

NumPy 安装

Anaconda 中自带 numpy 库，无需另行安装

Python标准版本，需要手动安装 numpy:

- 查看当前版本: `print(np.__version__)`
- 安装最新的稳定版本: `pip install numpy`
- 安装指定版本: `pip install numpy==版本号`

NumPy 简介

NumPy (Numerical Python) 是 Python 进行科学计算的一个扩展库，提供了大量的函数和操作，主要用于对多维数组执行计算，它比 Python 自身的嵌套列表结构要高效的多

NumPy 数组和 Python 列表的区别:

- NumPy 数组中的元素都需要具有相同的数据类型
- NumPy 数组在创建时具有固定的大小，与 Python 的列表（可以动态增长）不同，更改数组的大小将创建一个新的数组

创建数组

从现有的数据创建

`np.array(object, dtype=None)`

- **object:** `array_like`，类似于数组的对象。如果`object`是标量，则返回包含`object`的0维数组
- **dtype:** `data-type`，数组所需的数据类型。如果没有给出，会从输入数据推断数据类型
- 创建一个数组对象并返回（`ndarray`实例对象）

DTYPE常用值 描述

`np.int8` 字节（-128 to 127）

`np.int16` 整数（-32768 to 32767）

`np.int32` 整数（-2147483648 to 2147483647）

`np.int64` 整数（-9223372036854775808 to 9223372036854775807）

`np.uint8` 无符号整数（0 to 255）

`np.uint16` 无符号整数（0 to 65535）

`np.uint32` 无符号整数（0 to 4294967295）

`np.uint64` 无符号整数（0 to 18446744073709551615）

`np.float16` 半精度浮点数

`np.float32` 单精度浮点数

`np.float64` 双精度浮点数

NDARRAY常用属性 描述

`ndarray.ndim` 秩，即轴的数量或维度的数量

`ndarray.shape` 数组的形状

NDARRAY常用属性	描述
-------------	----

ndarray.size	数组元素的总个数
--------------	----------

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
------------------	---------------------------

```
import numpy as np
from typing import Iterable

# arr1 = np.array((1, 2, 3))
# arr1 = np.array(range(1, 4))
arr1 = np.array([1, 2, 3])
print(arr1)
print(type(arr1))
print(isinstance(arr1, np.ndarray))
print(isinstance(arr1, Iterable))

print(arr1.ndim)
print(arr1.shape)
print(arr1.size)
print(arr1.dtype)
print(arr1.itemsize)

arr2 = np.array([[1, 2, 3], [4, 5, 2147483648]])
print(arr2)
print(arr2.ndim)
print(arr2.shape)
print(arr2.size)
print(arr2.dtype)
print(arr2.itemsize)
```

```
arr3 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12.]])
print(arr3)
print(arr3.ndim)
print(arr3.shape)
print(arr3.size)
print(arr3.dtype)
print(arr3.itemsize)
```

```
arr4 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]], dtype=np.float32)
print(arr4)
print(arr4.ndim)
print(arr4.shape)
print(arr4.size)
print(arr4.dtype)
print(arr4.itemsize)
```

```
arr5 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12.]]], dtype=np.int32)
print(arr5)
print(arr5.ndim)
print(arr5.shape)
print(arr5.size)
print(arr5.dtype)
print(arr5.itemsize)
```

```
arr6 = np.array(13)
print(arr6)
print(type(arr6))
print(arr6.ndim)
print(arr6.shape)
```

```
print(arr6.size)
print(arr6.dtype)
print(arr6.itemsize)
```

`np.asarray(a, dtype=None)`

- 类似于 `np.array`，主要区别是当 `a` 是 `ndarray` 且 `dtype` 也匹配时，`np.asarray` 不执行复制操作，而 `np.array` 仍然会复制出一个副本，占用新的内存

```
import numpy as np

obj = [1, 2, 3] # a是array_like
# obj = np.array([1, 2, 3]) # a是ndarray
arr1 = np.array(obj)
arr2 = np.asarray(obj)
obj[1] = 4
print(obj)
print(arr1)
print(arr2)

# asarray中的dtype和obj不匹配
obj = np.array([1, 2, 3])
arr1 = np.array(obj, dtype=np.float32)
arr2 = np.asarray(obj, dtype=np.float32)
obj[1] = 4
print(obj)
print(arr1)
print(arr2)
```

`np.copy(a)`

- `a`: `array_like`

- 返回给定对象的数组副本

```
import numpy as np

a1 = [1, 2, 3]
arr1 = np.copy(a1)
print(arr1)

a2 = np.array([1, 2, 3])
arr2 = np.copy(a2)
print(arr2)
```

ndarray.copy()

- 对象方法，返回数组的副本

```
import numpy as np

arr = np.array([1, 2, 3])
print(arr.copy())
```

np.fromiter(iterable, dtype, count=-1)

- **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，表示读取所有数据
- 从可迭代对象创建一个新的 一维数组 并返回

```
import numpy as np

iterable = (x*x for x in range(5))
print(np.fromiter(iterable, dtype=np.float64))
```

从形状或值创建

`np.empty(shape, dtype=np.float64)`

- 返回给定形状和类型且未初始化的新数组

```
import numpy as np

print(np.empty((2, 3)))
print(np.empty((2, 3), dtype=np.int32))
```

`np.empty_like(prototype, dtype=None)`

- `prototype`: `array_like`
- `dtype`: 如果指定该参数，将会覆盖结果的数据类型
- 返回形状和类型与给定 `prototype` 相同的新数组

```
import numpy as np

a = ([1, 2, 3], [4, 5, 6])
print(np.empty_like(a))

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

a = np.array([[1, 2, 3], [4, 5, 6.]])
print(np.empty_like(a, dtype=np.int32))
```

`np.zeros(shape, dtype=np.float64)`

- 返回给定形状和类型的新数组，并用零填充

```
import numpy as np

print(np.zeros((2, 3)))
print(np.zeros((2, 3), dtype=np.int32))
```

`np.zeros_like(a, dtype=None)`

- **a:** array_like
- **dtype:** 如果指定该参数，将会覆盖结果的数据类型
- 返回一个与给定 **a** 具有相同形状和类型的零数组。


```
import numpy as np

a = ([1, 2, 3], [4, 5, 6])
print(np.zeros_like(a))

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

a = np.array([[1, 2, 3], [4, 5, 6.]])
print(np.zeros_like(a, dtype=np.int32))
```

`np.ones(shape, dtype=np.float64)`

- 返回给定形状和类型的新数组，并用1填充

```
import numpy as np

print(np.ones((2, 3)))
print(np.ones((2, 3), dtype=np.int32))
```

`np.ones_like(a, dtype=None)`

- `a`: `array_like`
- `dtype`: 如果指定该参数，将会覆盖结果的数据类型
- 返回一个与给定 `a` 具有相同形状和类型的1构成的数组。

```
import numpy as np

a = ([1, 2, 3], [4, 5, 6])
print(np.ones_like(a))

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

a = np.array([[1, 2, 3], [4, 5, 6.]])
print(np.ones_like(a, dtype=np.int32))
```

`np.full(shape, fill_value, dtype=None)`

- 返回给定形状和类型的新数组，并用 `fill_value` 填充

```
import numpy as np

print(np.full((2, 3), 6))
print(np.full((2, 3), 6.))
print(np.full((2, 3), 6., dtype=np.int32))
```

`np.full_like(a, fill_value, dtype=None)`

- `a`: `array_like`
- `dtype`: 如果指定该参数，将会覆盖结果的数据类型
- 返回一个与给定 `a` 具有相同形状和类型的 `fill_value` 填充的数组

```
import numpy as np

a = ([1, 2, 3], [4, 5, 6])
print(np.full_like(a, 6))

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

a = np.array([[1, 2, 3], [4, 5, 6.]])
print(np.full_like(a, 6., dtype=np.int32))
```

`np.eye(N, M=None, k=0, dtype=np.float64)`

- **N**: 输出数组的行数
- **M**: 输出数组的列数。如果为`None`，则默认为`N`
- **k**: 对角线的索引，默认为`0`，表示主对角线，正值表示上对角线，负值表示下对角线
- **dtype**: 数组的数据类型
- 对角线为`1`，其他地方为`0`（单位矩阵）

```
import numpy as np

print(np.eye(3))
print(np.eye(3, 4))
print(np.eye(3, 4, k=1))
print(np.eye(3, 4, k=1, dtype=np.int32))
```

`np.identity(n, dtype=np.float64)`

- 返回 $n \times n$ 的单位数组（主对角线为`1`，其他元素为`0`的方形数组）

```
import numpy as np

print(np.identity(3))
print(np.identity(3, dtype=np.int32))
```

从数值范围创建数组

`np.arange([start,] stop[, step,], dtype=None)`

- 返回给定区间内的均匀间隔值构成的数组

```
import numpy as np

print(np.arange(3))
print(np.arange(3.0))
print(np.arange(3, 7))
print(np.arange(3, 7, dtype=np.float64))
print(np.arange(3, 7, 2))
print(np.arange(7, 3, -2))
print(np.arange(3, 7, 0.5))
```

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

- **num**: 生成的样本数量
- **endpoint**: 如果为 `True`, **stop** 为最后一个样本。否则, 不包括在内
- **retstep**: 如果为 `True`, 返回 `(samples, step)`, **step** 是样本之间的间隔
- **dtype**: 如果没有给出 **dtype**, 则从 **start** 和 **stop** 推断数据类型。推断出的 **dtype** 永远不会是整数; 即使参数会产生一个整数数组, 也会选择 `np.float64`

- 把给定区间分成 **num** 个均匀间隔的样本，构成数组并返回（等差数列）

```
import numpy as np

print(np.linspace(1, 50))
print(np.linspace(1, 10, num=10))
print(np.linspace(1, 10, num=10, endpoint=False))
print(np.linspace(1, 10, num=10, retstep=True))
print(np.linspace(1, 10, num=10, dtype=np.int32))
```

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

- **start**: 序列的起始值为: $\text{base}^{\text{start}}$
- **stop**: 序列的终止值为: $\text{base}^{\text{stop}}$
- **num**: 生成的样本数量
- **endpoint**: 如果为 **True**, **stop** 为最后一个样本。否则，不包括在内
- **base**: 对数 \log 的底数
- **dtype**: 如果没有给出 **dtype**, 则从 **start** 和 **stop** 推断数据类型。推断出的 **dtype** 永远不会是整数; 即使参数会产生一个整数数组, 也会选择 `np.float64`
- 把给定区间分成 **num** 个按对数尺度均匀间隔的样本，构成数组并返回（等比数列）

```
import numpy as np

print(np.logspace(2.0, 3.0, num=4))
print(np.logspace(0, 9, 10, base=2))
```

基本运算

算术和比较操作 `ndarrays` 被定义为逐元素操作

```
import numpy as np

a = np.array([[1, 2], [3, 4], [5, 6]])
print(a + 2)
print(a - 2)
print(a * 2)
print(a / 2)
print(a < 4)
print(a > 3)

b = np.array([[2, 2], [2, 1], [1, 1]])
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a < b)
print(a > b)
```

广播机制

- 后缘维度相同或者不同的维度有1，可以广播

```
import numpy as np
```

```
a = np.arange(24).reshape((2, 3, 4))
b = np.arange(12).reshape((3, 4))
c = np.arange(4).reshape((1, 4))
d = np.arange(4).reshape(4)
e = np.arange(12).reshape((1, 3, 4))
f = np.arange(6).reshape((2, 3, 1))
g = np.arange(2).reshape((2, 1, 1))
h = np.arange(2).reshape((1, 2, 1, 1))
i = np.arange(10).reshape((5, 2, 1, 1))

print((a + b).shape)
print((a + c).shape)
print((a + d).shape)
print((a + e).shape)
print((a + f).shape)
print((a + g).shape)
print((a + h).shape)
print((a + i).shape)
```

索引和切片

数组的索引和切片与 Python 中序列的索引和切片操作基本类似，不同点在于：

- 数组切片不会复制内部数组数据，只会生成原始数据的新视图
- 数组支持多维数组的多维索引和切片

```
import numpy as np
```

```
arr = np.arange(2, 10)
print(arr)
item = arr[1]
print(item)
part = arr[-2: 1: -1]
print(part)

arr[1] = 33 # 索引修改数组
arr[3:] = [55, 66, 77, 88, 99] # 切片修改数组
print(item) # 非视图
print(part) # 新视图
```

```
import numpy as np

lis = [[1, 2], [3, 4], [5, 6]]
arr = np.array(lis)
print(arr[1, 0])
print(arr[1][0])
print(arr[:2, 1:2])
print(arr[:2][1:2])
```



```
import numpy as np

x = np.array([[1, 2], [3, 4], [5, 6]])
print(x[[1, 2]]) # 等价于x[1]和x[2]组成的数组
print(x[[0, 1, 2], [0, 1, 0]]) # 等价于x[0, 0]、x[1, 1]和x[2, 0]组成的数组

print(x[[True, False, True]])
print(x < 4)
print(x[x < 4])
print(x[np.array([[True, True], [True, False], [False, False]])])
```

常用操作

`np.reshape(a, newshape)`

- 保证 `size` 不变，在不更改数据的情况下为数组赋予新的形状
- `newshape`如果是整数，则结果将是该长度的1-D数组
- `newshape`的一个形状维度可以是-1，值将自行推断

```
import numpy as np

arr1 = np.arange(6).reshape((2, 1, 3))
arr2 = np.reshape(arr1, 6)
arr3 = np.reshape(arr1, -1)
arr4 = np.reshape(arr1, (-1,))
print(arr2) # [0 1 2 3 4 5]
```

```

print(arr3)  # [0 1 2 3 4 5]
print(arr4)  # [0 1 2 3 4 5]

arr1 = np.arange(24)
arr2 = np.reshape(arr1, (2, 2, -1, 2))
print(arr2.shape)  # (2, 2, 3, 2)

```

`np.resize(a, new_shape)`

- 返回具有指定形状的新数组

`ndarray.resize(new_shape)`

- 直接修改原数组的形状，无返回值

```

import numpy as np

a = np.array([[0, 1], [2, 3]])
print(np.resize(a, (1, 2)))
print(np.resize(a, (2, 4)))
a.resize((1, 4))
print(a)
a.resize((2, 4))
print(a)

""" 当数组的总大小不变时，应使用 reshape
    而 resize 是允许总大小发生改变的 """
print(np.reshape(a, (1, 4)))
# print(np.reshape(a, (2, 4))) # 总大小发生改变，所以报错

```

`ndarray.flatten()`

- 返回扁平化到一维的数组

```
import numpy as np

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

ndarray.T

- 转置数组

```
import numpy as np

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

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

np.swapaxes(a, axis1, axis2)

- 交换数组的两个轴

```
import numpy as np

x = np.array([[1, 2, 3]])
print(x)
print(x.shape)
y = np.swapaxes(x, 0, 1)
print(y.shape)
print(y)
```

```
x = np.arange(6).reshape((1, 2, 3))
print(x)
print(x.shape)
y = np.swapaxes(x, 0, 2)
print(y)
print(y.shape)
```

`np.transpose(a, axes=None)`

- 通过`axes`参数排列数组的`shape`，如果`axes`省略，则
`transpose(a).shape == a.shape[::-1]`

```
import numpy as np

a = np.ones((2, 3, 4, 5))
print(np.transpose(a).shape)

a = np.ones((1, 2, 3))
print(np.transpose(a, (1, 0, 2)).shape)

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

`np.expand_dims(a, axis)`

- 扩展数组的形状

```
import numpy as np

x = np.array([1, 2])
```

```

print(x.shape)

y = np.expand_dims(x, axis=0)
print(y.shape)

y = np.expand_dims(x, axis=1)
print(y.shape)

y = np.expand_dims(x, axis=(0, 1))
print(y.shape)

y = np.expand_dims(x, axis=(2, 0))
print(y.shape)

```

`np.squeeze(a, axis=None)`

- 从给定数组的形状中删除一维的条目

```

import numpy as np

x = np.array([[[0], [1], [2]]])
print(x.shape)

print(np.squeeze(x).shape)
print(np.squeeze(x, axis=0).shape)
print(np.squeeze(x, axis=2).shape)

```

`np.concatenate((a1, a2, ...), axis=0)`

- 沿现有轴连接一系列数组，如果axis为None，则数组在使用前会被扁平化

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
print(np.concatenate((a, b), axis=0))
print(np.concatenate((a, b.T), axis=1))
print(np.concatenate((a, b), axis=None))
```

`np.stack(arrays, axis=0)`

- `arrays`: Sequence[ArrayLike]
- 沿新轴连接一系列数组

```
import numpy as np

arrays = [np.ones((3, 4)) for _ in range(5)]
print(np.stack(arrays, axis=0).shape)
print(np.stack(arrays, axis=1).shape)
print(np.stack(arrays, axis=2).shape)
print(np.stack(arrays, axis=-1).shape)
```

`np.hstack(tup)`

- `tup`: Sequence[ArrayLike]
- 通过水平堆叠来生成数组
- 根据第二个轴进行拼接，如果是一维数据则根据第一个轴拼接

```
import numpy as np

a = np.array((1, 2, 3))
b = np.array((4, 5, 6))
print(np.hstack((a, b)))

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

np.vstack(tup)

- tup: Sequence[ArrayLike]
- 通过垂直堆叠来生成数组
- 根据第一个轴进行拼接，如果是一维(N,)数据，则转成二维(1, N)数据，再根据第一个轴拼接

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
print(np.vstack((a,b)))

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

np.repeat(a, repeats, axis=None)

- repeats: 重复次数
- axis: 指定重复值的轴。没指定时，默认扁平化处理
- 重复数组中的元素

```
import numpy as np

print(np.repeat(3, 4))

x = np.array([[1, 2], [3, 4]])
print(np.repeat(x, 2))
print(np.repeat(x, 3, axis=1))
print(np.repeat(x, [1, 2], axis=0))
```

`np.unique(ar, return_index=False, return_inverse=False, return_counts=False, axis=None)`

- **ar**: 输入的数组
- **return_index**: 为True时，还会返回新数组元素在旧数组中的下标
- **return_inverse**: 为True时，还会返回旧数组元素在新数组中的下标
- **return_counts**: 为True时，还会返回去重数组中的元素在原数组中的出现次数
- **axis**: 指定操作的轴。没指定时，默认扁平化处理
- 返回数组中排序的唯一元素，重复元素会被去除，只保留一个

```
import numpy as np

print(np.unique([3, 3, 1, 1, 2, 2]))

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

a = np.array([[1, 0, 0], [1, 0, 0], [2, 3, 4]])
print(np.unique(a, axis=0))

a = np.array([1, 2, 6, 4, 2, 3, 2])
print(np.unique(a, return_index=True))
print(np.unique(a, return_inverse=True))
```



```
print(np.unique(a, return_counts=True))
```

`np.dot(a, b)`

- 两个数组的点积

```
import numpy as np

a = [1, 2, 3]
b = [1, 0, 2]
print(np.dot(a, b))

a = [[1, 0], [0, 1]]
b = [[4, 1], [2, 2]]
print(np.dot(a, b))

print(np.dot(3, 4))

a = 2
b = [[4, 1], [2, 2]]
print(np.dot(a, b))
```

`np.matmul(x1, x2)`

@操作符也可表示

- 两个数组的矩阵乘积

```
import numpy as np

a = np.array([[1, 0],
               [0, 1]])
b = np.array([[4, 1],
```

```

        [2, 2]])
print(np.matmul(a, b))
print(a @ b)

a = np.array([[1, 0],
              [0, 1]])
b = np.array([1, 2])
print(np.matmul(a, b))
print(a @ b)

```

`np.greater(x1, x2)`

- `x1`, `x2`的形状必须相同，或者可以广播
- 按元素判断 `x1 > x2` 的结果

```

import numpy as np

print(np.greater([4, 2], [2, 2]))

a = np.array([[4, 2], [3, 1]])
b = np.array([[2, 2]])
print(np.greater(a, b))
print(a > b)

```

`np.greater_equal(x1, x2)`

- `x1`, `x2`的形状必须相同，或者可以广播
- 按元素判断 `x1 >= x2` 的结果

```
import numpy as np

print(np.greater_equal([4, 2], [2, 2]))

a = np.array([[4, 2], [3, 1]])
b = np.array([[2, 2]])
print(np.greater_equal(a, b))
print(a >= b)
```

`np.less(x1, x2)`

- `x1`, `x2`的形状必须相同，或者可以广播
- 按元素判断 $x1 < x2$ 的结果

```
import numpy as np

print(np.less([4, 2], [2, 2]))

a = np.array([[4, 2], [3, 1]])
b = np.array([[2, 2]])
print(np.less(a, b))
print(a < b)
```

`np.less_equal(x1, x2)`

- `x1`, `x2`的形状必须相同，或者可以广播
- 按元素判断 $x1 \leq x2$ 的结果

```
import numpy as np

print(np.less_equal([4, 2], [2, 2]))

a = np.array([[4, 2], [3, 1]])
b = np.array([[2, 2]])
print(np.less_equal(a, b))
print(a <= b)
```

`np.equal(x1, x2)`

- `x1`, `x2`的形状必须相同，或者可以广播
- 按元素判断 `x1 == x2` 的结果

```
import numpy as np

print(np.equal([4, 2], [2, 2]))

a = np.array([[4, 2], [3, 1]])
b = np.array([[2, 2]])
print(np.equal(a, b))
print(a == b)
```

`np.not_equal(x1, x2)`

- `x1`, `x2`的形状必须相同，或者可以广播
- 按元素判断 `x1 != x2` 的结果

```
import numpy as np

print(np.not_equal([4, 2], [2, 2]))

a = np.array([[4, 2], [3, 1]])
b = np.array([[2, 2]])
print(np.not_equal(a, b))
print(a != b)
```

np.sin(x)

- x: 角度（弧度值）
- 正弦函数

```
import numpy as np

print(np.sin(np.pi/2))
print(np.sin(np.array((0, 30, 90)) * np.pi / 180))
```

np.cos(x)

- x: 角度（弧度值）
- 余弦函数

```
import numpy as np

print(np.cos(np.pi/2))
print(np.cos(np.array((0, 60, 90)) * np.pi / 180))
```

np.tan(x)

- x : 角度（弧度值）
- 正切函数

```
import numpy as np

print(np.tan(-np.pi))
print(np.tan(np.array((0, 180)) * np.pi / 180))
```

`np.arcsin(x)`

- 反正弦函数

```
import numpy as np

print(np.arcsin(1)) # pi/2
print(np.arcsin(np.array([0.5, -0.5]))) # pi/6, -pi/6
```

`np.arccos(x)`

- 反余弦函数

```
import numpy as np

print(np.arccos(-1)) # pi
print(np.arccos(np.array([0.5, 1]))) # pi/3, 0
```

`np.arctan(x)`

- 反正切函数

```
import numpy as np

print(np.arctan(1)) # pi/4
print(np.arctan(np.array([0, -1]))) # 0, -pi/4
```

`np.floor(x)`

- 返回 x 的底限

```
import numpy as np

a = np.array([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0])
print(np.floor(a))
```

`np.ceil(x)`

- 返回 x 的上限

```
import numpy as np

a = np.array([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0])
print(np.ceil(a))
```

`ndarray.max(axis=None, keepdims=False)`

- 返回沿给定轴的最大值，`axis`没有指定时，默认为`None`，表示返回所有元素的最大值

```
import numpy as np

lis = [[0, 1, 7, 3], [4, 9, 6, 2], [8, 5, 11, 10]]
arr1 = np.array(lis)
print(arr1)
print(arr1.max())
print(arr1.max(axis=0))
print(arr1.max(axis=1))
```

`ndarray.min(axis=None, keepdims=False)`

- 返回沿给定轴的最小值，`axis`没有指定时，默认为`None`，表示返回所有元素的最小值

```
import numpy as np

lis = [[0, 1, 7, 3], [4, 9, 6, 2], [8, 5, 11, 10]]
arr1 = np.array(lis)
print(arr1)
print(arr1.min())
print(arr1.min(axis=0))
print(arr1.min(axis=1))
```

`ndarray.mean(axis=None, keepdims=False)`

- 返回沿给定轴的平均值，`axis`没有指定时，默认为`None`，表示返回所有元素的平均值


```
import numpy as np

lis = [[0, 1, 7, 3], [4, 9, 6, 2], [8, 5, 11, 10]]
arr1 = np.array(lis)
print(arr1)
print(arr1.mean())
print(arr1.mean(axis=0))
print(arr1.mean(axis=1))
```

`ndarray.var(axis=None, keepdims=False)`

- 返回沿给定轴的方差，`axis`没有指定时，默认为`None`，表示返回所有元素的方差

```
import numpy as np

lis = [[0, 1, 7, 3], [4, 9, 6, 2], [8, 5, 11, 10]]
arr1 = np.array(lis)
print(arr1)
print(arr1.var())
print(arr1.var(axis=0))
print(arr1.var(axis=1))
```

`ndarray.std(axis=None, keepdims=False)`

- 返回沿给定轴的标准差，`axis`没有指定时，默认为`None`，表示返回所有元素的标准差

```
import numpy as np

lis = [[0, 1, 7, 3], [4, 9, 6, 2], [8, 5, 11, 10]]
arr1 = np.array(lis)
print(arr1)
print(arr1.std())
print(arr1.std(axis=0))
print(arr1.std(axis=1))
```

`np.prod(a, axis=None, keepdims=np._NoValue, initial=np._NoValue)`

- 返回给定轴上数组元素的乘积

```
import numpy as np

# 默认的axis=None将计算输入数组中所有元素的乘积
print(np.prod([1, 2, 3, 4]))
print(np.prod([[1, 2], [3, 4]]))

print(np.prod([1, 2, 3, 4], initial=5))

print(np.prod([[1, 2], [3, 4]], axis=1))
print(np.prod([[1, 2], [3, 4]], axis=0))

print(np.prod([[1, 2], [3, 4]], axis=1, keepdims=True))
print(np.prod([[1, 2], [3, 4]], axis=0, keepdims=True))
```

`np.sum(a, axis=None, keepdims=np._NoValue, initial=np._NoValue)`

- 返回给定轴上数组元素的和

```
import numpy as np
```

```

# 默认的axis=None将计算输入数组中所有元素的和
print(np.sum([1, 2, 3, 4]))
print(np.sum([[1, 2], [3, 4]]))

print(np.sum([1, 2, 3, 4], initial=5))

print(np.sum([[1, 2], [3, 4]], axis=1))
print(np.sum([[1, 2], [3, 4]], axis=0))

print(np.sum([[1, 2], [3, 4]], axis=1, keepdims=True))
print(np.sum([[1, 2], [3, 4]], axis=0, keepdims=True))

```

np.exp(x)

- 计算 e 的 x 幂次方

```

import numpy as np

# e的0次方、e的1次方、e的2次方
print(np.exp([0, 1, 2]))

```

np.log(x)

- 计算 x 的自然对数

```

import numpy as np

print(np.log([1, np.e, np.e**2]))

```

np.log2(x)

- 计算 x 的以 2 为底的对数

```
import numpy as np

x = np.array([1, 2, 2**4])
print(np.log2(x))
```

np.log10(x)

- 计算 x 的以 10 为底的对数

```
import numpy as np

print(np.log10([1e-15, 1000]))
```

np.sort(a, axis=-1)

- 返回排序之后的新数组

```
import numpy as np

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

# 如果指定为None, 则数组扁平化处理
print(np.sort(a, axis=None))

# axis默认为-1, 表示最后一个维度
print(np.sort(a))

print(np.sort(a, axis=0))
```

np.nonzero(a)

- 返回非零元素的索引

```
import numpy as np

x = np.array([[3, 0, 0], [0, 4, 0], [5, 6, 0]])
print(x)
print(np.nonzero(x))
print(x[np.nonzero(x)])

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

np.where(condition, x=None, y=None)

- condition : array_like, bool
- x, y: array_like, 要么都传参, 要么都不传

- 如果传三个参数，条件成立返回x，不成立时返回y
- 如果只传第一个参数，返回符合条件的元素的索引

```
import numpy as np

a = np.arange(10)
print(np.where(a < 5, a, 10*a))

# 当where内有三个参数时，当condition成立时返回x，当condition
# 不成立时返回y
print(np.where([[True, False], [True, True]], [[1, 2],
[3, 4]], [[9, 8], [7, 6]]))

# 如果只传第一个参数，返回符合条件的元素的索引
a = np.array([2, 4, 6, 8, 10])
print(np.where(a > 5))
```

np.argwhere(a)

- 找出数组中按元素分组的非零元素的索引

```
import numpy as np

x = np.arange(6).reshape(2, 3)
print(x)
print(x>1)
print(np.argwhere(x>1))
```

np.maximum(x1, x2)

- 返回x1和x2逐个元素比较中的最大值

```
import numpy as np

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

np.minimum(x1, x2)

- 返回x1和x2逐个元素比较中的最小值

```
import numpy as np

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

np.argmax(a, axis=None)

- 返回沿轴的最大值的索引

```
import numpy as np

a = np.arange(6).reshape(2, 3) + 10
print(a)

# 没有指定轴，则数组扁平化处理
print(np.argmax(a))

print(np.argmax(a, axis=0))
print(np.argmax(a, axis=1))
```

np.argmin(a, axis=None)

- 返回沿轴的最小值的索引

```
import numpy as np

a = np.arange(6).reshape(2, 3) + 10
print(a)

# 没有指定轴，则数组扁平化处理
print(np.argmin(a))

print(np.argmin(a, axis=0))
print(np.argmin(a, axis=1))
```

`np.random.normal(loc=0.0, scale=1.0, size=None)`

- `loc`: 均值（中心）
- `scale`: 标准差
- `size`: 输出的形状
- 返回从正态分布中抽取的随机样本

```
import numpy as np

print(np.random.normal(3, 2.5, size=(2, 4)))
```

`np.random.randn(d0, d1, ..., dn)`

- 返回从标准正态分布中抽取的随机样本

```
import numpy as np

print(np.random.randn()) # 没有指定参数返回一个数
print(np.random.randn(2, 3))
```


`np.random.rand(d0, d1, ..., dn)`

- 返回从 $[0, 1)$ 均匀分布中抽取的给定形状随机样本

```
import numpy as np

print(np.random.rand()) # 没有指定参数返回一个数
print(np.random.rand(2, 3))
```

`np.random.randint(low, high=None, size=None)`

- 返回从 $[low, high)$ 离散均匀分布中抽取的随机整数

```
import numpy as np

print(np.random.randint(2, size=10)) # 等价于下一行
print(np.random.randint(0, 2, size=10)) # 等价于上一行
print(np.random.randint(1, 4, size=(2, 3)))
```

`np.random.uniform(low=0.0, high=1.0, size=None)`

- 返回从 $[low, high)$ 均匀分布中抽取的随机样本

```
import numpy as np

print(np.random.uniform(2, size=10)) # 等价于下一行
print(np.random.uniform(0, 2, size=10)) # 等价于上一行
print(np.random.uniform(1, 4, size=(2, 3)))
```

`np.random.permutation(x)`

- `x`: int or array_like
- 如果 `x` 是整数，返回随机排列的 `np.arange(x)`

- 如果 x 是数组，只对数组的第一个维度随机排列，返回新的数组

```
import numpy as np

print(np.random.permutation(6))

arr1 = np.array([0, 1, 2, 3, 4, 5])
print(np.random.permutation(arr1))

arr2 = np.arange(10).reshape(5, 2)
print(np.random.permutation(arr2))
```

np.random.seed(x)

- 随机数种子

```
import numpy as np

np.random.seed(3)
print(np.random.uniform(1, 2, size=4))

np.random.seed(5)
print(np.random.uniform(1, 2, size=4))

np.random.seed(3)
print(np.random.uniform(1, 2, size=4))

np.random.seed()
print(np.random.uniform(1, 2, size=4))
```