NumPy

1 NumPy简介

NumPy（Numerical Python）是 Python 的一个扩展程序库，支持大量的维度数组与矩阵运算，此外也针对数组运算提供了大量的数学函数库。

NumPy 是一个运行速度非常快的数学库，主要用于数组计算。

NumPy 的优点：

1. 对于同样的数值计算任务，使用 NumPy 要比直接编写 Python 代码更加便捷。
2. NumPy 中的数组的存储效率和输入输出性能均远远优于 Python 中等价的基本数据结构。
3. NumPy 中的大部分代码都是用 C 语言写的，其底层算法在设计时就有着优异的性能，这使得 NumPy 比纯 Python 代码高效得多。

2 NumPy数据类型

数据类型	说明
`bool_`	布尔型，取值为 `True` 或 `False`
`int_`	默认的整数类型，类似于 C 语言中的 `long`、`int32` 或 `int64`
`intc`	与 C 语言的 `int` 类型一样，一般是 `int32` 或 `int64`
`intp`	用于索引的整数类型，类似于 C 语言的 `ssize_t`， 一般情况下是 `int32` 或 `int64`
`int8`	字节，-128~127
`int16`	双字节整数，-32768~32767
`int32`	四字节整数，-2147483648~2147483647
`int64`	八字节整数，-9223372036854775808~9223372036854775807
`uint8`	无符号单字节整数，0~255
`uint16`	无符号双字节整数，0~65535
`uint32`	无符号四字节整数，0~4294967295
`uint64`	无符号八字节整数，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

数据类型	说明
`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
`complex64`	64 位复数，实部和虚部都为 32 位浮点数
`complex128`	128 位复数，实部和虚部都为 64 位浮点数

每个内建类型都有一个唯一的字符代码，如下表所示。

字符代码	类型
b	布尔型
i	有符号整型 `'i1'` 表示 `int8`，`'i2'` 表示 `int16`，`'i4'` 表示 `int32`，`'i8'` 表示 `int64`
u	无符号整型
f	浮点型
c	复数浮点型
m	`timedelta`（时间间隔）
M	`datetime`（日期时间）
O	Python 对象
S 或 a	字符串
U	Unicode
V	原始数据（`void`）

数据类型对象用来描述与数组对应的内存区域如何使用，这依赖于如下几个方面：

1. 数据的类型。
2. 数据的字节长度。
3. 数据的字节顺序（小端法或大端法）。
4. 在结构化类型的情况下，字段的名称、每个字段的数据类型和每个字段所取的内存块的部分。
5. 如果数据类型是子数组，它的形状和数据类型。

NumPy 的数据类型实际上是 `dtype` 类的实例，使用 `numpy.dtype()` 来创建数据类型对象，参数为：

1. `dtype`：要转换为的数据类型对象。

2. `align`：接收布尔类型参数，默认值为`False`。如果为`True`，填充字段使其类似于 C 语言的结构体。
3. `copy`：接收布尔类型参数，默认值为`False`。如果为`True`，则复制`dtype`对象；如果为`False`，则是对内置数据类型对象的引用。

```
import numpy as np

dt = np.dtype(np.int32)
print(dt)  # int32
print(type(dt))  # <class 'numpy.dtype[int32]'>
```

可以用字符代码字符串来表示类型，同时可以指定字节顺序，`<`表示小端法，`>`表示大端法。例如：

```
import numpy as np

dt = np.dtype('<i4')  # int32, 小端法
print(dt)  # int32
print(type(dt))  # <class 'numpy.dtype[int32]'>
```

要创建结构化数据类型（类似于结构体），可以传递一个列表，列表中每个元素为一个元组，表示一个字段，元组的第一个元素为字段名，第二个元素为字段类型。例如：

```
import numpy as np

student_type = np.dtype([('name', 'S20'), ('age', 'i4')])  # 'S20' 表示长度为 20 的字符串
print(student_type)  # [('name', 'S20'), ('age', '<i4')]
print(type(student_type))  # <class 'numpy.dtype[void]'>
```

3 `Ndarray`数组

3.1 `Ndarray`对象简介

`Ndarray`对象是用于存放同类型元素的多维数组。`Ndarray`数组中的每个元素在内存中都有相同大小的存储区域。

`Ndarray`对象的组成：

1. 一个指向数据的指针。
2. 数据类型，描述在数组中的固定大小值的格子。
3. 一个表示数组形状的元组。

4. 一个跨度元组，其中的整数指的是为了前进到当前维度下一个元素需要跨过的字节数。

3.2 `Ndarray` 数组的创建

`Ndarray` 数组中所有元素的类型是相同的。如果传入的数据中包含不同的类型，则统一为同一类型，优先级为 `str > float > int` 。

`numpy.array()` 函数用于创建 `Ndarray` 数组，参数为：

1. `object` ： `Ndarray` 数组，或嵌套的序列。
2. `dtype` ： 数组元素的数据类型。可选，默认值为 `None` ，表示由输入的数据决定。
3. `copy` ： 对象是否需要复制。布尔值，可选，默认值为 `True` 。
4. `order` ： 元素在内存中的存储方式。 `'C'` 为行优先， `'F'` 为列优先， `'A'` 为任意顺序， `'K'` 为输入顺序。可选，默认值为 `'K'` 。
5. `subok` ： 默认返回一个与基类类型一致的数组。
6. `ndmin` ： 指定生成数组的最小维度。可选，默认值为 0。

```
import numpy as np

# 一维数组
array = np.array([1, 2, 3, 4, 5])
print(array)  # [1 2 3 4 5]

# 多维数组
array = np.array([[1, 2, 3], [4, 5, 6]])
print(array)
# [[1 2 3]
#  [4 5 6]]

# ndmin 参数
array = np.array([1, 2, 3, 4, 5, 6], ndmin=2)
print(array)  # [[1 2 3 4 5 6]]

# dtype 参数
array = np.array([1, 2, 3, 4, 5, 6], dtype='f')
print(array)  # [1. 2. 3. 4. 5. 6.]

# 结构化数据类型
student_type = np.dtype([('name', 'S20'), ('age', 'i4')])
array = np.array([('Jack', 18), ('Mary', 20)], dtype=student_type)
print(array)  # [(b'Jack', 18) (b'Mary', 20)]
```

`numpy.asarray()` 函数也可以创建 `Ndarray` 数组，参数为：

1. `a` ： 任意形式的参数，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。

2. `dtype`：数组元素的数据类型。可选，默认值为`None`，表示由输入的数据决定。
3. `order`：元素在内存中的存储方式。可能的取值包括`'C'`、`'F'`、`'A'`、`'K'`。可选，默认值为`'K'`。

`numpy.empty()`函数可以创建指定形状且未初始化的数组，参数为：

1. `shape`：数组的形状。可以是整数、元组或列表。
2. `dtype`：数据类型。可选，默认值为`float`。
3. `order`：元素在内存中的存储方式。可能的取值包括`'C'`、`'F'`。可选，默认值为`'C'`。

```
import numpy as np

# 一维数组
arr = np.empty(5, dtype=int)
print(arr)  # [1166004928      32760 1166009440      32760    6881350]

# 多维数组
arr = np.empty([3, 2], dtype=int)
print(arr)
# [[1285870272      32760]
#  [1285874784      32760]
#  [   5439567         69]]
```

`numpy.zeros()`函数可以创建指定形状的数组，并且数组元素初始化为 0。参数与`numpy.empty()`函数相同。

`numpy.ones()`函数可以创建指定形状的数组，并且数组元素初始化为 1。参数与`numpy.empty()`函数相同。

`numpy.full()`函数可以创建指定形状的数组，并且用给定的值初始化数组元素。参数为：

1. `shape`：数组的形状。
2. `full_value`：用于初始化的数据。
3. `dtype`：数据类型。可选，默认值为`None`，表示由输入的数据决定。
4. `order`：元素在内存中的存储方式。可能的取值包括`'C'`、`'F'`。可选，默认值为`'C'`。

```
import numpy as np

arr = np.full(5, 3)
print(arr)  # [3 3 3 3 3]
```

`numpy.eye()`函数用于创建二维数组，并且将对角线元素置为 1，其他元素为 0。参数为：

1. `N`：行数。
2. `M`：列数。可选，默认与行数相同。

3. `k`：对角线的索引。可选，默认值为 0，表示主对角线。若为正数，表示主对角线上方；若为负数，表示主对角线下方。
4. `dtype`：数据类型。可选，默认值为`float`。
5. `order`：元素在内存中的存储方式。可能的取值包括`'C'`、`'F'`。可选，默认值为`'C'`。

```
import numpy as np

arr = np.eye(5)  # 创建 5 阶单位矩阵
print(arr)
# [[1. 0. 0. 0. 0.]
#  [0. 1. 0. 0. 0.]
#  [0. 0. 1. 0. 0.]
#  [0. 0. 0. 1. 0.]
#  [0. 0. 0. 0. 1.]]

arr = np.eye(5, k=1)  # 将主对角线上方 1 格的对角线元素置为 1
print(arr)
# [[0. 1. 0. 0. 0.]
#  [0. 0. 1. 0. 0.]
#  [0. 0. 0. 1. 0.]
#  [0. 0. 0. 0. 1.]
#  [0. 0. 0. 0. 0.]]
```

`numpy.arange()`函数根据给定的范围和步长创建一维数组。参数为：

1. `start`：起始值。可选，默认值为 0。
2. `stop`：终止值（不包括）。
3. `step`：步长。可选，默认值为 1。如果`step`参数为位置参数，则必须指定`start`参数。
4. `dtype`：数据类型。可选，默认值为`None`，表示由输入的数据决定。

```
import numpy as np

arr = np.arange(10)  # 起始值默认为 0，终止值为 10
print(arr)  # [0 1 2 3 4 5 6 7 8 9]

arr = np.arange(1, 10)  # 起始值为 1，终止值为 10
print(arr)  # [1 2 3 4 5 6 7 8 9]

arr = np.arange(1, 10, 2)  # 起始值为 1，终止值为 10，步长为 2
print(arr)  # [1 3 5 7 9]

arr = np.arange(10, step=2) # 起始值默认为 0，终止值为 10，步长为 2
print(arr)  # [0 2 4 6 8]
```

`numpy.frombuffer()`函数将其他对象转化成`Ndarray`数组。参数为：

1. `buffer`：可以是任意对象，会以流的形式读入。

2. `dtype`：数据类型。可选，默认值为`float`。
3. `count`：读取数据的数量。可选，默认值为 -1，读取所有数据。
4. `offset`：读取的起始位置。可选，默认值为 0。

注意：当`buffer`参数为字符串时，Python 3 默认`str`类型为 Unicode 串，需要在字符串前面添加字符`b`转化成字节流。

```
import numpy as np

s = b'hello world'

arr = np.frombuffer(s, dtype='S1')  # 读取所有字符
print(arr)  # [b'h' b'e' b'l' b'l' b'o' b' ' b'w' b'o' b'r' b'l' b'd']

arr = np.frombuffer(s, dtype='S1', count=5)  # 读取 5 个字符
print(arr)  # [b'h' b'e' b'l' b'l' b'o']

arr = np.frombuffer(s, dtype='S1', count=5, offset=3)  # 从索引 3 开始，读取 5 个字符
print(arr)  # [b'l' b'o' b' ' b'w' b'o']
```

`numpy.fromiter()`函数从可迭代对象中读取数据，创建一维数组。参数为：

1. `iterable`：可迭代对象。
2. `dtype`：数据类型。
3. `count`：读取数据的数量。可选，默认值为 -1，读取所有数据。

```
import numpy as np

l = [1, 2, 3, 4, 5]
arr = np.fromiter(l, dtype=int)
print(arr)  # [1 2 3 4 5]
```

`numpy.linspace()`函数用于创建等差数列。参数为：

1. `start`：起始值。
2. `stop`：终止值。
3. `num`：元素个数。可选，默认值为 50。
4. `endpoint`：是否包含终止值，为`True`则包含，为`False`则不包含。可选，默认为`True`。
5. `retstep`：是否显示间距，为`True`则显示，为`False`则不显示。可选，默认为`False`。
6. `dtype`：数据类型。可选。默认值为`None`，根据`start`和`stop`推断，此时不会推断出`int`类型，即使可以产生`int`类型的数组，也会用`float`类型。

```python
import numpy as np

arr = np.linspace(1, 10, 5)
print(arr)  # [ 1.    3.25  5.5   7.75 10.  ]

arr = np.linspace(1, 10, 5, endpoint=False)
print(arr)  # [1.  2.8 4.6 6.4 8.2]

arr = np.linspace(1, 10, 5, retstep=True)
print(arr)  # (array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ]), 2.25)
```

`numpy.logspace()` 函数用于创建等比数列。参数为：

1. `start`：起始值为 `base ** start`。
2. `stop`：终止值为 `base ** stop`。
3. `num`：元素个数。可选，默认值为 50。
4. `endpoint`：是否包含终止值，为 `True` 则包含，为 `False` 则不包含。可选，默认为 `True`。
5. `base`：底数。可选，默认值为 10.0。
6. `dtype`：数据类型。可选，默认值为 `None`，由 `start` 和 `stop` 决定。

```python
import numpy as np

arr = np.logspace(1, 4, 4)
print(arr)  # [   10.   100.  1000. 10000.]

arr = np.logspace(1, 10, 10, base=2)
print(arr)  # [   2.    4.    8.   16.   32.   64.  128.  256.  512. 1024.]
```

`numpy.random.rand()` 函数用于生成 $[0, 1)$ 区间内的随机数，参数为数组各维度的大小。例如：

```python
import numpy as np

arr = np.random.rand()  # 生成一个随机数
print(arr)  # 0.46223118949331554

arr = np.random.rand(3)  # 生成一个大小为 3 的一维数组
print(arr)  # [0.10185733 0.19878431 0.46851335]

arr = np.random.rand(3, 2)  # 生成一个 3 行 2 列的二维数组
print(arr)
# [[0.59310082 0.60284131]
#  [0.8154639  0.27099303]
#  [0.75609066 0.12476809]]
```

`numpy.random.random()` 函数用于生成 $[0, 1)$ 区间内的随机数，参数为元素个数。例如：

```python
import numpy as np

arr = np.random.random()  # 生成一个随机数
print(arr)  # 0.742250685821003

arr = np.random.random(3)  # 生成一个大小为 3 的一维数组
print(arr)  # [0.41693625 0.27771851 0.83707164]
```

`numpy.random.randint()`函数用于生成指定范围的随机数。参数为：

1. `low`：下界。
2. `high`：上界（不包含）。可选。默认值为`None`，此时范围为`[0, low)`。
3. `size`：数组的形状。可选。默认值为`None`，表示生成 1 个随机数。
4. `dtype`：元素类型。可选，默认值为`int`。

```python
import numpy as np

arr = np.random.randint(10)  # 范围为 [0, 10)
print(arr)  # 5

arr = np.random.randint(1, 10)  # 范围为 [1, 10)
print(arr)  # 8

arr = np.random.randint(1, 10, 3)  # 生成大小为 3 的一维数组
print(arr)  # [7 5 2]

arr = np.random.randint(1, 10, (3, 2))  # 生成 3 行 2 列的二维数组
print(arr)
# [[4 8]
#  [1 4]
#  [5 3]]
```

`numpy.random.randn()`函数生成一组服从标准正态分布的样本，参数为数组各维度的大小。例如：

```python
import numpy as np

arr = np.random.randn()  # 生成一个数
print(arr)  # -0.3512395770623732

arr = np.random.randn(5)  # 生成一个大小为 5 的一维数组
print(arr)  # [ 1.14616959 -0.4599372   0.90461256 -0.08188299  0.63645179]

arr = np.random.randn(3, 2)  # 生成一个 3 行 2 列的二维数组
print(arr)
# [[ 1.17801871 -0.06740504]
#  [ 0.15363981 -0.08419718]
#  [-0.65597993  0.30165878]]
```

`numpy.random.normal()`函数生成一组服从正态分布的样本。参数为：

1. `loc`：正太分布的均值，浮点型。可选，默认值为 0.0。
2. `scale`：正态分布的标准差，浮点型。可选，默认值为 1.0。
3. `size`：数组的形状。可选。默认值为`None`，表示生成一个数。

3.3 `Ndarray`数组的属性

`Ndarray`数组的维数称为**秩**（rank）。一维数组的秩为 1，二维数组的秩为 2，以此类推。

在 Numpy 中，每一个线性的数组称为一个轴（axis），也就是维度（dimension）。例如，二维数组相当于一维数组的一维数组，数组本身是一个一维数组，其中每个元素又是一个一维数组，因此外层数组是第一个轴，内层数组是第二个轴。轴的数量与维数相同。

很多函数会用到`axis`参数，用于指定操作的方向。取值为 0 表示沿着第 0 轴进行操作，即对每一列进行操作；取值为 1 表示沿着第 1 轴进行操作，即对每一行进行操作。

`Ndarray`数组的常用属性：

属性	说明
`ndim`	秩，即轴的数量或维度的数量
`shape`	数组的形状，表示为元组
`size`	数组元素的总个数
`dtype`	数组元素的数据类型
`itemsize`	每个数组元素的大小，单位为字节
`flags`	`Ndarray`对象的内存信息
`real`	数组元素的实部
`imag`	数组元素的虚部
`data`	包含实际数组元素的缓冲区

`shape`属性是可以修改的，例如：

```python
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr.shape)  # (2, 3)

arr.shape = (3, 2)
print(arr)
# [[1 2]
#  [3 4]
#  [5 6]]
```

`flags`属性中的所有信息都是布尔类型，具体包含以下内容：

值	说明
`C_CONTIGUOUS` （C）	数据是否在一个单一的 C 风格的连续段中（行优先）
`F_CONTIGUOUS` （F）	数据是否在一个单一的 Fortran 风格的连续段中（列有限）
`OWNDATA` （O）	数组拥有它所使用的内存，还是从另一个对象借用
`WRITEABLE` （W）	数据区域是否可被写入，为`True`则可以写入，为`False`则只读
`ALIGNED` （A）	数据和所有元素是否都适当地对齐到硬件上
`UPDATEIFCOPY` （U）	这个数组是其他数组的一个副本，当这个数组被释放时， 原数组的内容将被更新

3.4 `Ndarray`数组元素的操作

3.4.1 索引和切片

`Ndarray`数组可以通过索引和切片来访问数组元素，与列表的索引和切片操作一样。索引从 0 开始，用`[]`括起来跟在数组名的后面。

`x[exp1, exp2, ..., expN]`等价于`x[(exp1, exp2, ..., expN)]`，此时在`[]`中用的是元组，执行基本索引操作。

基本切片生成的所有数组始终是原始数组的视图，而不会复制原始数组。也就是说，基本切片生成的数组仍然指向原始数组对象，而没有创建新对象。

索引和切片操作也适用于多维数组，多个维度之间用逗号隔开。例如：

```python
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])

print(arr)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

print(arr[2:])
# [[ 7  8  9]
#  [10 11 12]]

print(arr[2:, 1:])
# [[ 8  9]
#  [11 12]]
```

可以在切片中使用`...`，表示选择相应维度的所有内容。例如：

```python
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])

print(arr)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

# 所有行、索引为 1 及以后的列
print(arr[..., 1:])
# [[ 2  3]
#  [ 5  6]
#  [ 8  9]
#  [11 12]]

# 所有行、索引为 1 的列
print(arr[..., 1])  # [ 2  5  8 11]

# 索引为 2 的行、所有列
print(arr[2, ...])  # [7 8 9]
```

3.4.2 高级索引

如果在`[]`中用的是非元组的序列对象，或者`Ndarray`对象（元素类型为`int`或`bool`），或者是至少包含一个序列对象或`Ndarray`对象的元组，则会触发高级索引。

高级索引有两种类型：整数数组索引、布尔数组索引。

高级索引始终返回数组的副本。

3.4.2.1 整数数组索引

整数数组索引可以选择数组中的任意项，每个整数数组表示该维度的若干索引，多个维度之间用逗号隔开。例如：

```
import numpy as np

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

# 选择索引为 3, 2, 7, -1 的元素
print(arr[np.array([3, 2, 7, -1])])  # [3 2 7 9]

# 用列表进行高级索引，效果相同
print(arr[[3, 2, 7, -1]])  # [3 2 7 9]
```

对多维数组进行整数数组索引时，如果索引数组的形状相同，并且被索引数组的每个维度都有一个索引数组，则结果数组与索引数组形状相同，并且结果数组中的每个元素对应于索引数组中每个位置的索引集。

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(arr)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

# 分别取得坐标为 (0, 2), (1, 1), (2, 0), (3, 1) 的元素，结果数组是长度为 4 的一维数组
print(arr[[0, 1, 2, 3], [2, 1, 0, 1]])  # [ 3  5  7 11]

# 分别取得坐标为 (0, 0), (0, 2), (3, 0), (3, 2) 的元素，结果数组是 2*2 的二维数组
rows = [[0, 0], [3, 3]]
cols = [[0, 2], [0, 2]]
print(arr[rows, cols])
# [[ 1  3]
#  [10 12]]
```

如果索引数组的形状不同，则会尝试将它们广播到相同的形状。如果无法广播到相同的形状，则抛出`IndexError`异常。

广播机制允许索引数组与标量值结合使用，效果是将标量值作用于索引数组的所有对应值。例如：

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(arr)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

# 相当于 arr[[3, 1, 2], [1, 1, 1]]
print(arr[[3, 1, 2], 1])  # [11  5  8]
```

当索引数组的数量小于被索引数组的维度时，将对被索引数组进行部分索引。例如：

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(arr)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

# 只有一个索引数组，索引数组中的每个元素选择被索引数组中的一行
# 分别选择下标为 3、1、2 的行
print(arr[[3, 1, 2]])
# [[10 11 12]
#  [ 4  5  6]
#  [ 7  8  9]]
```

通常，结果数组的形状将是索引数组的形状（或所有索引数组广播到的形状）与被索引数组中任何未使用维度（未索引的维度）的形状的级联。例如，在上面的例子中，结果数组的行数为索引数组的大小，结果数组的列数为被索引数组的列数，而被索引数组的列没有被索引。

3.4.2.2 布尔数组索引

当`[]`中用的是布尔数组时会发生布尔数组索引。布尔数组索引通过布尔运算来获取符合指定条件的元素。

设被索引数组为`x`，布尔数组为`obj`，当`x`与`obj`的维数相同时，`x[obj]`返回一个一维数组，其中的元素为与`obj`中`True`值的位置对应的`x`中的元素。搜索顺序为行优先。如果`obj`在`x`的索引范围之外有`True`值，则会抛出`IndexError`。如果`obj`小于`x`，相当于用`False`填充空余位置。例如：

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(arr)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

# 选择大于 5 的元素
print(arr[arr > 5])  # [ 6  7  8  9 10 11 12]
```

```
import numpy as np

arr = np.array([[1, np.nan, 3], [np.nan, 5, np.nan], [7, 8, np.nan], [np.nan, 11, 12]])
print(arr)
# [[ 1. nan  3.]
#  [nan  5. nan]
#  [ 7.  8. nan]
#  [nan 11. 12.]]

# 去除数组中所有的 nan
print(arr[~np.isnan(arr)])  # [ 1.  3.  5.  7.  8. 11. 12.]
```

可以对满足条件的元素执行某种操作，例如：

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(arr)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

# 将所有大于 5 的元素加 10
arr[arr > 5] += 10
print(arr)
# [[ 1  2  3]
#  [ 4  5 16]
#  [17 18 19]
#  [20 21 22]]
```

当布尔数组的维数小于被索引数组的维数时，将在缺失的维度上自动使用`:`，选中该维度的所有内容。例如：

```python
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(arr)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

b = arr > 5
print(b)
# [[False False False]
#  [False False  True]
#  [ True  True  True]
#  [ True  True  True]]

print(b[:, 1])  # [False False  True  True]

# 布尔数组是一维，被索引数组是二维
# 布尔数组作用在被索引数组的第 0 维，被索引数组的第 1 维全部选中
print(arr[b[:, 1]])
# [[ 7  8  9]
#  [10 11 12]]
```

3.4.2.3 高级索引与切片结合使用

当一个高级索引与一个切片相结合时，切片操作与高级索引相互独立。例如：

```python
import numpy as np

arr = np.arange(35).reshape(5, 7)
print(arr)
# [[ 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 32 33 34]]

# 选择索引为 3、1、2 的行，以及索引为 1-3 的列
print(arr[[3, 1, 2], 1:4])
# [[22 23 24]
#  [ 8  9 10]
#  [15 16 17]]
```

当存在多个切片时，可以将其中的一个替换为高级索引，并使结果保持不变，但此时的结果数组是原数组的副本，可能具有不同的内存布局。因此一般直接使用切片，避免用高级索引替换切片。

3.4.3 广播

广播（broadcast）是 NumPy 对不同形状的数组进行数值计算的方式。

如果两个数组形状相同，则对两个数组做算术运算时，结果是将两个数组对应位置的元素做运算。例如：

```
import numpy as np

a = np.array([1, 2, 3, 4])
b = np.array([10, 20, 30, 40])
print(a + b)  # [11 22 33 44]
```

如果两个数组形状不同，则会将较小的数组广播到较大数组的大小，使得它们的形状可以兼容。例如：

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([10, 20, 30])

# a 的形状为 (2, 3)，b 的形状为 (3, )
# 将 b 广播为 (2, 3)，也就是将第一行复制若干次，使 b 的形状与 a 相同
print(a + b)
# [[11 22 33]
#  [14 25 36]]
```

广播的规则：

1. 让所有输入数组都向其中形状元组最长（即维度最高）的数组看齐，形状元组中不足的部分通过在前面加 1 补齐。
2. 输出数组的形状是输入数组形状的各个维度上的最大值。
3. 如果输入数组的某个维度和输出数组的对应维度的长度相同或者其长度为 1 时，这个数组能够用来计算，否则报错。
4. 当输入数组的某个维度的长度为 1 时，沿着此维度运算时都用此维度上的第一组值。

简言之，对两个数组进行运算时，将从右向左依次比较它们的每一个维度。如果当前维度长度相同，或者其中有一个为 1，则当前维度是兼容的；如果条件不满足，则抛出`ValueError: operands could not be broadcast together`异常。

3.4.4 迭代

`numpy.nditer`是一个多维迭代器对象，可以在数组上进行迭代。数组的每个元素可以使用 Python 的标准`Iterator`接口来访问。例如：

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(arr)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

for x in np.nditer(arr):
    print(x, end=', ')
# 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
```

迭代时默认按照内存中的存储顺序访问元素。设数组为`a`，它的转置为`a.T`，它们在内存中的存储顺序是相同的，因此遍历顺序也是一样的。但是`a.T.copy(order='C')`的遍历结果是不同的，因为它和前两种的存储方式不同，默认按行访问。例如：

```python
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(a)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

for x in np.nditer(a):
    print(x, end=', ')
print()
# 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,

print(a.T)
# [[ 1  4  7 10]
#  [ 2  5  8 11]
#  [ 3  6  9 12]]

for x in np.nditer(a.T):
    print(x, end=', ')
print()
# 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,

b = a.T.copy(order='C')
print(b)
# [[ 1  4  7 10]
#  [ 2  5  8 11]
#  [ 3  6  9 12]]

for x in np.nditer(b):
    print(x, end=', ')
print()
# 1, 4, 7, 10, 2, 5, 8, 11, 3, 6, 9, 12,
```

可以使用`order`参数指定访问顺序，`'F'`表示列优先，`'C'`表示行优先，默认值为`'C'`。例如：

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(a)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

for x in np.nditer(a, order='C'):
    print(x, end=', ')
print()
# 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,

for x in np.nditer(a, order='F'):
    print(x, end=', ')
print()
# 1, 4, 7, 10, 2, 5, 8, 11, 3, 6, 9, 12,
```

默认情况下，迭代过程中访问到的元素是只读的，不能修改。可以通过`op_flags`参数设置访问权限，它是一个列表，当参数值为`['readwrite']`时即可读写数组元素。例如：

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(a)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

for x in np.nditer(a, op_flags=['readwrite']):
    x[...] = x * 2

print(a)
# [[ 2  4  6]
#  [ 8 10 12]
#  [14 16 18]
#  [20 22 24]]
```

`flags`参数是一个列表，可以接受下列值：

- `c_index`：可以跟踪 C 顺序的索引。
- `f_index`：可以跟踪 F 顺序的索引。
- `multi-index`：每次迭代可以跟踪一种索引类型。
- `external_loop`：给出的值是具有多个值的一维数组，而不是零维数组。

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(a)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

for x in np.nditer(a, flags=['external_loop']):
    print(x, end=', ')
print()
# [ 1  2  3  4  5  6  7  8  9 10 11 12],

for x in np.nditer(a, flags=['external_loop'], order='F'):
    print(x, end=', ')
print()
# [ 1  4  7 10], [ 2  5  8 11], [ 3  6  9 12],
```

如果两个数组是可广播的，`nditer`组合对象能够同时迭代它们，并将较小的数组广播到较大的形状。例如：

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
b = np.array([-1, -2, -3])

for x, y in np.nditer([a, b]):
    print('%d:%d' % (x, y), end=', ')
# 1:-1, 2:-2, 3:-3, 4:-1, 5:-2, 6:-3, 7:-1, 8:-2, 9:-3, 10:-1, 11:-2, 12:-3,
```

`flat`属性是一个数组元素迭代器，也可以用来对数组进行迭代。例如：

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(a)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

for x in a.flat:
    print(x, end=', ')
# 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
```

3.5 `Ndarray`数组的操作

3.5.1 修改数组形状

`reshape()` 方法在不改变元素的条件下修改数组的形状，参数为：

1. `shape`：形状
2. `order`：元素在内存中的存储顺序。`'C'` 表示行优先，`'F'` 表示列优先，`'A'` 表示原顺序，`'K'` 表示按原数组在内存中的存储顺序。

```
import numpy as np

a = np.arange(12)
print(a)  # [ 0  1  2  3  4  5  6  7  8  9 10 11]

b = a.reshape(3, 4)
print(b)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]]

c = a.reshape((3, 4), order='F')
print(c)
# [[ 0  3  6  9]
#  [ 1  4  7 10]
#  [ 2  5  8 11]]
```

`flatten()` 方法将数组展开成一维数组，返回新数组。新数组有独立的存储空间，修改新数组不会影响原数组。它只有一个 `order` 参数，可以接受的值有 `'C'`、`'F'`、`'A'`、`'K'`，默认值为 `'C'`。例如：

```
import numpy as np

a = np.arange(12).reshape(3, 4)
print(a)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]]

b = a.flatten()
print(b)  # [ 0  1  2  3  4  5  6  7  8  9 10 11]

c = a.flatten(order='F')
print(c)  # [ 0  4  8  1  5  9  2  6 10  3  7 11]
```

`ravel()` 方法也可以将数组展开成一维数组，用法与 `flatten()` 方法相同。不同在于，`ravel()` 方法按 C、A 或 K 顺序展开时，返回数组视图，返回的数组与原数组共用同一片存储空间，对返回的数组进行修改时会影响原数组；按 F 顺序展开时，返回数组的副本，修改新数组不会影响原数组。例如：

```
import numpy as np

a = np.arange(12).reshape(3, 4)
print(a)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]]

b = a.ravel()
print(b)  # [ 0  1  2  3  4  5  6  7  8  9 10 11]

b[0] = 100
print(b)  # [100   1   2   3   4   5   6   7   8   9  10  11]
print(a)
# [[100   1   2   3]
#  [  4   5   6   7]
#  [  8   9  10  11]]

c = a.ravel(order='F')
print(c)  # [100   4   8   1   5   9   2   6  10   3   7  11]

c[0] = 200
print(c)  # [200   4   8   1   5   9   2   6  10   3   7  11]
print(a)
# [[100   1   2   3]
#  [  4   5   6   7]
#  [  8   9  10  11]]
```

3.5.2 翻转数组

`numpy.transpose()`函数用于对换数组的维度，返回数组视图，修改返回的数组会影响原始数组。参数为：

1. `a`：要操作的数组。
2. `axes`：整数列表，指定要对换的维度。默认将所有维度进行对换。

```
import numpy as np

a = np.arange(12).reshape(3, 4)
print(a)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]]

b = np.transpose(a)
print(b)
# [[ 0  4  8]
#  [ 1  5  9]
#  [ 2  6 10]
#  [ 3  7 11]]

b[0][0] = 100
print(b)
# [[100   4   8]
#  [  1   5   9]
#  [  2   6  10]
#  [  3   7  11]]

print(a)
# [[100   1   2   3]
#  [  4   5   6   7]
#  [  8   9  10  11]]
```

`T`属性也可以对数组进行转置，效果与`numpy.transpose()`函数相同。

`numpy.rollaxis()`函数将特定的轴向后滚动到特定的位置。参数为：

1. `a`：要操作的数组。
2. `axis`：要向后滚动的轴。其他轴的相对位置不会改变。
3. `start`：滚动的目标位置。可选，默认值为 0。

```python
import numpy as np

a = np.arange(24).reshape(2, 3, 4)
print(a)
# [[[ 0  1  2  3]
#   [ 4  5  6  7]
#   [ 8  9 10 11]]
#
#  [[12 13 14 15]
#   [16 17 18 19]
#   [20 21 22 23]]]

# 将 2 轴滚动到 0 轴的位置
b = np.rollaxis(a, 2)
print(b)
# [[[ 0  4  8]
#   [12 16 20]]
#
#  [[ 1  5  9]
#   [13 17 21]]
#
#  [[ 2  6 10]
#   [14 18 22]]
#
#  [[ 3  7 11]
#   [15 19 23]]]

print(b.shape)  # (4, 2, 3)

# 将 2 轴滚动到 1 轴的位置
c = np.rollaxis(a, 2, 1)
print(c)
# [[[ 0  4  8]
#   [ 1  5  9]
#   [ 2  6 10]
#   [ 3  7 11]]
#
#  [[12 16 20]
#   [13 17 21]
#   [14 18 22]
#   [15 19 23]]]

print(c.shape)  # (2, 4, 3)
```

`np.swapaxes()` 函数用于交换数组的两个轴。参数为：

1. `a`：要操作的数组
2. `axis1`：第一个轴
3. `axis2`：第二个轴

```python
import numpy as np

a = np.arange(24).reshape(2, 3, 4)
print(a)
# [[[ 0  1  2  3]
#   [ 4  5  6  7]
#   [ 8  9 10 11]]
#
#  [[12 13 14 15]
#   [16 17 18 19]
#   [20 21 22 23]]]

# 交换 0 轴和 2 轴
b = np.swapaxes(a, 0, 2)
print(b)
# [[[ 0 12]
#   [ 4 16]
#   [ 8 20]]
#
#  [[ 1 13]
#   [ 5 17]
#   [ 9 21]]
#
#  [[ 2 14]
#   [ 6 18]
#   [10 22]]
#
#  [[ 3 15]
#   [ 7 19]
#   [11 23]]]

print(b.shape)  # (4, 3, 2)
```

3.5.3 修改数组维度

`numpy.broadcast`类封装了将一个数组广播到另一个数组的结果，使用`numpy.broadcast()`来创建`broadcast`对象。用法如下：

```python
import numpy as np

x = np.array([[1], [2], [3]])
y = np.array([4, 5, 6])

b = np.broadcast(x, y)
print(b)  # <numpy.broadcast object at 0x000002A2A89F1E10>
print(b.shape)  # (3, 3)

# 迭代获取广播结果
for u, v in b:
    print(f'{u}:{v}', end=', ')
print()
# 1:4, 1:5, 1:6, 2:4, 2:5, 2:6, 3:4, 3:5, 3:6,

# 手动执行带广播的数组加法
b = np.broadcast(x, y)
z = np.empty(b.shape)
z.flat = [u + v for u, v in b]
print(z)
# [[5. 6. 7.]
#  [6. 7. 8.]
#  [7. 8. 9.]]
```

`numpy.broadcast_to()`函数将数组广播到指定的形状，返回数组的只读视图。如果新形状不符合广播规则，则抛出`ValueError`异常。参数为：

1. `array`：待广播的数组
2. `shape`：广播到的形状

```python
import numpy as np

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

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

`numpy.expand_dims()`函数通过在指定位置插入新的轴来扩展数组维度。参数为：

1. `arr`：输入数组
2. `axis`：新轴插入的位置

```python
import numpy as np

a = np.arange(4).reshape(2, 2)
print(a)
# [[0 1]
#  [2 3]]

print(a.shape)  # (2, 2)
print(a.ndim)  # 2

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

print(b.shape)  # (1, 2, 2)
print(b.ndim)  # 3
```

`numpy.squeeze()` 函数用于删除数组中的维度。参数为：

1. `arr`：输入数组
2. `axis`：要删除的维度。可选。不指定此参数时，默认删除所有长度为 1 的维度；指定此参数时，若该维度的长度为 1 则删除该维度，否则抛出 `ValueError` 异常。

```python
import numpy as np

a = np.arange(9).reshape(1, 3, 1, 3)
print(a)
# [[[[0 1 2]]
#
#   [[3 4 5]]
#
#   [[6 7 8]]]]

print(a.shape)  # (1, 3, 1, 3)
print(a.ndim)  # 4

# 删除所有长度为 1 的维度
b = np.squeeze(a)
print(b)
# [[0 1 2]
#  [3 4 5]
#  [6 7 8]]

print(b.shape)  # (3, 3)
print(b.ndim)  # 2
```

```python
import numpy as np

a = np.arange(9).reshape(1, 3, 1, 3)
print(a)
# [[[[0 1 2]]
#
#   [[3 4 5]]
#
#   [[6 7 8]]]]

print(a.shape)  # (1, 3, 1, 3)
print(a.ndim)  # 4

# 删除 2 轴
b = np.squeeze(a, 2)
print(b)
# [[[0 1 2]
#   [3 4 5]
#   [6 7 8]]]

print(b.shape)  # (1, 3, 3)
print(b.ndim)  # 3
```

3.5.4 连接数组

`numpy.concatenate()` 函数用于沿指定轴连接相同形状的两个或多个数组。参数为：

1. `arrays`：形状相同的数组序列。
2. `axis`：用于连接的轴。默认值为 0。

```python
import numpy as np

a = np.arange(4).reshape(2, 2)
print(a)
# [[0 1]
#  [2 3]]

b = np.arange(5, 9).reshape(2, 2)
print(b)
# [[5 6]
#  [7 8]]

# 沿 0 轴连接两个数组
c = np.concatenate((a, b))
print(c)
# [[0 1]
#  [2 3]
#  [5 6]
#  [7 8]]

# 沿 1 轴连接两个数组
d = np.concatenate((a, b), 1)
print(d)
# [[0 1 5 6]
#  [2 3 7 8]]
```

`numpy.stack()` 函数用于沿新轴堆叠数组序列。参数为：

1. `arrays`：形状相同的数组序列。
2. `axis`：数组中的轴，输入数组沿着它来堆叠。

```
import numpy as np

a = np.arange(4).reshape(2, 2)
print(a)
# [[0 1]
#  [2 3]]

b = np.arange(5, 9).reshape(2, 2)
print(b)
# [[5 6]
#  [7 8]]

c = np.stack((a, b), 0)
print(c)
# [[[0 1]
#   [2 3]]
#
#  [[5 6]
#   [7 8]]]

d = np.stack((a, b), 1)
print(d)
# [[[0 1]
#   [5 6]]
#
#  [[2 3]
#   [7 8]]]
```

`numpy.hstack()` 函数对数组序列进行水平堆叠，`numpy.vstack()` 函数对数组进行垂直堆叠。它们只有一个参数，用来接收数组序列。例如：

```python
import numpy as np

a = np.arange(4).reshape(2, 2)
print(a)
# [[0 1]
#  [2 3]]

b = np.arange(5, 9).reshape(2, 2)
print(b)
# [[5 6]
#  [7 8]]

# 水平堆叠
c = np.hstack((a, b))
print(c)
# [[0 1 5 6]
#  [2 3 7 8]]

# 垂直堆叠
d = np.vstack((a, b))
print(d)
# [[0 1]
#  [2 3]
#  [5 6]
#  [7 8]]
```

3.5.5 分割数组

`numpy.split()` 函数沿特定的轴分割数组。参数为：

1. `ary`：待分割的数组。
2. `indices_or_sections`：如果是一个整数，就平均切分出指定数量的子数组，无法平均切分时会抛出 `ValueError` 异常；如果是一个序列，则由序列元素指定切分的位置，切分点处的元素分到后面的子数组中。
3. `axis`：沿指定的轴进行切分。默认值为 0。

```python
import numpy as np

a = np.arange(12)
print(a)  # [ 0  1  2  3  4  5  6  7  8  9 10 11]

# 平均分成 3 份
b = np.split(a, 3)
print(b)  # [array([0, 1, 2, 3]), array([4, 5, 6, 7]), array([ 8,  9, 10, 11])]

# 索引 2、7 为切分点
c = np.split(a, [2, 7])
print(c)  # [array([0, 1]), array([2, 3, 4, 5, 6]), array([ 7,  8,  9, 10, 11])]
```

```
import numpy as np

a = np.arange(12).reshape(3, 4)
print(a)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]]

# 默认沿 0 轴切分
b = np.split(a, 3)
print(b)  # [array([[0, 1, 2, 3]]), array([[4, 5, 6, 7]]), array([[ 8,  9, 10, 11]])]

# 沿 1 轴平均分成 2 份
c = np.split(a, 2, axis=1)
print(c)
# [array([[0, 1],
#        [4, 5],
#        [8, 9]]), array([[ 2,  3],
#        [ 6,  7],
#        [10, 11]])]
```

`numpy.hsplit()` 函数用于水平分割数组，`numpy.vsplit()` 函数用于垂直分割数组。它们的参数与 `numpy.split()` 函数的前两个参数相同。例如：

```
import numpy as np

a = np.arange(12).reshape(3, 4)
print(a)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]]

# 水平分割，平均分成 2 份
b = np.hsplit(a, 2)
print(b)
# [array([[0, 1],
#        [4, 5],
#        [8, 9]]), array([[ 2,  3],
#        [ 6,  7],
#        [10, 11]])]

# 垂直分割，平均分成 3 份
c = np.vsplit(a, 3)
print(c) # [array([[0, 1, 2, 3]]), array([[4, 5, 6, 7]]), array([[ 8,  9, 10, 11]])]
```

3.5.6 添加和删除数组元素

`numpy.resize()` 函数返回指定大小的新数组，按照内存中的存储顺序读取原数组中的元素并填充到新数组中。参数为：

1. `arr`：原数组
2. `shape`：新的形状

```
import numpy as np

a = np.arange(6).reshape(2, 3)
print(a)
# [[0 1 2]
#  [3 4 5]]

b = np.resize(a, (3, 2))
print(b)
# [[0 1]
#  [2 3]
#  [4 5]]

# 新数组中的元素个数大于原数组，则循环读取原数组中的元素
c = np.resize(a, (4, 4))
print(c)
# [[0 1 2 3]
#  [4 5 0 1]
#  [2 3 4 5]
#  [0 1 2 3]]
```

`numpy.append()`函数在数组的末尾添加元素。追加操作会分配新的内存空间，并把原数组复制到新数组中。参数为：

1. `arr`：原数组。
2. `values`：要添加的元素组成的序列。形状必须和原数组匹配，否则抛出`ValueError`异常。
3. `axis`：默认值为`None`。当`axis`无定义时，将元素添加到原数组末尾，并返回一维数组；当`axis`有定义时，要求`arr`与`values`维数相同。当`axis`为 0 时，将新元素作为新的行，要求列数相同；当`axis`为 1 时，将新元素作为新的列，要求行数相同。

```python
import numpy as np

a = np.arange(6).reshape(2, 3)
print(a)
# [[0 1 2]
#  [3 4 5]]

# 追加一个元素
b = np.append(a, 10)
print(b)  # [ 0  1  2  3  4  5 10]

# 追加多个元素
c = np.append(a, [10, 11, 12])
print(c)  # [ 0  1  2  3  4  5 10 11 12]

# 追加行
d = np.append(a, [[10, 11, 12], [7, 8, 9]], axis=0)
print(d)
# [[ 0  1  2]
#  [ 3  4  5]
#  [10 11 12]
#  [ 7  8  9]]

# 追加列
e = np.append(a, [[10, 11], [12, 13]], axis=1)
print(e)
# [[ 0  1  2 10 11]
#  [ 3  4  5 12 13]]
```

`numpy.insert()` 函数在给定的索引之前、沿给定轴向数组中插入元素，返回新数组。参数为：

1. `arr`：待插入的数组。
2. `obj`：插入位置，将元素插入此位置之前。
3. `values`：要插入的元素。
4. `axis`：沿着此参数指定的轴插入元素。如果未提供，则返回一维数组。

```python
import numpy as np

a = np.arange(6).reshape(2, 3)
print(a)
# [[0 1 2]
#  [3 4 5]]

# 不指定 axis 参数，返回一维数组
b = np.insert(a, 3, [9, 10])
print(b)  # [ 0  1  2  9 10  3  4  5]

# 将 values 广播，沿 1 轴插入指定位置，即作为新的一列
c = np.insert(a, 1, 10, axis=1)
print(c)
# [[ 0 10  1  2]
#  [ 3 10  4  5]]

# 沿 0 轴插入，即作为新的一行
d = np.insert(a, 1, 10, axis=0)
print(d)
# [[ 0  1  2]
#  [10 10 10]
#  [ 3  4  5]]
```

`numpy.delete()`函数删除数组中的指定子数组，返回新数组。参数为：

1. `arr`：待删除的数组。
2. `obj`：可以为整数、整数序列或`slice`对象，指定要删除的子数组。
3. `axis`：沿着此参数指定的轴删除元素。如果未提供，则返回一维数组。

```python
import numpy as np

a = np.arange(12).reshape(3, 4)
print(a)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]]

# 不指定 axis 参数，返回一维数组
b = np.delete(a, 1)
print(b)  # [ 0  2  3  4  5  6  7  8  9 10 11]

# 删除整行
c = np.delete(a, 1, axis=0)
print(c)
# [[ 0  1  2  3]
#  [ 8  9 10 11]]

# 删除整列
d = np.delete(a, 1, axis=1)
print(d)
# [[ 0  2  3]
#  [ 4  6  7]
#  [ 8 10 11]]
```

`numpy.unique()` 函数去除数组中的重复元素，返回去重后的新数组。参数为：

1. `arr`：待去重的数组。如果不是一维数组，则结果会展开成一维数组。
2. `return_index`：可选。如果为 `True`，返回新数组元素在原数组中的索引。
3. `return_inverse`：可选。如果为 `True`，返回原数组元素在新数组中的索引。
4. `return_counts`：可选。如果为 `True`，则返回新数组中的元素在原数组中的出现次数。

```python
import numpy as np

a = np.array([1, 2, 3, 1, 2, 1, 4, 7, 3, 5])
print(a)  # [1 2 3 1 2 1 4 7 3 5]

b = np.unique(a)
print(b)  # [1 2 3 4 5 7]

c, index, inverse, counts = np.unique(a, True, True, True)
print(c)  # [1 2 3 4 5 7]
print(index)  # [0 1 2 6 9 7]
print(inverse)  # [0 1 2 0 1 0 3 5 2 4]
print(counts)  # [3 2 2 1 1 1]
```

3.5.7 `Ndarray` 数组的拷贝

视图又称为浅拷贝，是对象的一个别称或引用，通过该别称或引用可以访问原始对象。视图指向原始对象，对视图进行修改会影响到原始对象。例如，`Ndarray`数组的切片操作返回原数组的视图。

通过视图修改数组元素时，原数组会受到影响。但修改视图的形状时，原数组不受影响。

`view()`方法返回当前数组对象的视图。

副本又称为深拷贝，是对象的一个完整拷贝。副本和原始对象有不同的内存空间，修改副本不会影响原始对象。

`copy()`方法返回当前数组对象的副本。

4 NumPy函数

4.1 字符串函数

NumPy 字符串函数定义在`numpy.char`类中，用于对`dtype`为`numpy.string_`或`numpy.unicode_`的数组执行向量化字符串操作。

`add()`函数对两个数组中的元素进行字符串拼接。例如：

```
import numpy as np

print(np.char.add('hello', ' world'))  # hello world
print(np.char.add(['hello', 'my'], [' world', ' son']))  # ['hello world' 'my son']
```

`multiply()`函数将字符串重复若干次。例如：

```
import numpy as np

print(np.char.multiply('hello', 3))  # hellohellohello
print(np.char.multiply(['ok', 'id'], 2))  # ['okok' 'idid']
```

`center()`函数将字符串居中，并使用指定字符在左侧和右侧进行填充。例如：

```
import numpy as np

print(np.char.center('ok', 10, fillchar='-'))  # ----ok----
print(np.char.center(['good', 'hello'], 8, fillchar='*'))  # ['**good**' '*hello**']
```

`capitalize()`函数将字符串的第一个字母转换为大写。例如：

```
import numpy as np

print(np.char.capitalize('hello world!'))  # Hello world!
print(np.char.capitalize(['hello world!', 'you pro?']))  # ['Hello world!' 'You pro?']
```

`title()`函数将字符串中每个单词的第一个字母转换为大写。
`lower()`函数将字符串的所有字母转换成小写。
`upper()`函数将字符串的所有字母转换成大写。

`split()`函数用指定分隔符对字符串进行分割，并返回数组列表。例如：

```
import numpy as np

print(np.char.split(['you are my son', 'i can trust you'], sep=' '))
# [list(['you', 'are', 'my', 'son']) list(['i', 'can', 'trust', 'you'])]
```

`splitlines()`函数用换行符分割字符串，返回数组列表。例如：

```
import numpy as np

print(np.char.splitlines(['you are\n my son', 'i \ncan trust you']))
# [list(['you are', ' my son']) list(['i ', 'can trust you'])]
```

`strip()`函数去除字符串开头和结尾的指定字符。例如：

```
import numpy as np

print(np.char.strip(['**my**', '*ok'], '*'))  # ['my' 'ok']
```

`join()`函数用指定分隔符连接字符串中的各个字符。例如：

```
import numpy as np

print(np.char.join('-', ['good', 'nice']))  # ['g-o-o-d' 'n-i-c-e']
print(np.char.join(['-', ':'], ['good', 'nice']))  # ['g-o-o-d' 'n:i:c:e']
```

`replace()`函数用新字符串替换字符串中的指定子串。例如：

```
import numpy as np

print(np.char.replace(['i am a good man', 'very good'], 'good', 'nice'))
# ['i am a nice man' 'very nice']
```

`encode()`函数对字符串进行编码，将字符串转换成`bytes`类型。例如：

```
import numpy as np

print(np.char.encode(['i am a good man', 'very good'], 'utf-8'))
# [b'i am a good man' b'very good']
```

`decode()`函数用于解码，将`bytes`类型值转换成字符串。例如：

```
import numpy as np

print(np.char.decode([b'i am a good man', b'very good'], 'utf-8'))
# ['i am a good man' 'very good']
```

4.2 数学函数

标准三角函数：`numpy.sin()`、`numpy.cos()`、`numpy.tan()`。参数为弧度制。
反三角函数：`numpy.arcsin()`、`numpy.arccos()`、`numpy.arctan()`。返回值为弧度制。

`numpy.around()`函数对浮点数进行四舍五入。参数为：

1. `a`：数组。
2. `decimals`：保留的小数位数。可选，默认值为 0，表示四舍五入到整数。如果为负数，将四舍五入到小数点左侧的位置。

`numpy.floor()`函数用于向下取整，`numpy.ceil()`函数用于向上取整。

4.3 算数函数

四则运算：`numpy.add()`、`numpy.subtract()`、`numpy.multiply`、`numpy.divide()`

`numpy.reciprocal()`函数用于取倒数。例如：

```
import numpy as np

print(np.reciprocal([0.25, 0.5, 2, 10]))  # [4.  2.  0.5 0.1]
```

`numpy.power()`函数为幂运算，第一个参数为底数，第二个参数为指数。例如：

```
import numpy as np

print(np.power([2, 3, 4], 3))  # [ 8 27 64]
print(np.power([2, 3, 4], [4, 3, 2]))  # [16 27 16]
```

`numpy.mod()`和`numpy.remainder()`函数用于取余数。例如：

```python
import numpy as np

print(np.mod([15, 16, 17], [5, 6, 7]))  # [0 4 3]
```

4.4 统计函数

`numpy.amax()`：获取数组沿指定轴的最大值
`numpy.amin()`：获取数组沿指定轴的最小值
`numpy.ptp()`：计算数组中最大值与最小值的差

```python
import numpy as np

x = np.random.randint(0, 100, 9).reshape(3, 3)
print(x)
# [[62 32 73]
#  [77 35 34]
#  [20 27 98]]

# 所有元素中的最大值
print(np.amax(x))  # 98

# 每列的最大值
print(np.amax(x, 0))  # [77 35 98]

# 每行的最大值
print(np.amax(x, 1))  # [73 77 98]

# 所有元素中的最小值
print(np.amin(x))  # 20

# 每列的最小值
print(np.amin(x, 0))  # [20 27 34]

# 每行的最小值
print(np.amin(x, 1))  # [32 34 20]

print(np.ptp(x))  # 78
print(np.ptp(x, 0))  # [57  8 64]
print(np.ptp(x, 1))  # [41 43 78]
```

`numpy.percentile()`函数计算一个数，使得该数大于数组中指定百分比的数。参数为：

1. `a`：数组
2. `q`：百分数，取值在 0—100 之间。
3. `axis`：轴

```python
import numpy as np

x = np.array([[10, 7, 4], [3, 2, 1]])
print(x)
# [[10  7  4]
#  [ 3  2  1]]

# 计算一个数，使得该数大于数组 x 中 50% 的数
print(np.percentile(x, 50))  # 3.5

# 每列计算一个数，使得该数大于该列中 30% 的数
print(np.percentile(x, 30, axis=0))  # [5.1 3.5 1.9]

# 每行计算一个数，使得该数大于该行中 70% 的数
print(np.percentile(x, 70, axis=1))  # [8.2 2.4]
```

`numpy.average()`函数用于计算数组元素的加权平均值。参数为：

1. `a`：待计算的数组
2. `axis`：轴。如果没有指定轴，则会将数组展开为一维数组。
3. `weights`：权重
4. `returned`：布尔值，如果为`True`，则返回权重的和，否则不返回。

```python
import numpy as np

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

# 算术平均值
print(np.average(x))  # 2.5

# 加权平均值
print(np.average(x, weights=w))  # 2.0
```

```python
import numpy as np

x = np.random.randint(1, 100, 6).reshape(3, 2)
print(x)
# [[16 39]
#  [34 64]
#  [ 9 39]]

# 计算每列的加权平均值，权值的个数必须和行数匹配
w0 = np.array([4, 3, 2])
print(np.average(x, weights=w0, axis=0))  # [20.44444444 47.33333333]

# 计算每行的加权平均值，权值的个数必须和列数匹配
w1 = np.array([5, 7])
print(np.average(x, weights=w1, axis=1))  # [29.41666667 51.5        26.5       ]
```

`numpy.median()`：计算数组元素的中位数
`numpy.mean()`：计算数组元素的算术平均值
`numpy.std()`：计算数组元素的标准差
`numpy.var()`：计算数组元素的方差

4.5 排序函数

`numpy.sort()`函数对数组进行排序，返回排序后的数组的副本。参数为：

1. `a`：待排序的数组
2. `axis`：轴。可选，如果不指定轴，则会沿着最后一个轴排序。
3. `kind`：采用的排序算法。`'quicksort'`表示快速排序，`'mergesort'`表示归并排序，`'heapsort'`表示堆排序。可选，默认值为`'quicksort'`。
4. `order`：如果数组元素包含字段，则指定要排序的字段。可选。

```python
import numpy as np

x = np.random.randint(1, 100, 6).reshape(3, 2)
print(x)
# [[95 97]
#  [37 24]
#  [96 85]]

print(np.sort(x))
# [[95 97]
#  [24 37]
#  [85 96]]

# 按列排序
print(np.sort(x, axis=0))
# [[37 24]
#  [95 85]
#  [96 97]]

# 按行排序
print(np.sort(x, axis=1))
# [[95 97]
#  [24 37]
#  [85 96]]
```

`numpy.argsort()` 函数对输入数组沿给定轴执行间接排序，并使用指定排序类型返回索引数组，这个索引数组用于构造排序后的数组。例如：

```python
import numpy as np

x = np.random.randint(1, 100, 6)
print(x)  # [40 80 79 10 27 53]

# 返回索引数组
i = np.argsort(x)
print(i)  # [3 4 0 5 2 1]

# 用索引数组重构原数组
y = x[i]
print(y)  # [10 27 40 53 79 80]
```

`numpy.lexsort()` 函数使用键序列执行间接稳定排序。给定多个排序键（可以解释为电子表格中的列），此函数返回一个整数索引数组。序列中的最后一个键为主键，先按照最后一个键排序，再按照倒数第二个键排序，依此类推。`keys` 参数必须是一个可以转换为相同形状数组的对象序列。如果为 `keys` 参数提供了一个二维数组，那么它的行将被解释为排序键，并根据最后一行、倒数第二行等进行排序。例如：

```python
import numpy as np

first = ('raju', 'anli', 'ravi', 'amar')
second = ('f.y.', 'p.y.', 's.y.', 'n.y.')

# 先按 first 排序，再按 second 排序
i = np.lexsort((second, first))
print(i)  # [3 1 0 2]

print([first[k] for k in i])  # ['amar', 'anli', 'raju', 'ravi']
print([second[k] for k in i])  # ['n.y.', 'p.y.', 'f.y.', 's.y.']
```

`numpy.msort()` 函数将数组按第一个轴排序。`numpy.msort(a)` 相当于 `numpy.sort(a, axos=0)`。

`numpy.sort_complex()` 函数对复数按照先实部后虚部的顺序进行排序。

`numpy.partition()` 函数执行分划操作。参数为：

1. `a`：待操作的数组
2. `kth`：枢轴元素的索引。将排序后索引应为 `kth` 的元素放在正确位置，也就是确定第 `kth + 1` 小的元素，小于该元素的元素放在左边，大于该元素的元素放在右边。
3. `axis`：轴，可选。
4. `kind`：排序算法。可选，默认值为 `'introselect'`。
5. `order`：排序字段，可选。

```python
import numpy as np

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

# 将排序后索引为 3 的元素放在正确位置
print(np.partition(x, 3))  # [2 1 3 4]

# 将排序后索引为 1 和 3 的元素放在正确位置，其余元素按大小关系放在这两个元素左右
print(np.partition(x, (1, 3)))  # [1 2 3 4]
```

`numpy.argpartition()` 函数执行分划操作，返回索引数组。参数与 `numpy.partition()` 函数相同。例如：

```python
import numpy as np

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

# 寻找第 4 小的元素
i = np.argpartition(x, 3)
print(i)  # [2 3 0 1]
print(x[i])  # [2 1 3 4]
print(x[i][3])  # 4

# 寻找第 2 大的元素
i = np.argpartition(x, -2)
print(i)  # [3 2 0 1]
print(x[i])  # [1 2 3 4]
print(x[i][-2])  # 3
```

4.6 搜索函数

`numpy.max()`：返回最大值。可以指定轴。
`numpy.min()`：返回最小值。可以指定轴。
`numpy.argmax()`：返回最大元素的索引。可以指定轴。
`numpy.argmin()`：返回最小元素的索引。可以指定轴。
`numpy.nonzero()`：返回非零元素的索引。
`numpy.where()`：返回数组中满足给定条件的元素的索引。
`numpy.extract()`：返回数组中满足给定条件的元素。

```python
import numpy as np

x = np.array([[31, 25, 0], [40, 0, 53], [0, 14, 6]])
print(x)
# [[31 25  0]
#  [40  0 53]
#  [ 0 14  6]]

print(np.max(x))  # 53
print(np.max(x, axis=0))  # [40 25 53]
print(np.max(x, axis=1))  # [31 53 14]

print(np.min(x))  # 0
print(np.min(x, axis=0))  # [0 0 0]
print(np.min(x, axis=1))  # [0 0 0]

print(np.argmax(x))  # 5
print(np.argmax(x, axis=0))  # [1 0 1]
print(np.argmax(x, axis=1))  # [0 2 1]

print(np.argmin(x))  # 2
print(np.argmin(x, axis=0))  # [2 1 0]
print(np.argmin(x, axis=1))  # [2 1 0]

print(np.nonzero(x))
# (array([0, 0, 1, 1, 2, 2], dtype=int64), array([0, 1, 0, 2, 1, 2], dtype=int64))
print(x[np.nonzero(x)])  # [31 25 40 53 14  6]

print(np.where(x > 30))  # (array([0, 1, 1], dtype=int64), array([0, 0, 2], dtype=int64))
print(x[np.where(x > 30)])  # [31 40 53]

print(np.extract(x % 2 == 0, x))  # [ 0 40  0  0 14  6]
```

4.7 IO函数

NumPy 可以读写磁盘上的文本文件或二进制文件。此外，NumPy 为`Ndarray`对象引入了一个简单的文件格式 npy，npy 文件用于存储重建`Ndarray`对象所需的数据、图形、`dtype`和其他信息。

`numpy.save()`函数将数组保存到 .npy 文件中。参数为：

1. `file`：文件路径。要求文件扩展名为 .npy，如果没有扩展名则会自动添加。
2. `arr`：要保存的数组。
3. `allow_pickle`：布尔值，可选，默认为`True`，表示允许使用 Python pickles 保存对象数组。Python 中的 pickle 用于在保存到磁盘文件或从磁盘文件读取前，对对象进行序列化和反序列化。
4. `fix_imports`：布尔值，可选，默认为`True`，为了方便 Python 2 中读取 Python 3 保存的数据。

`numpy.load()` 函数从 .npy 文件中读取数组，参数为文件路径，返回值为数组。例如：

```
import numpy as np

a = np.array([1, 2, 3])
np.save('out.npy', a)

arr = np.load('out.npy')
print(arr)  # [1 2 3]
```

`numpy.savez()` 函数用于将多个数组保存到 .pyz 文件中。参数为：

1. `file`：文件路径。要求文件扩展名为 .npz，如果没有扩展名则会自动添加。
2. `*args`：要保存的数组。
3. `**kwds`：要保存的数组，关键字参数的名称作为数组的名字。对于用位置参数传递的数组，会自动命名为 `arr_0, arr_1, ...`。

`numpy.load()` 函数也可以读取 .npy 文件，参数为文件路径，返回 `numpy.lib.npyio.NpzFile` 对象。可以用保存数组时指定的名字来获取指定的数组。例如：

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.array([7, 8, 9])
np.savez('out.npz', a, b, c=c)

arrs = np.load('out.npz')
print(arrs['arr_0'])  # [1 2 3]
print(arrs['arr_1'])  # [4 5 6]
print(arrs['c'])  # [7 8 9]
```

`numpy.savetxt()` 函数用文本文件格式存储数据。参数为：

1. `FILENAME`：文件路径
2. `a`：数组
3. `fmt`：格式描述符。可选，默认为 `"%d"`。
4. `delimiter`：分隔符。可选，默认值为 `","`。

`numpy.loadtxt()` 函数读取文本文件中保存的数据。参数为：

1. `FILENAME`：文件路径
2. `dtype`：数组元素类型。可选，默认值为 `int`。
3. `delimiter`：分隔符。可选，默认值为 `" "`。