1.基础

1.1 概念

- NumPy 的全称是“ Numeric Python”，它是 Python 的第三方扩展包，主要用来计算、处理一维或多维数组
- 在数组算术计算方面， NumPy 提供了大量的数学函数
- NumPy 的底层主要用 C语言编写，因此它能够高速地执行数值计算
- NumPy 还提供了多种数据结构，这些数据结构能够非常契合的应用在数组和矩阵的运算上

1.2 优点

NumPy 可以很便捷高效地处理大量数据，使用 NumPy 做数据处理的优点如下：

- NumPy 是 Python 科学计算基础库
- NumPy 可以对数组进行高效的数学运算
- NumPy 的 ndarray 对象可以用来构建多维数组
- NumPy 能够执行傅立叶变换与重塑多维数组形状
- NumPy 提供了线性代数，以及随机数生成的内置函数

1.3 与python列表区别

NumPy 数组是同质数据类型（homogeneous），即数组中的所有元素必须是相同的数据类型。数据类型在创建数组时指定，并且数组中的所有元素都必须是该类型。

Python 列表是异质数据类型（heterogeneous），即列表中的元素可以是不同的数据类型。列表可以包含整数、浮点数、字符串、对象等各种类型的数据。

NumPy 数组提供了丰富的数学函数和操作，如矩阵运算、线性代数、傅里叶变换等。

Python 列表提供了基本的列表操作，如添加、删除、切片、排序等。

1.4 安装

```
pip install numpy==1.26.0 -i https://pypi.tuna.tsinghua.edu.cn/simple/
```

2.ndarray

NumPy 定义了一个 n 维数组对象，简称 ndarray 对象，它是一个一系列相同类型元素组成的数组集合。数组中的每个元素都占有大小相同的内存块，您可以使用索引或切片的方式获取数组中的每个元素。

ndarray 对象采用了数组的索引机制，将数组中的每个元素映射到内存块上，并且按照一定的布局对内存块进行排列，常用的布局方式有两种，即按行或者按列。

主要特点

- 多维数组：ndarray 可以表示任意维度的数组，包括一维、二维、三维等。
- 同质数据类型：ndarray 中的所有元素必须是相同的数据类型。
- 高效内存布局：ndarray 在内存中是连续存储的，这使得数组的访问和操作非常高效。

- 丰富的功能和方法：ndarray 提供了丰富的数学函数和操作，如矩阵运算、线性代数、傅里叶变换等。

使用方式:

- ndarray 是通过 array 函数或其他 NumPy 函数（如 zeros、ones、arange 等）创建的。
- array 函数接受一个序列作为输入，并返回一个 ndarray 对象。

2.1 array创建对象

通过 NumPy 的内置函数 array() 可以创建 ndarray 对象，其语法格式如下：

```
numpy.array(object, dtype = None, copy = True, order = None,ndmin = 0)
```

参数说明：

序号	参数	描述说明
1	object	表示一个数组序列
2	dtype	可选参数，通过它可以更改数组的数据类型
3	copy	可选参数，表示数组能否被复制，默认是 True
4	order	以哪种内存布局创建数组，有 3 个可选值，分别是 C(行序列)/F(列序列)/A(默认)
5	ndmin	用于指定数组的维度

案例：

```
import numpy as np

# 创建一个一维数组
def array_test_one():
    dimensionalArray = np.array([1,2,3,4])
    print("array 创建一维数组：",dimensionalArray)
    print("array 创建以为数组：",type(dimensionalArray))
    print("ndim 查看/指定数组维度：",dimensionalArray.ndim)

# 创建一个二维数组
def array_test_two():
    dimensionalArray = np.array([[1,2,3,4],[5,6,7,8]])
    print("array 创建二维数组：",dimensionalArray)
    print("array 创建二维数组类型：",type(dimensionalArray))
    print("ndim 查看维度：",dimensionalArray.ndim)
```

2.2 指定/查看数组维度

数组的维度就是一个数组中的某个元素，当用数组下标表示的时候，需要用几个数字来表示才能唯一确定这个元素，这个数组就是几维

`ndmin` 是 NumPy 中用于创建数组时指定最小维度的参数。它通常在 `numpy.array()` 函数中使用。通过 `ndmin`，你可以确保生成的数组至少具有指定的维度。

`ndim` 是 NumPy 数组的一个属性，用于返回数组的维度数（即数组的秩）。它表示数组有多少个维度。

案例：

```
import numpy as np

# 指定/查看数组维度
def ndmin_test():
    threeArray = np.array([1,2,3,4],ndmin=2)
    print("ndmin 指定数组：",threeArray)
    print("ndmin 指定数组：",type(threeArray))
    print("ndim 查看维度：",threeArray.ndim)
```

2.3 reshape数组变维

reshape() 函数允许你在不改变数组数据的情况下，改变数组的维度。

reshape() 返回的是一个新的数组，原数组的形状不会被修改。reshape() 可以用于多维数组，例如将一个一维数组重塑为二维数组。

但是，reshape后产生的新数组是原数组的一个视图，即它与原数组共享相同的数据，但可以有不同的形状或维度，且对视图的修改会直接影响原数组。

元素总数必须匹配：新形状中的元素总数必须与原数组中的元素总数相同。

例如，一个长度为6的一维数组可以被重塑为 (2, 3) 或 (3, 2)，表示原数组被重塑为2行3列或3行2列的数组，但不能被重塑为 (2, 2)。

案例：

将数组改变为二维数组：

```
import numpy as np

# 数组变维
def reshape_test():
    oneArray = np.array([1,2,3,4,5,6])
    print("oneArray 原数组内容：",oneArray)
    print("oneArray 原数组维度：",oneArray.ndim)
    oneArray = oneArray.reshape((3,2))
    print("oneArray 新数组内容：",oneArray)
    print("oneArray 新数组维度：",oneArray.ndim)
```

-1 作为占位符：你可以使用 -1 作为占位符，让 numpy 自动计算某个维度的大小。

例如：

```
import numpy as np

# 数组变维
def reshape_test():
    oneArray = np.array([1,2,3,4,5,6])
    print("oneArray 原数组内容：",oneArray)
    print("oneArray 原数组维度：",oneArray.ndim)
    oneArray = oneArray.reshape((3,-1))
    print("oneArray 新数组内容：",oneArray)
    print("oneArray 新数组维度：",oneArray.ndim)
```

reshape() 还可以将一维数组重塑为三维数组：

例如：

```
import numpy as np

# 创建一个一维数组
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

# 使用 reshape() 函数将其转换为三维数组
reshaped_arr = arr.reshape((2, 3, 2))

print(reshaped_arr)
print("ndim:", reshaped_arr.ndim)
```

说明：

reshape((2, 3, 2))表示将数组重塑为一个三维数组，其形状为 (2, 3, 2)，具体来说，这个形状表示：

- **第一个维度**：也叫做层维度，有 2 个元素。
- **第二个维度**：也叫做行维度，每个元素有 3 个元素。
- **第三个维度**：也叫做列维度，每个元素有 2 个元素。

具体过程：

首先，将一维数组分成 2 个部分：

```
[1, 2, 3, 4, 5, 6] 和 [7, 8, 9, 10, 11, 12]
```

然后，将每个部分分成 3 个部分：

```
[1, 2], [3, 4], [5, 6] 和 [7, 8], [9, 10], [11, 12]
```

最后，将每个部分分成 2 个部分，如：

```
[1, 2]
```

注意：形状 (2, 3, 2)中的参数个数相乘要等于数组中元素的个数，如：数组[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]有12个元素，形状(2,3,2)的参数相乘：2x3x2=12。

示例：改变原数组的形状产生新数组，修改新数组中的元素值，原数组的值也会改变。

```
import numpy as np
```

```
# 创建一个一维数组
c = np.array([1, 2, 3, 4, 5, 6])

# 通过 reshape 创建视图
d = c.reshape((2, 3))
print(d)
# 输出:
# [[1 2 3]
#  [4 5 6]]

# 修改视图中的元素
d[0, 0] = 100
print(c)  # 原数组也被修改

#输出:
[100   2   3   4   5   6]
```

3.数据类型

NumPy 提供了比 Python 更加丰富的数据类型，如下所示：

序号	数据类型	语言描述
1	bool_	布尔型数据类型（True 或者 False）
2	int_	默认整数类型，类似于 C 语言中的 long，取值为 int32 或 int64
3	intc	和 C 语言的 int 类型一样，一般是 int32 或 int 64
4	intp	用于索引的整数类型（类似于 C 的 ssize_t，通常为 int32 或 int64）
5	int8	代表与1字节相同的8位整数。值的范围是-128到127
6	int16	代表 2 字节（16位）的整数。范围是-32768至32767
7	int32	代表 4 字节（32位）整数。范围是-2147483648至2147483647
8	int64	表示 8 字节（64位）整数。范围是-9223372036854775808至9223372036854775807
9	uint8	1字节（8位）无符号整数
10	uint16	2 字节（16位）无符号整数

序号	数据类型	语言描述
11	uint32	4 字节（32位）无符号整数
12	uint64	8 字节（64位）无符号整数
13	float_	float64 类型的简写
14	float16	半精度浮点数，包括：1 个符号位，5 个指数位，10个尾数位
15	float32	单精度浮点数，包括：1 个符号位，8 个指数位，23个尾数位
16	float64	双精度浮点数，包括：1 个符号位，11 个指数位，52个尾数位
17	complex_	复数类型，与 complex128 类型相同
18	complex64	表示实部和虚部共享 32 位的复数
19	complex128	表示实部和虚部共享 64 位的复数
20	str_	表示字符串类型，等价于unicode_
21	bytes_	表示字节串类型，基于字节
22	string_	表示字节串类型，等价于 `bytes_`，基于字节，NumPy 2.0以后版本被移除，使用 `bytes_` 代替
23	unicode_	表示字节串类型，基于字符，NumPy 2.0以后版本被移除，使用str_`代替

3.1 数据类型对象

数据类型对象（Data Type Object）又称 dtype 对象，是用来描述与数组对应的内存区域如何使用。

1.可以在创建数组时指定 dtype 参数来定义数组中元素的数据类型。

例如：

```
import numpy as np

# 创建一个 int32 类型的数组
arr_int = np.array([1, 2, 3], dtype=np.int32)
print(arr_int.dtype)  # 输出: int32

# 创建一个 float64 类型的数组
arr_float = np.array([1, 2, 3], dtype=np.float64)
print(arr_float.dtype)  # 输出: float64
```

```
arr_str = np.array([1,2,3],dtype=np.string_)
print(arr_str)
#输出：
AttributeError: `np.string_` was removed in the NumPy 2.0 release. Use
`np.bytes_` instead.

arr_str = np.array([1,2,3],dtype=np.bytes_)
print(arr_str)
```

```
#输出：
[b'1' b'2' b'3']

arr_str = np.array([1,2,3],dtype=np.unicode_)
print(arr_str)
# 输出：
AttributeError: `np.unicode_` was removed in the NumPy 2.0 release. Use
`np.str_` instead.

arr_str = np.array([1,2,3],dtype=np.str_)
print(arr_str)
# 输出：
['1' '2' '3']
```

2.可以使用数组的 dtype 属性来获取数组中元素的数据类型。

例如：

```
arr = np.array([1, 2, 3])
print(arr.dtype)  # 输出: int32
```

3.可以使用 **astype()** 方法来转换数组中元素的数据类型。

例如：

```
arr = np.array([1, 2, 3])
arr_float = arr.astype(np.float64)
print(arr_float.dtype)  # 输出: float64
```

3.2 数据类型标识码

NumPy 中每种数据类型都有一个唯一标识的字符码，int8, int16, int32, int64 四种数据类型可以使用字符串 'i1', 'i2','i4','i8' 代替，如下所示：

字符	对应类型
b	代表布尔型
i	带符号整型
u	无符号整型
f	浮点型
c	复数浮点型
m	时间间隔（timedelta）
M	datatime（日期时间）
O	Python对象
S,a	字节串（S）与字符串（a）
U	Unicode
V	原始数据（void）

以下是 NumPy 中常见的数据类型标识码及其对应的详细列表：

整数类型

数据类型	标识码	描述
int8	i1	8 位有符号整数
int16	i2	16 位有符号整数
int32	i4	32 位有符号整数
int64	i8	64 位有符号整数
uint8	u1	8 位无符号整数
uint16	u2	16 位无符号整数
uint32	u4	32 位无符号整数
uint64	u8	64 位无符号整数

浮点数类型

数据类型	标识码	描述
float16	f2	16 位浮点数（半精度）
float32	f4	32 位浮点数（单精度）
float64	f8	64 位浮点数（双精度）

复数类型

数据类型	标识码	描述
complex64	c8	64 位复数（单精度）
complex128	c16	128 位复数（双精度）

布尔类型

数据类型	标识码	描述
bool	b1	布尔类型

字符串类型

数据类型	标识码	描述
bytes	S10	长度为 10 的字节串
str	U10	长度为 10 的 Unicode 字符串

Python 对象类型

数据类型	标识码	描述
O	O	Python 对象类型

例如：自定义一个int32的数据类型

```
dt=np.dtype('i4')
data = np.array([1,2,3,4],dtype=dt)
print(data)   #输出：[1 2 3 4]
print(data.dtype)   #输出：int32
```

可以自定义复杂的数据结构：

```
dt=np.dtype([('name','S10'),('age','i4')])
data = np.array([('zhangsan',20),('lisi',21)],dtype=dt)
print(data)         #输出：[(b'zhangsan', 20) (b'lisi', 21)]
print(data.dtype)   #输出：[('name', 'S10'), ('age', '<i4')]
```

说明：

在编程中，字节序标记用于指定数据的字节顺序。常见的字节序标记包括：

- `<`：小端序，数据的最低有效字节存储在内存的最低地址，而最高有效字节存储在内存的最高地址。
- `>`：大端序，数据的最高有效字节存储在内存的最低地址，而最低有效字节存储在内存的最高地址。

4.数组属性

4.1 shape

返回一个元组，元组中的每个元素表示数组在对应维度上的大小。元组的长度等于数组的维度数。

shape 属性功能：

1. 返回一个由数组维度构成的元组
2. 通过赋值，可以用来调整数组维度的大小

```
def shape_test():
    array_one = np.array([[1,2,3],[4,5,6]])
    print('array_one 原数组维度：',array_one.shape)
    print('array_one 原数组内容：',array_one)
    array_one.shape = (3,2)
    print('array_one 转变数组维度大小之后的数组维度：',array_one.shape)
    print('array_one 转变数组维度大小之后的数组内容：',array_one)
```

如果使用shape属性修改数组的形状，则修改的是原数组的形状，reshape修改数组的形状会返回一个新数组，不修改原数组的形状。

4.2 ndim

ndim 属性功能：

返回的是数组的维数

4.3 itemsize

itemsize 属性功能：

1. 返回数组中每个元素的大小（以字节为单位），即每个元素占用的字节数

```
'''
    itemsize 属性功能：
        1、返回数组中每个元素的大小（以字节为单位）
'''
def itemsize_test():
    array_one = np.array([[1,2,3],[4,5,6]],dtype=np.int8)
    print('array_one 数组中每个元素的大小：',array_one.itemsize)    #输出：1
    array_one = np.array([[1,2,3],[4,5,6]],dtype=np.int16)
    print('array_one 数组中每个元素的大小：',array_one.itemsize)    #输出：2
```

4.4 flags

flags 属性功能：

1. 返回 ndarray 数组的内存信息

```
'''
    flags 属性功能：
        1、返回 ndarray 数组的内存信息
'''
def flags_test():
    array_one = np.array([[1,2,3],[4,5,6]])
    print('array_one 数数组的内存信息：\n',array_one.flags)
```

输出：

```
C_CONTIGUOUS : True
F_CONTIGUOUS : False
OWNDATA : True
WRITEABLE : True
ALIGNED : True
WRITEBACKIFCOPY : False
```

说明：

```
C_CONTIGUOUS:
表示数组在内存中是 C 风格连续的（行优先）。
如果为 True，则数组是 C 风格连续的。

F_CONTIGUOUS:
表示数组在内存中是 Fortran 风格连续的（列优先）。
如果为 True，则数组是 Fortran 风格连续的。

OWNDATA:
表示数组是否拥有自己的数据（即是否是视图）。
如果为 True，则数组拥有自己的数据；如果为 False，则数组可能是从另一个数组或对象借用数据的。

WRITEABLE:
表示数组是否可写。
如果为 True，则数组是可写的；如果为 False，则数组是只读的。

ALIGNED:
表示数组是否对齐。
如果为 True，则数组的数据在内存中是对齐的。

WRITEBACKIFCOPY:
表示数组是否是通过 np.copy 创建的副本，并且需要将更改写回原始数组。
如果为 True，则数组是通过 np.copy 创建的副本，并且需要将更改写回原始数组。

UPDATEIFCOPY:
表示数组是否是通过 np.copy 创建的副本，并且需要将更改写回原始数组。
如果为 True，则数组是通过 np.copy 创建的副本，并且需要将更改写回原始数组。
```

5.创建数组的其他方法

5.1 empty()

empty() 方法用来创建一个指定形状（shape）、数据类型（dtype）且**未初始化**的数组（数组元素为随机值）

格式：

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

参数	描述
shape	数组的形状，可以是整数或整数元组
dtype	数组的数据类型，默认为 float
order	数组的内存布局，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

案例：

```
import numpy as np

'''
    empty 函数：
        1、用来创建一个指定形状（shape）、数据类型（dtype）且未初始化的数组（数组元素为随机值）
'''
def empty_test():
    array_one = np.empty(shape=(2,3),dtype='i1',order='C')
    print("empty 函数创建数组：",array_one)
```

5.2 zeros()

创建指定大小的数组，数组元素以 0 来填充

格式：

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

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

案例：

```
import numpy as np

'''
    zeros 函数：
        1、创建指定大小的数组，数组元素以 0 来填充
'''
def zeros_test():
    array_one = np.zeros(shape=(2,3),dtype='i1',order='C')
    print("zeros 函数创建数组：",array_one)
```

5.3 ones()

创建指定形状的数组，数组元素以 1 来填充

格式：

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

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

案例：

```
import numpy as np

'''
    ones 函数：
        1、创建指定形状的数组，数组元素以 1 来填充
'''
def ones_test():
    array_one = np.ones(shape=(2,3),dtype='i1',order='C')
    print("zeros 函数创建数组：",array_one)
```

5.4 arange()

arange() 函数用于创建一个等差数列的数组。它类似于 Python 内置的 range() 函数，但返回的是一个 NumPy 数组而不是一个列表。

格式：

```
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	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

案例1：

```python
import numpy as np

# 创建一个从 0 到 9 的数组
arr = np.arange(10)

print(arr)
# 输出:
# [0 1 2 3 4 5 6 7 8 9]
```

案例2:

```python
import numpy as np

# 创建一个从 1 到 10 的数组
arr = np.arange(1, 11)

print(arr)
# 输出:
# [ 1  2  3  4  5  6  7  8  9 10]
```

案例3:

```python
import numpy as np

# 创建一个从 0 到 9，步长为 2 的数组
arr = np.arange(0, 10, 2)

print(arr)
# 输出:
# [0 2 4 6 8]
```

案例4:

```python
import numpy as np

# 创建一个从 10 到 0，步长为 -1 的数组
arr = np.arange(10, 0, -1)

print(arr)
# 输出:
# [10  9  8  7  6  5  4  3  2  1]
```

案例5:

```python
import numpy as np

# 创建一个从 0 到 1，步长为 0.1 的浮点数数组
arr = np.arange(0, 1, 0.1)

print(arr)
# 输出:
# [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
```

注意：

- arange() 函数生成的数组不包含 stop 值。
- 如果 step 为负数，则 start 必须大于 stop，否则生成的数组为空。
- arange() 函数在处理浮点数时可能会出现精度问题，因为浮点数的表示和计算存在精度误差。

5.5 linspace

在指定的数值区间内，返回均匀间隔的一维等差数组，默认均分 50 份

格式：

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

参数	描述
start	起始值，默认为 0
stop	终止值（默认包含）
num	表示数值区间内要生成多少个均匀的样本，默认值为 50
endpoint	默认为 True，表示数列包含 stop 终止值，反之不包含
retstep	表示是否返回步长。如果为 True，则返回一个包含数组和步长的元组；如果为 False，则只返回数组。默认为 False。
dtype	返回 ndarray 的数据类型，默认为 None，表示根据输入参数自动推断数据类型。

案例：

```
'''
    linspace 函数：
        1、在指定的数值区间内，返回均匀间隔的一维等差数组，默认均分 50 份
'''

array_one = np.linspace(0,9,3,dtype='i1')
print("linspace 函数创建数组：",array_one)


array_one, step = np.linspace(0, 9, 3, retstep=True, dtype='i1')
print("linspace 函数创建数组：", array_one)
print(step)
```

说明：以上几个函数通常用于创建数字类型的数组，也可以用于创建布尔、字符串等类型的数组。但更适合用于创建数字数组。

5.6 full()

full()用于创建一个填充指定值的数组。

格式：

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

参数	描述
shape	数组的形状（如 `(2, 3)` 表示 2 行 3 列的数组）
fill_value	填充的值
dtype	数组的数据类型（如 `np.float32`）
order	数组的内存布局（如 `'C'` 表示 C 风格，`'F'` 表示 Fortran 风格）

案例：

```
import numpy as np

# 创建一个 2x3 的数组，填充值为 5
x = np.full((2, 3), 5)
print(x)
```

full_like():用于创建一个与给定数组形状和数据类型相同的数组，并用指定的值填充。

格式：

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

参数	描述
a	输入的数组，新数组的形状和数据类型将与 `a` 相同
fill_value	填充的值
dtype	数组的数据类型（如 `np.float32`）
order	数组的内存布局（如 `'C'` 表示 C 风格，`'F'` 表示 Fortran 风格）

案例：

```
import numpy as np

# 创建一个随机数组
a = np.random.rand(2, 3)
print("原始数组:\n", a)

# 创建一个与 a 形状和数据类型相同的数组，填充值为 5
b = np.full_like(a, 5)
print("填充后的数组:\n", b)
```

6.切片

ndarray 对象的内容可以通过索引或切片来访问和修改，与 Python 中 list 的切片操作一样；

slice()

在 Python 中，slice 可以作为一个对象来使用。你可以创建一个 slice 对象，然后使用它来获取序列的片段。

参数:

- start 是切片开始的位置（包含该位置）。
- stop 是切片结束的位置（不包含该位置）。
- step 是切片的步长，即选取元素的间隔。

案例:

```
import numpy as np

'''
    slice 函数：
        1、从原数组的上切割出一个新数组
'''
def slice_test():
    array_one = np.array([1,2,3,4])
    print("array_one 数组内容：",array_one)
    result = slice(0,len(array_one),2)
    print("slice 截取 array_one 数组内容：",array_one[result])
```

slice 操作也可通过 [start:stop:step] 的形式来实现

```
import numpy as np

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

冒号 : 的作用

- 表示范围: 冒号用于表示一个范围。例如，array[1:3] 表示从索引 1 到索引 3（不包括 3）的元素。
- 表示所有元素: 单独使用冒号表示所有元素。例如，array[:, 1] 表示所有行的第 1 列。
- 步长: 双冒号后面可以跟一个步长值，表示每隔多少个元素取一个。例如，array[::2] 表示每隔一个元素取一个。

注：冒号对于一维数组按索引号截取，二维数组按行和列截取。

省略号 ... 的作用

- 表示所有维度: 省略号用于表示数组的所有维度。例如，array[..., 1] 表示取所有行的第 1 列。
- 简化多维切片: 在多维数组中，省略号可以简化切片操作，避免显式地写出所有维度的索引。

注意：slice返回的数组是原数组的一个视图，修改数据会影响原数组中的数据。

```python
import numpy as np

array_one = np.array([[1, 2, 3], [4, 5, 6]])
array_two = array_one[..., 1]
print(array_two)
array_two[0] = 100
print(array_two)
print(array_one)
```

7.高级索引

NumPy 中的高级索引指的是使用整数数组、布尔数组或者其他序列来访问数组的元素。相比于基本索引，高级索引可以访问到数组中的任意元素，并且可以用来对数组进行复杂的操作和修改。

7.1 整数数组索引

整数数组索引是指使用一个数组来访问另一个数组的元素。这个数组中的每个元素都是目标数组中某个维度上的索引值。

适用于需要访问非连续元素或特定位置元素的场景。

注意：返回的新数组是一个**副本**，修改它不会影响原数组。

案例：

```python
import numpy as np

def one():
    array_one = np.array([[1,2,3],[4,5,6]])
    print(array_one)
    # [0,1,0]代表行索引、[0,1,2]代表列索引；即取出索引坐标 (0,0)、(1,1)、(0,2) 的元素
    array_one = array_one[[0,1,0],[0,1,2]]
    print(array_one)
```

取出 4 * 3 数组四个角的数据：

```python
import numpy as np

def two():
    array_one = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])
    print('原数组：\n',array_one)
    array_one = array_one[[0,0,-1,-1], [0,-1,0,-1]]
    print('这个数组的四个角元素是：')
    print(array_one)
```

7.2 布尔索引

布尔索引通过布尔运算（如：比较运算符）来获取符合指定条件的元素的数组。

案例1：一维数组的布尔索引

```
import numpy as np

# 创建一个一维数组
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

# 使用布尔索引筛选大于 5 的元素
bool_idx = arr > 5
print(bool_idx)  # 输出: [False False False False False  True  True  True  True
True]

# 使用布尔索引获取满足条件的元素
result = arr[bool_idx]
print(result)  # 输出: [ 6  7  8  9]
```

案例2：多维数组的布尔索引

```
import numpy as np

# 创建一个二维数组
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# 使用布尔索引筛选大于 5 的元素
bool_idx = arr > 5
print(bool_idx)
# 输出:
# [[False False False]
#  [False False  True]
#  [ True  True  True]]

# 使用布尔索引获取满足条件的元素
result = arr[bool_idx]
print(result)  # 输出: [6 7 8 9]
```

案例3：

使用逻辑运算符（如 &、|、~）组合多个条件。

```
import numpy as np

# 创建一个一维数组
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# 使用布尔索引筛选大于 5 且小于 9 的元素
bool_idx = (arr > 5) & (arr < 9)
print(bool_idx)  # 输出: [False False False False False  True  True  True False
False]

# 使用布尔索引获取满足条件的元素
result = arr[bool_idx]
print(result)  # 输出: [6 7 8]
```

逻辑运算符：

- &：与运算，组合多个条件。

- |：或运算，组合多个条件。
- ~：非运算，取反条件。

8.广播（*）

广播（Broadcast）是 numpy 对不同形状（shape）的数组进行数值计算的方式， 对数组的算术运算通常在相应的元素上进行。这要求维数相同，且各维度的长度相同，如果不相同，可以通过广播机制，这种机制的核心是对形状较小的数组，在横向或纵向上进行一定次数的重复，使其与形状较大的数组拥有相同的维度。

广播规则

1. 维度匹配：如果两个数组的维度数不同，维度数较少的数组会在前面补上长度为 1 的维度。

```
# 示例 1：维度匹配
a = np.array([[1, 2, 3], [4, 5, 6]])  # 形状: (2, 3)
b = np.array([10, 20, 30])            # 形状: (3,)
c = a + b
print(c)
# 输出:
# [[11 22 33]
#  [14 25 36]]
```

2. 形状匹配：如果两个数组在某个维度上的长度不同，但其中一个数组在该维度上的长度为 1，则该数组会沿着该维度进行广播。

```
# 示例 2：形状匹配
# 1.d是2维数组，e是一维数组，e自动广播为2为数组：[[10 20 30] [10 20 30]]，形状为(2,3)
# 2.d和e在行维度上都为2，d在列维度上是一维，自动广播为：[[1 1 1] [2 2 2]]，形状为(2,3)
d = np.array([[1], [2]])  # 形状: (2, 1)
e = np.array([10, 20, 30])  # 形状: (3,)
f = d + e
print(f)
# 输出:
# [[11 21 31]
#  [12 22 32]]
```

3. 不匹配：如果两个数组在某个维度上的长度既不相同也不为 1，则广播失败，抛出 ValueError。

案例1：一维数组与标量相加

```
import numpy as np

# 创建一个一维数组
arr = np.array([1, 2, 3])

# 标量相加
# 1自动广播为[1 1 1]
result = arr + 1

print(result)  # 输出: [2 3 4]
```

标量 1 被广播到与 arr 相同的形状，然后进行逐元素相加。

案例2：二维数组与一维数组相加

```
import numpy as np

# 创建一个二维数组
arr2d = np.array([[1, 2, 3], [4, 5, 6]])

# 创建一个一维数组
# 自动广播为[[10, 20, 30],[10, 20, 30]]
arr1d = np.array([10, 20, 30])

# 相加
result = arr2d + arr1d

print(result)
# 输出:
# [[11 22 33]
#  [14 25 36]]
```

arr1d 被广播到与 arr2d 相同的形状，然后进行逐元素相加。

```
import numpy as np

# 创建一个一维数组
arr1d = np.array([0,1,2,3])

# 创建一个二维数组
arr2d = np.array([0,1,2,3]).reshape(4,1)

# 相加
result = np.add(arr1d,arr2d)

print(result)
# 输出:
[[0 1 2 3]
 [1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]]
```

arr1d是一维数组，arr2d是一个4行1列的二维数组，按行arr1d广播为4行，按列arr2d被广播为4列，最终成为4行4列的数组

案例3：二维数组与二维数组相加

```
import numpy as np

# 创建一个二维数组
arr2d = np.array([[1, 2, 3], [4, 5, 6]])

# 创建一个形状为 (1, 3) 的二维数组
arr2d_broadcast = np.array([[10, 20, 30]])

# 相加
result = arr2d + arr2d_broadcast
```

```
print(result)
# 输出:
# [[11 22 33]
#  [14 25 36]]
```

arr2d_broadcast 被广播到与 arr2d 相同的形状，然后进行逐元素相加。

案例4：广播失败

```
import numpy as np

# 创建一个二维数组
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# 创建一个形状为 (2, 3) 的二维数组
arr2d_broadcast = np.array([[10, 20, 30], [40, 50, 60]])

# 尝试相加
try:
    result = arr2d + arr2d_broadcast
except ValueError as e:
    print(e)  # 输出: operands could not be broadcast together with shapes (3,3) (2,3)
```

9.遍历数组

9.1 遍历数组的第一维度

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])
for i in arr:
    print(i)
```

`for i in arr:` 遍历数组的**第一维度**，即按行或列的顺序逐个访问元素。

返回的是数组的子数组（如行或列），而不是单个元素。

9.2 nditer逐个访问元素

nditer 是 NumPy 中的一个强大的迭代器对象，用于高效地遍历多维数组。nditer 提供了多种选项和控制参数，使得数组的迭代更加灵活和高效。

控制参数

nditer 提供了多种控制参数，用于控制迭代的行为。

1.order 参数

order 参数用于指定数组的遍历顺序。默认情况下，nditer 按照 C 风格（行优先）遍历数组。

- **C 风格（行优先）**：order='C'
- **Fortran 风格（列优先）**：order='F'

```
# 创建一个二维数组
arr = np.array([[1, 2, 3], [4, 5, 6]])

# 使用 C 风格遍历数组
for x in np.nditer(arr, order='C'):
    print(x)
# 输出:
# 1
# 2
# 3
# 4
# 5
# 6

# 使用 Fortran 风格遍历数组
for x in np.nditer(arr, order='F'):
    print(x)
# 输出:
# 1
# 4
# 2
# 5
# 3
# 6
```

2.flags 参数

flags 参数用于指定迭代器的额外行为。

- multi_index: 返回每个元素的多维索引。
- external_loop: 返回一维数组而不是单个元素，减少函数调用的次数，从而提高性能。

```
# 创建一个三维数组
    arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

    # 使用 nditer 遍历数组并获取多维索引
    it = np.nditer(arr, flags=['multi_index'])
    for x in it:
        print(f"Element: {x}, Index: {it.multi_index}")

# 输出:
# Element: 1, Index: (0, 0, 0)
# Element: 2, Index: (0, 0, 1)
# Element: 3, Index: (0, 1, 0)
# Element: 4, Index: (0, 1, 1)
# Element: 5, Index: (1, 0, 0)
# Element: 6, Index: (1, 0, 1)
# Element: 7, Index: (1, 1, 0)
# Element: 8, Index: (1, 1, 1)


# 创建一个二维数组
arr = np.array([[1, 2, 3], [4, 5, 6]])
# 使用外部循环遍历数组，列优先
for x in np.nditer(arr, flags=['external_loop'], order='F'):
```

```
    print(x)
# 输出:
# [1 4]
# [2 5]
# [3 6]
# 注意：external_loop 和 order='F' 一起使用时，会将每列的元素打包成一个数组返回。

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])
# 使用外部循环遍历数组，行优先
for x in np.nditer(arr, flags=['external_loop'], order='C'):
    print(x)
# 输出:
[1 2 3 4 5 6]
# 注意：external_loop 会将所有元素打包成一个数组返回，而不是按行打包。
```

3.op_flags 参数

op_flags 参数用于指定操作数的行为。

- readonly: 只读操作数。
- readwrite: 读写操作数。
- writeonly: 只写操作数。

```
# 创建一个二维数组
arr = np.array([[1, 2, 3], [4, 5, 6]])

# 使用读操作数遍历数组
for x in np.nditer(arr, op_flags=['readonly']):
    # 不能对数据修改，否则报错：ValueError: assignment destination is read-only
    # x[...] = x * 2
    print(x)

# 使用读写操作数遍历数组
for x in np.nditer(arr, op_flags=['readwrite']):
    x[...] = 2 * x

print(arr)
# 输出:
# [[ 2  4  6]
#  [ 8 10 12]]
```

x[...]：是 NumPy 中的一种索引方式，表示获取数组 `x` 的一个视图，对数组 `x` 进行原地修改，而不是创建一个新的数组。

返回一个可写视图（writable view），允许直接修改原数组中的元素。

注意：当你使用 x[...] 时，x 必须是一个数组或数组的视图，而不是一个标量。

示例1：修改单个元素

```
import numpy as np

# 创建一个数组
arr = np.array([1, 2, 3])

# 使用 x[...] 修改单个元素
x = arr[0]
x[...] = 100
print(arr)

#输出:
TypeError: 'numpy.int64' object does not support item assignment
```

此时x是一个标量，不能使用x[...] 修改，可以直接赋值：

```
import numpy as np

# 创建一个数组
arr = np.array([1, 2, 3])

# 使用 x[...] 修改单个元素
arr[0] = 100
print(arr)
```

示例2：在迭代中修改元素

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])
# 使用读写操作数遍历数组
# x[...] = x * 2 将数组 arr 中的每个元素乘以 2，并写回到原数组中。
# x实际返回一个包含当前元素的数组，而不是标量，所以可以使用x[...]
for x in np.nditer(arr, op_flags=['readwrite']):
    x[...] = 2 * x

print(arr)
# 输出:
# [[ 2  4  6]
#  [ 8 10 12]]
```

10.数组操作

10.1 数组变维

函数名称	函数介绍
reshape	在不改变数组元素的条件下，修改数组的形状
flat属性	返回是一个迭代器，可以用 for 循环遍历其中的每一个元素
flatten	以一维数组的形式返回一份数组的副本，对副本的操作不会影响到原数组
ravel	返回一个连续的扁平数组（即展开的一维数组），与 flatten不同，它返回的是数组视图（修改视图会影响原数组）

10.1.1 flat

返回一个一维迭代器，用于遍历数组中的所有元素。无论数组的维度如何，ndarray.flat属性都会将数组视为一个扁平化的一维数组，按行优先的顺序遍历所有元素。

语法：

```
ndarray.flat
```

案例：

```
import numpy as np

def flat_test():
    array_one = np.arange(4).reshape(2,2)
    print("原数组元素：")
    for i in array_one:
        print(i,end=" ")
    print()
    print("使用flat属性，遍历数组：")
    for i in array_one.flat:
        print(i,end=" ")
```

10.1.2 flatten()

用于将多维数组转换为一维数组。flatten() 返回的是原数组的一个拷贝，因此对返回的数组进行修改不会影响原数组。

语法：

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

参数

order: 指定数组的展开顺序。

- `'C'`：按行优先顺序展开（默认）。
- `'F'`：按列优先顺序展开。
- `'A'`：如果原数组是 Fortran 连续的，则按列优先顺序展开；否则按行优先顺序展开。
- `'K'`：按元素在内存中的顺序展开。

案例：

```python
import numpy as np

# 创建一个二维数组
arr = np.array([[1, 2, 3], [4, 5, 6]])

# 使用 flatten 方法按行优先顺序展开
flat_arr = arr.flatten(order='C')

print(flat_arr)
# 输出:
# [1 2 3 4 5 6]
```

如果order='F'，输出结果是什么?

10.1.3 ravel()

用于将多维数组转换为一维数组。与 flatten() 不同，ravel() 返回的是原数组的一个视图（view），而不是副本。因此，对返回的数组进行修改会影响原数组。

语法:

```python
ndarray.ravel()
```

参数

order: 指定数组的展开顺序。

- `'C'`：按行优先顺序展开（默认）。
- `'F'`：按列优先顺序展开。
- `'A'`：如果原数组是 Fortran 连续的，则按列优先顺序展开；否则按行优先顺序展开。
- `'K'`：按元素在内存中的顺序展开。

案例:

```python
import numpy as np

# 创建一个二维数组
arr = np.array([[1, 2, 3], [4, 5, 6]])

# 使用 ravel 方法按行优先顺序展开
ravel_arr = arr.ravel()

print(ravel_arr)
# 输出:
# [1 2 3 4 5 6]

ravel_arr[-1] = 7
print(arr)
# 输出:
# [[1 2 3]
#  [4 5 7]]
```

10.2 数组转置

函数名称	说明
transpose	将数组的维度值进行对换，比如二维数组维度(2,4)使用该方法后为(4,2)
ndarray.T	与 transpose 方法相同

案例：

```
import numpy as np

def transpose_test():
    array_one = np.arange(12).reshape(3, 4)
    print("原数组：")
    print(array_one)
    print("使用transpose()函数后的数组：")
    print(np.transpose(array_one))


def T_test():
    array_one = np.arange(12).reshape(3, 4)
    print("原数组：")
    print(array_one)
    print("数组转置：")
    print(array_one.T)
```

10.3 修改数组维度

多维数组（也称为 ndarray）的维度（或轴）是从外向内编号的。这意味着最外层的维度是轴0，然后是轴1，依此类推。

函数名称	参数	说明
expand_dims(arr, axis)	arr：输入数组 axis：新轴插入的位置	在指定位置插入新的轴(**相对于结果数组而言**)，从而扩展数组的维度
squeeze(arr, axis)	arr：输入数的组 axis：取值为整数或整数元组，用于指定需要删除的维度所在轴，指定的维度值必须为 1 ，否则将会报错，若为 None，则删除数组维度中所有为 1 的项	删除数组中维度为 1 的项

案例1：增加数组维度

```
import numpy as np

# 创建一个一维数组
a = np.array([1, 2, 3])
print(a.shape)  # 输出: (3,)

# 在第 0 维插入新维度
b = np.expand_dims(a, axis=0)
```

```
print(b)
# 输出:
# [[1 2 3]]
print(b.shape)  # 输出: (1, 3)

# 在第 1 维插入新维度
c = np.expand_dims(a, axis=1)
print(c)
# 输出:
# [[1]
#  [2]
#  [3]]
print(c.shape)  # 输出: (3, 1)
```

案例2：适应广播操作

```
# 创建一个一维数组
a = np.array([1, 2, 3])

# 创建一个标量
b = 2

# 扩展 a 的维度
c = np.expand_dims(a, axis=0)
print(c)
# 输出:
# [[1 2 3]]

# 进行广播操作
d = c + b
print(d)
# 输出:
# [[3 4 5]]
```

案例3：移除长度为1的维度

```
import numpy as np

# 创建一个数组
c = np.array([[[1, 2, 3]]])
print(c.shape)  # 输出: (1, 1, 3)

# 移除第 0 维
d = np.squeeze(c, axis=0)
print(d)
# 输出:
# [[1 2 3]]
print(d.shape)  # 输出: (1, 3)

# 移除第 1 维
e = np.squeeze(c, axis=1)
print(e)
# 输出:
# [[1 2 3]]
```

```
print(e.shape)  # 输出: (1, 3)

# 移除第 2 维
f = np.squeeze(c, axis=2)
print(f)
# 输出:
# ValueError: cannot select an axis to squeeze out which has size not equal to
one
print(f.shape)
```

```
import numpy as np

# 创建一个数组
g = np.array([[[1, 2, 3]], [[4, 5, 6]]])
print(g.shape)  # 输出: (2, 1, 3)

# 移除第 1 维
h = np.squeeze(g, axis=1)
print(h)
# 输出:
# [[1 2 3]
#  [4 5 6]]
print(h.shape)  # 输出: (2, 3)
```

np.newaxis 是 NumPy 中的一个特殊索引，用于在数组的指定位置插入一个新的轴（维度）。它通常用于将一维数组转换为二维数组，或者在其他情况下增加数组的维度。

```
import numpy as np

# 创建一个一维数组
arr = np.array([1, 2, 3, 4])
print("原始数组:", arr)

# 使用 np.newaxis 在行上插入新轴
arr_row = arr[np.newaxis, :]
print("行向量:", arr_row)

# 使用 np.newaxis 在列上插入新轴
arr_col = arr[:, np.newaxis]
print("列向量:", arr_col)

#输出:
原始数组: [1 2 3 4]
行向量: [[1 2 3 4]]
列向量: [[1]
         [2]
         [3]
         [4]]
```

10.4 连接数组

函数名称	参数	说明
hstack(tup)	tup：可以是元组，列表，或者numpy数组，返回结果为numpy的数组	按水平顺序堆叠序列中数组（列方向）
vstack(tup)	tup：可以是元组，列表，或者numpy数组，返回结果为numpy的数组	按垂直方向堆叠序列中数组（行方向）

hstack函数要求堆叠的数组在垂直方向（行）上具有相同的形状。如果行数不一致，hstack() 将无法执行，并会抛出 ValueError 异常。

vstack() 要求堆叠的数组在水平方向（列）上具有相同的形状。如果列数不一致，将无法执行堆叠操作。

vstack() 和 hstack() 要求堆叠的数组在某些维度上具有相同的形状。如果维度不一致，将无法执行堆叠操作。

案例：

hstack：

```
import numpy as np

# 创建两个形状不同的数组，行数一致
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5], [6]])
print(arr1.shape)   # (2, 2)
print(arr2.shape)   # (2, 1)

# 使用 hstack 水平堆叠数组
result = np.hstack((arr1, arr2))
print(result)
# 输出：
# [[1 2 5]
#  [3 4 6]]
```

```
# 创建两个形状不同的数组，行数不一致
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5], [6], [7]])
print(arr1.shape)   # (2, 2)
print(arr2.shape)   # (3, 1)

# 使用 hstack 水平堆叠数组
result = np.hstack((arr1, arr2))

print(result)

# ValueError: all the input array dimensions except for the concatenation axis
must match exactly
# 第一个数组在第0维有2个元素，而第二个数组在第0维有3个元素，因此无法直接连接。
```

vstack：

```python
# 创建两个一维数组
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# 使用 vstack 垂直堆叠数组
result = np.vstack((arr1, arr2))

print(result)
# 输出:
# [[1 2 3]
#  [4 5 6]]
```

```python
import numpy as np
# 创建两个形状不同的数组，列数一致
arr1 = np.array([[1, 2, 3], [4, 5, 6]])
arr2 = np.array([[7, 8, 9], [10, 11, 12]])

# 使用 vstack 垂直堆叠数组
result = np.vstack((arr1, arr2))

print(result)
```

```python
# 创建两个形状不同的数组，列数不一致
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6, 7], [8, 9, 10]])

# 使用 vstack 垂直堆叠数组
result = np.vstack((arr1, arr2))

print(result)
# ValueError: all the input array dimensions except for the concatenation axis must match exactly
# 第一个数组在第1维有2个元素，而第二个数组在第1维有3个元素，因此无法直接连接。
```

10.5 分割数组

函数名称	参数	说明
hsplit(ary, indices_or_sections)	ary：原数组 indices_or_sections：按列分割的索引位置	将一个数组水平分割为多个子数组（按列）
vsplit(ary, indices_or_sections)	ary：原数组 indices_or_sections：按行分割的索引位置	将一个数组垂直分割为多个子数组（按行）

案例：

```python
import numpy as np

'''
    hsplit 函数：
```

```
        1、将一个数组水平分割为多个子数组（按列）
        2、ary：原数组
        3、indices_or_sections：按列分割的索引位置
'''
# 创建一个二维数组
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])

# 使用 np.hsplit 将数组分割成三个子数组
# 分割点在索引1和3处，这意味着：
# 第一个子数组将包含从第0列到索引1（不包括索引1）的列，即第0列。
# 第二个子数组将包含从索引1（包括索引1）到索引3（不包括索引3）的列，即第1列到第2列。
# 第三个子数组将包含从索引3（包括索引3）到末尾的列，即第3列。
result = np.hsplit(arr, [1, 3])

# 查看结果
print("第一个子数组:\n", result[0])  # 输出包含第0列的子数组
print("第二个子数组:\n", result[1])  # 输出包含第1列和第2列的子数组
print("第三个子数组:\n", result[2])  # 输出包含第3列的子数组


'''
    vsplit 函数：
        1、将一个数组垂直分割为多个子数组（按行）
        2、ary：原数组
        3、indices_or_sections：按列分割的索引位置
'''
array_one = np.arange(12).reshape(2,6)
print('array_one 原数组：\n', array_one)
array_two = np.vsplit(array_one,[1])
print('vsplit 之后的数组：\n', array_two)
```

10.6 矩阵运算

np.dot

是一个通用的点积函数，适用于多种维度的数组。

- 对于二维数组（矩阵），`np.dot` 等价于矩阵乘法。
- 对于一维数组（向量），`np.dot` 计算的是向量的点积（内积）。

案例1：矩阵运算

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])

result = np.dot(a, b)
print(result)
```

案例2：向量点积

计算公式为：

$$a \cdot b = a_1b_1 + a_2b_2 + \cdots + a_nb_n$$

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

result = np.dot(a, b)
print(result)
```

np.matmul

是专门用于矩阵乘法的函数，适用于二维及更高维度的数组。

案例：矩阵相乘

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

result = np.matmul(a, b)
print(result)
```

np.dot是通用点积函数，np.matmul专门用于矩阵运算，性能要好于np.dot

np.linalg.det

计算一个方阵（行数和列数相等的矩阵）的行列式。

案例：

```
a = np.array([[1, 2], [3, 4]],dtype=int)
# 计算行列式
det_a = np.linalg.det(a)
print(det_a)

#输出：
-2.0000000000000004
```

结果不是-2，这是由于浮点数的二进制表示和计算过程中的舍入误差导致的。可以通过四舍五入来近似表示：

```
det_a = np.round(np.linalg.det(a))
```

11.数组元素的增删改查

11.1 resize

函数名称	参数	说明
resize(a, new_shape)	a：操作的数组 new_shape：返回的数组的形状，如果元素数量不够，重复数组元素来填充新的形状	返回指定形状的新数组

案例：

```
import numpy as np

array_one = np.arange(6).reshape(2, 3)
print(array_one)
print('resize 后数组：\n', np.resize(array_one, (3, 4)))

# 输出：
# [[0 1 2 3]
#  [4 5 0 1]
#  [2 3 4 5]]
```

最后一行代码将数组形状修改为(3, 4)，原数组的元素数量不够，则重复原数组的元素填充。

11.2 append

函数名称	参数	说明
append(arr, values, axis=None)	arr：输入的数组 values：向 arr 数组中添加的值，需要和 arr 数组的形状保持一致 axis：默认为 None，返回的是一维数组；当 axis =0 时，追加的值会被添加到行，而列数保持不变，若 axis=1 则与其恰好相反	在数组的末尾添加值，返回一个一维数组

案例：

```
'''
    append(arr, values, axis=None) 函数：
        1、将元素值添加到数组的末尾，返回一个一维数组
        2、arr：输入的数组
        3、values：向 arr 数组中添加的值，需要和 arr 数组的形状保持一致
        4、axis：默认为 None，返回的是一维数组；当 axis =0 时，追加的值会被添加到行，而列数保持不变，若 axis=1 则与其恰好相反
'''
def append_test():
    array_one = np.arange(6).reshape(2,3)
    print('原数组：\n', array_one)
    array_two = np.append(array_one,[[1,1,1],[1,1,1]],axis=None)
    print('append 后数组 axis=None：\n', array_two)
    array_three = np.append(array_one, [[1, 1, 1], [1, 1, 1]], axis=0)
    print('append 后数组 axis=0：\n', array_three)
    array_three = np.append(array_one, [[1, 1, 1], [1, 1, 1]], axis=1)
    print('append 后数组 axis=1：\n', array_three)
```

11.3 insert

函数名称	参数	说明
insert(arr, obj, values, axis)	arr：输入的数组 obj：表示索引值，在该索引值之前插入 values 值 values：要插入的值 axis：默认为 None，返回的是一维数组；当 axis =0 时，追加的值会被添加到行，而列数保持不变，若 axis=1 则与其恰好相反	沿规定的轴将元素值插入到指定的元素前

案例：

```
import numpy as np

def insert_test():
    array_one = np.arange(6).reshape(2,3)
    print('原数组：\n', array_one)
    array_two = np.insert(array_one, 1, [6],axis=None)
    print('insert 后数组 axis=None：\n', array_two)
    # 在索引为1的行插入[6],并自动广播
    array_three = np.insert(array_one,1, [6], axis=0)
    print('insert 后数组 axis=0：\n', array_three)
    # 在索引为1的列插入[6],并自动广播
    array_three = np.insert(array_one, 1, [6], axis=1)
    print('insert 后数组 axis=1：\n', array_three)

    array_three = np.insert(array_one, 1, [6,7], axis=1)
    print('insert 后数组 axis=1：\n', array_three)

    # 在列上插入数组的形状和原数组列的形状不一致
    # ValueError: could not broadcast input array from shape (3,1) into shape
(2,1)
    array_three = np.insert(array_one, 1, [6,7,8], axis=1)
    print('insert 后数组 axis=1：\n', array_three)
```

如果obj为-1，表示插入在倒数第一个元素之前，**不是在最后一列。**

```
array_three = np.insert(array_one, -1, [6,7], axis=1)
print('insert 后数组 axis=1：\n', array_three)

#输出：
 [[0 1 6 2]
 [3 4 7 5]]
```

11.4 delete

函数名称	参数	说明
delete(arr, obj, axis)	arr：输入的数组 obj：表示索引值，要删除数据的索引 axis：默认为 None，返回的是一维数组；当 axis =0 时，删除指定的行，若 axis=1 则与其恰好相反	删掉某个轴上的子数组，并返回删除后的新数组

案例：

一维数组：

```
import numpy as np

# 创建一个 NumPy 数组
arr = np.array([1, 2, 3, 4, 5, 6])

# 删除索引为 2 和 4 的元素
new_arr = np.delete(arr, [2, 4])

print(new_arr)
```

二维数组：

```
import numpy as np


def delete_test():
    array_one = np.arange(6).reshape(2,3)
    print('原数组：\n', array_one)
    array_two = np.delete(array_one,1,axis=None)
    print('delete 后数组 axis=None：\n', array_two)
    array_three = np.delete(array_one,1, axis=0)
    print('delete 后数组 axis=0：\n', array_three)
    array_three = np.delete(array_one, 1, axis=1)
    print('delete 后数组 axis=1：\n', array_three)
```

11.5 argwhere

返回数组中非 0 元素的索引，若是多维数组则返回行、列索引组成的**索引坐标**

案例：

```
import numpy as np

'''
    argwhere(a) 函数：
        1、返回数组中非 0 元素的索引，若是多维数组则返回行、列索引组成的索引坐标
'''
def argwhere_test():
    array_one = np.arange(6).reshape(2,3)
    print('原数组：\n', array_one)
    print('argwhere 返回非0元素索引：\n', np.argwhere(array_one))
    print('argwhere 返回所有大于 1 的元素索引：\n', np.argwhere(array_one > 1))
```

11.6 unique

函数名称	参数	说明
unique(ar, return_index=False, return_inverse=False, return_counts=False, axis=None)	ar：输入的数组 return_index：如果为 True，则返回新数组元素在原数组中的位置（索引） return_inverse：如果为 True，则返回原数组元素在新数组中的位置（逆索引） return_counts：如果为 True，则返回去重后的数组元素在原数组中出现的次数	删掉某个轴上的子数组，并返回删除后的新数组

案例1：返回唯一元素的索引

```
import numpy as np

# 创建一个 NumPy 数组
arr = np.array([1, 2, 2, 3, 4, 4, 5])
unique_elements, indices = np.unique(arr, return_index=True)
print(unique_elements)
print(indices)
```

案例2：返回唯一元素及其逆索引

```
mport numpy as np

# 创建一个一维数组
arr = np.array([1, 2, 2, 3, 3, 3, 4, 4, 4, 4])

# 使用 np.unique 查找唯一元素及其逆索引
unique_elements, inverse_indices = np.unique(arr, return_inverse=True)

print(unique_elements)
# 输出:
# [1 2 3 4]

print(inverse_indices)
# 输出:
# [0 1 1 2 2 2 3 3 3 3]
# 逆索引数组，表示原始数组中的每个元素在唯一元素数组中的位置。
```

案例3：返回唯一元素的计数

```
import numpy as np

# 创建一个一维数组
```

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

# 使用 np.unique 查找唯一元素及其计数
unique_elements, counts = np.unique(arr, return_counts=True)

print(unique_elements)
# 输出:
# [1 2 3 4]

print(counts)
# 输出:
# [1 2 3 4]
```

对于多维数组，unique 函数同样适用。默认情况下，unique 函数会将多维数组展平为一维数组，然后查找唯一元素。

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

# 查找数组中的唯一元素
unique_elements = np.unique(arr)
print(unique_elements)
```

12.统计函数

12.1 amin() 和 amax()

- 计算数组沿指定轴的最小值与最大值，并以数组形式返回
- 对于二维数组来说，axis=1 表示沿着水平方向，axis=0 表示沿着垂直方向

案例:

```
'''
    numpy.amin() 和 numpy.amax() 函数:
        1、计算数组沿指定轴的最小值与最大值，并以数组形式返回
        2、对于二维数组来说，axis=1 表示沿着水平方向，axis=0 表示沿着垂直方向
'''
def amin_amax_test():
    array_one = np.array([[1,23,4,5,6],[1,2,333,4,5]])
    print('原数组元素：\n', array_one)
    print('原数组水平方向最小值：\n', np.amin(array_one, axis=1))
    print('原数组水平方向最大值：\n', np.amax(array_one, axis=1))
    print('原数组垂直方向最小值：\n', np.amin(array_one, axis=0))
    print('原数组垂直方向最大值：\n', np.amax(array_one, axis=0))
```

输出:

```
原数组元素：
 [[  1  23   4   5   6]
 [  1   2 333   4   5]]
原数组水平方向最小值：
 [1 1]
原数组水平方向最大值：
 [ 23 333]
原数组垂直方向最小值：
 [1 2 4 4 5]
原数组垂直方向最大值：
 [  1  23 333   5   6]
```

按1轴求最小值，表示在最内层轴中（每列中）分别找最小值；按1轴求最小值表示在最外层轴中（所有行中按列）找最小值。求最大值类似。

12.2 ptp()

- 计算数组元素中最值之差值，即最大值 - 最小值
- 对于二维数组来说，axis=1 表示沿着水平方向，axis=0 表示沿着垂直方向

```python
# 创建一个二维数组
arr = np.array([[1, 2, 3], [4, 5, 6]])

# 使用 np.ptp 计算峰峰值
ptp_value = np.ptp(arr)

print(ptp_value)
# 输出:
# 5

# 使用 np.ptp 按行计算峰峰值
ptp_values_row = np.ptp(arr, axis=1)

# 使用 np.ptp 按列计算峰峰值
ptp_values_col = np.ptp(arr, axis=0)

print(ptp_values_row)
# 输出:
# [2 2]

print(ptp_values_col)
# 输出:
# [3 3 3]
```

12.3 median()

用于计算中位数，中位数是指将数组中的数据按从小到大的顺序排列后，位于中间位置的值。如果数组的长度是偶数，则中位数是中间两个数的平均值。

```python
# 创建一个二维数组
arr = np.array([[1, 2, 3], [4, 5, 6]])

# 使用 np.median 计算中位数
```

```
median_value = np.median(arr,axis=None)

print(median_value)
# 输出:
# 3.5

# 使用 np.median 按行计算中位数
median_values_row = np.median(arr, axis=1)

# 使用 np.median 按列计算中位数
median_values_col = np.median(arr, axis=0)

print(median_values_row)
# 输出:
# [2. 5.]

print(median_values_col)
# 输出:
# [2.5 3.5 4.5]
```

12.4 mean()

沿指定的轴，计算数组中元素的算术平均值（即元素之总和除以元素数量）

```
# 创建一个一维数组
arr = np.array([1, 2, 3, 4, 5])

# 使用 np.mean 计算平均值
mean_value = np.mean(arr)

print(mean_value)
# 输出:
# 3.0


# 创建一个二维数组
arr = np.array([[1, 2, 3], [4, 5, 6]])

# 使用 np.mean 计算平均值
mean_value = np.mean(arr)

print(mean_value)
# 输出:
# 3.5


# 使用 np.mean 按行计算平均值
mean_values_row = np.mean(arr, axis=1)

# 使用 np.mean 按列计算平均值
mean_values_col = np.mean(arr, axis=0)

print(mean_values_row)
# 输出:
# [2. 5.]
```

```
print(mean_values_col)
# 输出:
# [2.5 3.5 4.5]
```

6、average()

加权平均值是将数组中各数值乘以相应的权数，然后再对权重值求总和，最后以权重的总和除以总的单位数（即因子个数）；根据在数组中给出的权重，计算数组元素的加权平均值。该函数可以接受一个轴参数 axis，如果未指定，则数组被展开为一维数组。

$$\text{加权平均值} = \frac{\sum_{i=1}^{n}(x_i \cdot w_i)}{\sum_{i=1}^{n} w_i}$$

其中 xi是数组中的元素，wi是对应的权重。

如果所有元素的权重之和等于1，则表示为数学中的期望值。

```
# 创建一个一维数组
    arr = np.array([1, 2, 3, 4, 5])

    # 创建权重数组
    weights = np.array([0.1, 0.2, 0.3, 0.2, 0.2])

    # 使用 np.average 计算加权平均值
    average_value = np.average(arr, weights=weights)

    print(average_value)
    # 输出:
    # 3.2
```

7、var()

在 NumPy 中，计算方差时使用的是统计学中的方差公式，而不是概率论中的方差公式，主要是因为 NumPy 的设计目标是处理实际数据集，而不是概率分布。

np.var 函数默认计算的是总体方差（Population Variance），而不是样本方差（Sample Variance）。

总体方差：

对于一个总体数据集 X={x1,x2,...,xN}，总体方差的计算公式为：

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

其中：

- N是总体数据点的总数。
- μ是总体的均值。

```python
# 创建一个数组
arr = np.array([1, 2, 3, 4, 5])

# 计算方差
variance = np.var(arr)

print(variance)

#输出：2
```

样本方差：

对于一个样本数据集 X={x1,x2,...,xn}，样本方差 的计算公式为：

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

其中：

- n是样本数据点的总数。
- x¯是样本的均值。

在样本数据中，样本均值的估计会引入一定的偏差。通过使用 n−1作为分母，可以校正这种偏差，得到更准确的总体方差估计。

```python
# 创建一个数组
arr = np.array([1, 2, 3, 4, 5])

# 计算方差
variance = np.var(arr, ddof=1)

print(variance)

#输出：2.5
```

8、std()

标准差是方差的算术平方根，用来描述一组数据平均值的分散程度。若一组数据的标准差较大，说明大部分的数值和其平均值之间差异较大；若标准差较小，则代表这组数值比较接近平均值

```python
# 创建一个数组
arr = np.array([1, 2, 3, 4, 5])

# 计算标准差
std_dev = np.std(arr)

print(std_dev)

# 输出：1.4142135623730951
```