[1,2],[3,4]])

print ('第一个数组：')
print (a)
print ('\n')
b = np.array([[5,6],[7,8]])

print ('第二个数组：')
print (b)
print ('\n')

print ('沿轴 0 堆叠两个数组：')
print (np.stack((a,b),0))
print ('\n')

print ('沿轴 1 堆叠两个数组：')
print (np.stack((a,b),1))

'''
输出：
第一个数组：
[[1 2]
 [3 4]]


第二个数组：
[[5 6]
 [7 8]]


沿轴 0 堆叠两个数组：
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]


沿轴 1 堆叠两个数组：
[[[1 2]
  [5 6]]
```

```
 [[3 4]
  [7 8]]]
'''
```

numpy.hstack

numpy.hstack 是 numpy.stack 函数的变体，它通过水平堆叠来生成数组。

numpy.vstack

numpy.vstack 是 numpy.stack 函数的变体，它通过垂直堆叠来生成数组。

分割数组

numpy.split

numpy.split 函数沿特定的轴将数组分割为子数组，格式如下：

```
numpy.split(ary, indices_or_sections, axis)
```

> 参数说明：
>
> - ary：被分割的数组
> - indices_or_sections：如果是一个整数，就用该数平均切分，如果是一个数组，为沿轴切分的位置（左开右闭）
> - axis：设置沿着哪个方向进行切分，默认为 0，横向切分，即水平方向。为 1 时，纵向切分，即竖直方向。

```
import numpy as np

a = np.arange(9)

print ('第一个数组：')
print (a)
print ('\n')

print ('将数组分为三个大小相等的子数组：')
b = np.split(a,3)
print (b)
print ('\n')

print ('将数组在一维数组中表明的位置分割：')
b = np.split(a,[4,7])
print (b)

'''
输出：
```

```
第一个数组：
[0 1 2 3 4 5 6 7 8]


将数组分为三个大小相等的子数组：
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8])]


将数组在一维数组中表明的位置分割：
[array([0, 1, 2, 3]), array([4, 5, 6]), array([7, 8])]
'''
```

```
array_26 = np.arange(12).reshape(3, 4)
print('array_26:\n', array_26)
array_sp = np.split(array_26, [1, 2])
print('array_sp:\n', array_sp)

'''
输出：
array_26:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
array_sp:
 [array([[0, 1, 2, 3]]), array([[4, 5, 6, 7]]), array([[ 8,  9, 10,
11]])]
'''
```

> 解释：
>
> 让我们逐步解释拆分的过程：
>
> 1. 第一个拆分点是在索引为1的位置，将数组分成两部分，第一部分包含了第一行，第二部分包含了剩余的两行。
> 2. 第二个拆分点是在索引为2的位置，将数组的第二部分（剩余的两行）再次拆分，第一部分包含了第二行，第二部分包含了剩余的一行。

`numpy.hsplit`

`numpy.hsplit`函数用于水平分割数组，通过指定要返回的相同形状的数组**数量**来拆分原数组。

`numpy.vsplit`

`numpy.vsplit`沿着垂直轴分割，其分割方式与`hsplit`用法相同。

数组元素的添加与删除

`numpy.resize`

`numpy.resize` 函数返回指定大小的新数组，如果新数组大小大于原始大小，在变大的部分重复添加已有的元素。

```
numpy.resize(arr, shape)
```

> 参数说明：
>
> - `arr`：要修改大小的数组
> - `shape`：返回数组的新形状

```
array_27 = np.arange(12).reshape(3, 4)
print('array_27:\n', array_27)
# 改变形状：
array_28 = np.resize(array_27, (4, 4))    # 重复了一行元素
print('array_28:\n', array_28)

'''
输出：
array_27:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
array_28:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [ 0  1  2  3]]
'''
```

`numpy.append`

`numpy.append` 函数在数组的末尾添加值。 追加操作会分配整个数组，并把原来的数组复制到新数组中。 此外，输入数组的维度必须匹配否则将生成 `ValueError`。

`append` 函数返回的始终是一个一维数组。

```
numpy.append(arr, values, axis=None)
```

> 参数说明：
>
> - `arr`：输入数组
> - `values`：要向`arr`添加的值，需要和`arr`形状相同（除了要添加的轴）

- `axis`：默认为 `None`。当`axis`无定义时，是横向加成，返回总是为一维数组！当`axis`有定义的时候，分别为`0`和`1`的时候。当`axis`有定义的时候，分别为`0`和`1`的时候（列数要相同）。当`axis`为`1`时，数组是加在右边（行数要相同）。

```
# 添加元素
array_ap = np.append(array_27, [1, 1, 1])
print('array_ap:\n', array_ap)

array_ap0 = np.append(array_27, [[1, 1, 1, 1]], axis = 0)
print('array_ap0:\n', array_ap0)
'''
输出：
array_ap:
 [ 0  1  2  3  4  5  6  7  8  9 10 11  1  1  1]
array_ap0:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [ 1  1  1  1]]
'''
```

`numpy.insert`

`numpy.insert`函数在给定索引之前，沿给定轴在输入数组中插入值。

如果值的类型转换为要插入，则它与输入数组不同。 插入没有原地的，函数会返回一个新数组。 此外，如果未提供轴，则输入数组会被展开。

```
numpy.insert(arr, obj, values, axis)
```

参数说明：

- `arr`：输入数组
- `obj`：在其之前插入值的索引
- `values`：要插入的值
- `axis`：沿着它插入的轴，如果未提供，则输入数组会被展开

```
# 插入元素
array_in = np.insert(array_27, 1, [[1, 1, 1, 1]], axis = 0)
print('array_in:\n', array_in)
'''
输出:
array_in:
 [[ 0  1  2  3]
 [ 1  1  1  1]
 [ 4  5  6  7]
 [ 8  9 10 11]]
'''
```

numpy.delete

`numpy.delete` 函数返回从输入数组中删除指定子数组的新数组。 与 `insert()` 函数的情况一样，如果未提供轴参数，则输入数组将展开。

```
Numpy.delete(arr, obj, axis)
```

> 参数说明:
>
> - `arr`：输入数组
> - `obj`：可以被切片，整数或者整数数组，表明要从输入数组删除的子数组
> - `axis`：沿着它删除给定子数组的轴，如果未提供，则输入数组会被展开

```
# 删除元素
array_del = np.delete(array_27, 1, axis = 0)  # 删除第二行
print('array_del:\n', array_del)
'''
array_del:
 [[ 0  1  2  3]
 [ 8  9 10 11]]
'''
```

numpy.unique

`numpy.unique` 函数用于去除数组中的重复元素。

```
numpy.unique(arr, return_index, return_inverse, return_counts)
```

- `arr`：输入数组，如果不是一维数组则会展开
- `return_index`：如果为`True`，返回新列表元素在旧列表中的位置（下标），并以列表形式储

- `return_inverse`：如果为`True`，返回旧列表元素在新列表中的位置（下标），并以列表形式储
- `return_counts`：如果为`True`，返回去重数组中的元素在原数组中的出现次数

```
# 去重
array_un = np.unique(array_27)
print('array_un:\n', array_un)

array_un = np.unique(array_27, return_index = True)  # 去重并返回在原数组中的下标
print('array_un:\n', array_un)

'''
输出：
array_un:
 [ 0  1  2  3  4  5  6  7  8  9 10 11]
array_un:
 (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]), array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11], dtype=int64))
'''
```

十一、位运算

函数	描述
`bitwise_and`	按位与，对数组元素执行位与操作
`bitwise_or`	按位或，对数组元素执行位或操作
`bitwise_xor`	按位异或
`bitwise_not`	按位取反
`invert`	按位取反
`left_shift`	左移位运算，向左移动二进制表示的位
`right_shift`	右移位运算，向右移动二进制表示的位

> 也可以使用以下运算符：
>
> 1. **与运算（&）：** 对应位上的两个数字都为1时，结果为1；否则，结果为0。
>
>   例如：1010 & 1100 = 1000
> 2. **或运算（|）：** 对应位上的两个数字有一个为1时，结果为1；否则，结果为0。
>
>   例如：1010 | 1100 = 1110
> 3. **异或运算（^）：** 对应位上的两个数字相异时，结果为1；相同时，结果为0。

例如：1010 ^ 1100 = 0110

4. **取反运算（~）：** 对数字的每个位取反，即0变为1，1变为0。

   例如：~1010 = 0101

5. **左移运算（<<）：** 将数字的所有位向左移动指定的位数，右侧用0填充。

   例如：1010 << 2 = 101000

6. **右移运算（>>）：** 将数字的所有位向右移动指定的位数，左侧根据符号位或补零。

   例如：1010 >> 2 = 0010

```python
import numpy as np

arr1 = np.array([True, False, True], dtype=bool)
arr2 = np.array([False, True, False], dtype=bool)

result_and = np.bitwise_and(arr1, arr2)
result_or = np.bitwise_or(arr1, arr2)
result_xor = np.bitwise_xor(arr1, arr2)
result_not = np.bitwise_not(arr1)

print("AND:", result_and)  # [False, False, False]
print("OR:", result_or)    # [True, True, True]
print("XOR:", result_xor)  # [True, True, True]
print("NOT:", result_not)  # [False, True, False]

# 按位取反
arr_invert = np.invert(np.array([1, 2], dtype=np.int8))
print("Invert:", arr_invert)  # [-2, -3]

# 左移位运算
arr_left_shift = np.left_shift(5, 2)
print("Left Shift:", arr_left_shift)  # 20

# 右移位运算
arr_right_shift = np.right_shift(10, 1)
print("Right Shift:", arr_right_shift)  # 5
```

十二、字符串函数

函数	描述
`add()`	对两个数组的逐个字符串元素进行连接
`multiply()`	返回按元素多重连接后的字符串
`center()`	居中字符串

函数	描述
`capitalize()`	将字符串第一个字母转换为大写
`title()`	将字符串的每个单词的第一个字母转换为大写
`lower()`	数组元素转换为小写
`upper()`	数组元素转换为大写
`split()`	指定分隔符对字符串进行分割，并返回数组列表
`splitlines()`	返回元素中的行列表，以换行符分割
`strip()`	移除元素开头或者结尾处的特定字符
`join()`	通过指定分隔符来连接数组中的元素
`replace()`	使用新字符串替换字符串中的所有子字符串
`decode()`	数组元素依次调用`str.decode`， `numpy.char.decode()` 函数对编码的元素进行 `str.decode()` 解码。
`encode()`	数组元素依次调用`str.encode`， `numpy.char.encode()` 函数对数组中的每个元素调用 `str.encode` 函数。 默认编码是`utf-8`，可以使用标准 `Python` 库中的编解码器。

十三、数学函数

三角函数

```
import numpy as np

a = np.array([0,30,45,60,90])
print ('不同角度的正弦值：')
# 通过乘 pi/180 转化为弧度
print (np.sin(a*np.pi/180))
print ('\n')
print ('数组中角度的余弦值：')
print (np.cos(a*np.pi/180))
print ('\n')
print ('数组中角度的正切值：')
print (np.tan(a*np.pi/180))

'''
输出：
不同角度的正弦值：
[0.         0.5        0.70710678 0.8660254  1.        ]
```

```
数组中角度的余弦值：
[1.00000000e+00 8.66025404e-01 7.07106781e-01 5.00000000e-01
 6.12323400e-17]


数组中角度的正切值：
[0.00000000e+00 5.77350269e-01 1.00000000e+00 1.73205081e+00
 1.63312394e+16]
'''
```

舍入函数

`numpy.around()`

`numpy.around()` 函数返回指定数字的四舍五入值。

```
numpy.around(a,decimals)
```

> 参数说明：
>
> - `a`: 数组
> - `decimals`: 舍入的小数位数。 默认值为0。 如果为负，整数将四舍五入到小数点左侧的位置

`numpy.floor()`

`numpy.floor()` 返回小于或者等于指定表达式的最大整数，即向下取整。

`numpy.ceil()`

`numpy.ceil()` 返回大于或者等于指定表达式的最小整数，即向上取整。

十四、算数运算

`numpy` 中包含简单的**加减乘除**: **add()**，**subtract()**，**multiply()** 和 **divide()**，需要注意的是数组必须具有相同的形状或符合数组广播规则。

```
import numpy as np

a = np.arange(9, dtype = np.float_).reshape(3,3)
print ('第一个数组：')
print (a)
print ('\n')
print ('第二个数组：')
```

```
b = np.array([10,10,10])
print (b)
print ('\n')
print ('两个数组相加：')
print (np.add(a,b))
print ('\n')
print ('两个数组相减：')
print (np.subtract(a,b))
print ('\n')
print ('两个数组相乘：')
print (np.multiply(a,b))
print ('\n')
print ('两个数组相除：')
print (np.divide(a,b))

'''
输出：
第一个数组：
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]


第二个数组：
[10 10 10]


两个数组相加：
[[10. 11. 12.]
 [13. 14. 15.]
 [16. 17. 18.]]


两个数组相减：
[[-10.  -9.  -8.]
 [ -7.  -6.  -5.]
 [ -4.  -3.  -2.]]


两个数组相乘：
[[ 0. 10. 20.]
 [30. 40. 50.]
 [60. 70. 80.]]


两个数组相除：
[[0.  0.1 0.2]
 [0.3 0.4 0.5]
```

```
 [0.6 0.7 0.8]]
'''
```

numpy.reciprocal()

`numpy.reciprocal()` 函数返回参数逐元素的倒数。如 **1/4** 倒数为 **4/1**。

> 参数：一个数组

numpy.power()

`numpy.power()` 函数将第一个输入数组中的元素作为底数，计算它与第二个输入数组中相应元素的幂。

> 参数：两个数组，第一个表示底数，第二个表示幂

numpy.mod()

`numpy.mod()` 计算输入数组中相应元素的相除后的余数。 函数 `numpy.remainder()` 也产生相同的结果。

> 参数：两个数组，第一个是被除数，第二个是除数

十五、统计

numpy.amin()

`numpy.amin()` 用于计算数组中的元素沿指定轴的最小值。

```
numpy.amin(a, axis=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>)
```

> 参数说明：
>
> - `a`: 输入的数组，可以是一个`Numpy`数组或类似数组的对象。
> - `axis`: 可选参数，用于指定在哪个轴上计算最小值。如果不提供此参数，则返回整个数组的最小值。可以是一个整数表示轴的索引，也可以是一个元组表示多个轴。
> - `out`: 可选参数，用于指定结果的存储位置。
> - `keepdims`: 可选参数，如果为`True`，将保持结果数组的维度数目与输入数组相同。如果为`False`（默认值），则会去除计算后维度为1的轴。
> - `initial`: 可选参数，用于指定一个初始值，然后在数组的元素上计算最小值。
> - `where`: 可选参数，一个布尔数组，用于指定仅考虑满足条件的元素。

numpy.amax()

`numpy.amax()` 用于计算数组中的元素沿指定轴的最大值。

```
numpy.amax(a, axis=None, out=None, keepdims=<no value>, initial=<no
value>, where=<no value>)
```

> 参数说明：
>
> - `a`: 输入的数组，可以是一个NumPy数组或类似数组的对象。
> - `axis`: 可选参数，用于指定在哪个轴上计算最大值。如果不提供此参数，则返回整个数组的最大值。可以是一个整数表示轴的索引，也可以是一个元组表示多个轴。
> - `out`: 可选参数，用于指定结果的存储位置。
> - `keepdims`: 可选参数，如果为True，将保持结果数组的维度数目与输入数组相同。如果为False（默认值），则会去除计算后维度为1的轴。
> - `initial`: 可选参数，用于指定一个初始值，然后在数组的元素上计算最大值。
> - `where`: 可选参数，一个布尔数组，用于指定仅考虑满足条件的元素。

numpy.ptp()

`numpy.ptp()` 函数计算数组中元素最大值与最小值的差（最大值 - 最小值）。

```
numpy.ptp(a, axis=None, out=None, keepdims=<no value>, initial=<no
value>, where=<no value>)
```

> 参数说明：
>
> - `a`: 输入的数组，可以是一个 `Numpy` 数组或类似数组的对象。
> - `axis`: 可选参数，用于指定在哪个轴上计算 `峰-峰值`。如果不提供此参数，则返回整个数组的 `峰-峰值`。可以是一个整数表示轴的索引，也可以是一个元组表示多个轴。
> - `out`: 可选参数，用于指定结果的存储位置。
> - `keepdims`: 可选参数，如果为 `True`，将保持结果数组的维度数目与输入数组相同。如果为 `False`（默认值），则会去除计算后维度为 `1` 的轴。
> - `initial`: 可选参数，用于指定一个初始值，然后在数组的元素上计算 `峰-峰值`。
> - `where`: 可选参数，一个布尔数组，用于指定仅考虑满足条件的元素。

numpy.percentile()

百分位数是统计中使用的度量，表示小于这个值的观察值的百分比，返回一个值表示的是数组中小于这个值的数的个数占整个数组的百分比为指定的百分比`q`。 函数`numpy.percentile()`接受以下参数。

```
numpy.percentile(a, q, axis)
```

> 参数说明:
>
> - `a`: 输入数组
> - `q`: 要计算的百分位数，在`0 ~ 100`之间
> - `axis`: 沿着它计算百分位数的轴

numpy.median()

`numpy.median()`函数用于计算数组`a`中元素的中位数（中值）

```
numpy.median(a, axis=None, out=None, overwrite_input=False, keepdims=
<no value>)
```

> 参数说明:
>
> - `a`: 输入的数组，可以是一个`Numpy`数组或类似数组的对象。
> - `axis`: 可选参数，用于指定在哪个轴上计算中位数。如果不提供此参数，则计算整个数组的中位数。可以是一个整数表示轴的索引，也可以是一个元组表示多个轴。
> - `out`: 可选参数，用于指定结果的存储位置。
> - `overwrite_input`: 可选参数，如果为`True`，则允许在计算中使用输入数组的内存。这可能会在某些情况下提高性能，但可能会修改输入数组的内容。
> - `keepdims`: 可选参数，如果为`True`，将保持结果数组的维度数目与输入数组相同。如果为`False`（默认值），则会去除计算后维度为`1`的轴。

numpy.mean()

`numpy.mean()`函数返回数组中元素的算术平均值，如果提供了轴，则沿其计算。

算术平均值是沿轴的元素的总和除以元素的数量。

```
numpy.mean(a, axis=None, dtype=None, out=None, keepdims=<no value>)
```

> 参数说明:
>
> - `a`: 输入的数组，可以是一个`Numpy`数组或类似数组的对象。

- `axis`: 可选参数，用于指定在哪个轴上计算平均值。如果不提供此参数，则计算整个数组的平均值。可以是一个整数表示轴的索引，也可以是一个元组表示多个轴。
- `dtype`: 可选参数，用于指定输出的数据类型。如果不提供，则根据输入数据的类型选择合适的数据类型。
- `out`: 可选参数，用于指定结果的存储位置。
- `keepdims`: 可选参数，如果为`True`，将保持结果数组的维度数目与输入数组相同。如果为`False`（默认值），则会去除计算后维度为`1`的轴。

`numpy.average()`

`numpy.average()`函数根据在另一个数组中给出的各自的权重计算数组中元素的加权平均值。

该函数可以接受一个轴参数。 如果没有指定轴，则数组会被展开。

加权平均值即将各数值乘以相应的权数，然后加总求和得到总体值，再除以总的单位数。

考虑数组`[1,2,3,4]`和相应的权重`[4,3,2,1]`，通过将相应元素的乘积相加，并将和除以权重的和，来计算加权平均值。

```
加权平均值 = (1*4+2*3+3*2+4*1)/(4+3+2+1)
```

函数语法：

```
numpy.average(a, axis=None, weights=None, returned=False)
```

参数说明：

- `a`: 输入的数组，可以是一个 `Numpy` 数组或类似数组的对象。
- `axis`: 可选参数，用于指定在哪个轴上计算加权平均值。如果不提供此参数，则计算整个数组的加权平均值。可以是一个整数表示轴的索引，也可以是一个元组表示多个轴。
- `weights`: 可选参数，用于指定对应数据点的权重。如果不提供权重数组，则默认为等权重。
- `returned`: 可选参数，如果为`True`，将同时返回加权平均值和权重总和。

标准差

```
import numpy as np

print (np.std([1,2,3,4]))
'''
输出:
1.1180339887498949
'''
```

方差

```
import numpy as np

print (np.var([1,2,3,4]))
'''
输出:
1.25
'''
```

十六、排序

`numpy.sort()`

`numpy.sort()` 函数返回输入数组的排序副本。函数格式如下:

```
numpy.sort(a, axis, kind, order)
```

> 参数说明:
>
> - `a`: 要排序的数组
> - `axis`: 沿着它排序数组的轴，如果没有数组会被展开，沿着最后的轴排序，`axis=0` 按列排序，`axis=1` 按行排序
> - `kind`: 默认为`quicksort（快速排序）`
> - `order`: 如果数组包含字段，则是要排序的字段

`numpy.argmax()` 和`numpy.argmin()`

`numpy.argmax()` 和 `numpy.argmin()`函数分别沿给定轴返回最大和最小元素的索引。

> 参数：一个数组，和可选的轴`axis`

numpy.argsort()

`numpy.argsort()` 函数返回的是数组值从小到大的索引值。

> 参数：一个数组

numpy.lexsort()

> 参数：　一个元组（元组中可以包含多个数组）

`numpy.lexsort()` 用于对多个序列进行排序。把它想象成对电子表格进行排序，每一列代表一个序列，排序时优先照顾靠后的列。

这里举一个应用场景：小升初考试，重点班录取学生按照总成绩录取。在总成绩相同时，数学成绩高的优先录取，在总成绩和数学成绩都相同时，按照英语成绩录取..... 这里，总成绩排在电子表格的最后一列，数学成绩在倒数第二列，英语成绩在倒数第三列。

numpy.nonzero()

`numpy.nonzero()` 函数返回输入数组中非零元素的索引。

> 参数：一个数组

numpy.extract()

`numpy.extract()` 函数根据某个条件从数组中抽取元素，返回满条件的元素。

> 参数：条件数组（可以是一个bool数组），和一个需要筛选数据的数组。

```
import numpy as np

x = np.arange(9.).reshape(3,  3)
print ('我们的数组是：')
print (x)
# 定义条件, 选择偶数元素
condition = np.mod(x,2)  ==  0
print ('按元素的条件值：')
print (condition)
print ('使用条件提取元素：')
print (np.extract(condition, x))

'''
输出：
我们的数组是：
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
按元素的条件值：
```

```
[[ True False  True]
 [False  True False]
 [ True False  True]]
使用条件提取元素：
[0. 2. 4. 6. 8.]
'''
```

十七、线性代数

`Numpy` 提供了线性代数函数库 `linalg`，该库包含了线性代数所需的所有功能，可以看看下面的说明：

函数	描述
`dot`	两个数组的点积，即元素对应相乘。
`vdot`	两个向量的点积
`inner`	两个数组的内积
`matmul`	两个数组的矩阵积
`det`	数组的行列式
`solve`	求解线性矩阵方程
`inv`	计算矩阵的乘法逆矩阵

表中所函数的使用方法都需要使用`np.linalg.函数名`的方式，如：`np.linalg.dot(a, b)`

前四个函数的参数都为两个数组，第四个和第六个的参数都是一个数组。

*第五个用于**求解线性方程组的解**，参数为两个数组，一个为**系数数组**，一个为**结果数组**。*

让我们通过一个例子来说明如何使用 `numpy.linalg.solve()` 函数：

假设有如下线性方程组：

```
2x + 3y = 7
4x - 2y = 6
```

我们可以用矩阵形式表示为：

```
[[2, 3], [4, -2]] * [[x], [y]] = [[7], [6]]
```

其中，系数矩阵 `a` 为 `[[2, 3], [4, -2]]`，右侧常数向量 `b` 为 `[[7], [6]]`。

现在，我们可以使用 `numpy.linalg.solve()` 函数来求解这个线性方程组：

```
import numpy as np

# 定义系数矩阵 a 和右侧常数向量 b
a = np.array([[2, 3], [4, -2]])
b = np.array([7, 6])

# 求解线性方程组
solution = np.linalg.solve(a, b)

print("Solution:", solution)
```

运行这段代码会得到结果：

```
Solution: [1. 2.]
```

这表示方程组的解为 `x = 1`，`y = 2`。