

# Numpy教程



## 1 常量

- 1.1 numpy.nan
- 1.2 numpy.inf
- 1.3 numpy.pi
- 1.4 numpy.e

#### 2 数据类型

- 2.1 常见数据类型
- 2.2 创建数据类型
- 2.3 数据类型信息

#### 3 时间日期和时间增量

- 3.1 datetime64 基础
- 3.2 datetime64 和 timedelta64 运算
- 3.3 datetime64 的应用

#### 4 数组的创建

- 4.1 1. 依据现有数据来创建 ndarray
  - 4.1.1 (a) 通过array()函数进行创建。
  - 4.1.2 (b) 通过asarray()函数进行创建
  - 4.1.3 (c) 通过fromfunction()函数进行创建
- 4.2 2. 依据 ones 和 zeros 填充方式
  - 4.2.1 (a) 零数组
  - 4.2.2 (b) 1数组
  - 4.2.3 (c) 空数组
  - 4.2.4 (d) 单位数组
  - 4.2.5 (e) 对角数组
  - 4.2.6 (f) 常数数组
- 4.3 3. 利用数值范围来创建ndarray
- 4.4 4. 结构数组的创建
  - 4.4.1 (a) 利用字典来定义结构
  - 4.4.2 (b) 利用包含多个元组的列表来定义结构
- 5 数组的属性
- 6 副本与视图
- 7 索引与切片
- 7.1 整数索引
- 7.2 切片索引
- 7.3 dots 索引
- 7.4 整数数组索引
- 7.5 布尔索引
- 8 数组迭代
- 9 数组操作
  - 9.1 更改形状
- 9.2 数组转置
- 9.3 更改维度
- 9.4 数组组合
- 9.5 数组拆分
- 9.6 数组平铺
- 9.7 添加和删除元素
- 10 向量化和广播
- 11 数学函数
- 11.1 算数运算
  - 11.1.1 numpy.add

- 11.1.2 numpy.subtract
- 11.1.3 numpy.multiply
- 11.1.4 numpy.divide
- 11.1.5 numpy.floor\_divide
- 11.1.6 numpy.power
- 11.1.7 numpy.sqrt
- 11.1.8 numpy.square
- 11.2 三角函数
- 11.2.1 numpy.sin
- 11.2.2 numpy.cos
- 11.2.3 numpy.tan
- 11.2.4 numpy.arcsin
- 11.2.5 numpy.arccos
- 11.2.6 numpy.arctan
- 11.3 指数和对数
- 11.3.1 numpy.exp
- 11.3.2 numpy.log
- 11.3.3 numpy.exp2
- 11.3.4 numpy.log2
- 11.3.5 numpy.log10
- 11.4 加法函数、乘法函数
  - 11.4.1 numpy.sum
  - 11.4.2 numpy.cumsum
  - 11.4.3 numpy.prod 乘积
  - 11.4.4 numpy.cumprod 累乘
- 11.4.5 numpy.diff 差值
- 11.5 四舍五入
  - 11.5.1 numpy.around 舍入
  - 11.5.2 numpy.ceil 上限
  - 11.5.3 numpy.floor 下限
- 11.6 杂项
- 11.6.1 numpy.clip 裁剪
- 11.6.2 numpy.absolute 绝对值
- 11.6.3 numpy.abs
- 11.6.4 numpy.sign 返回数字符号的逐元素指示

#### 12 逻辑函数

- 12.1 真值测试
- 12.1.1 numpy.all
- 12.1.2 numpy.any
- 12.2 数组内容
- 12.2.1 numpy.isnan
- 12.3 逻辑运算
  - 12.3.1 numpy.logical\_not
  - 12.3.2 numpy.logical\_and
  - 12.3.3 numpy.logical\_or
  - 12.3.4 numpy.logical\_xor
- 12.4 对照
  - 12.4.1 numpy.greater
- 12.4.2 numpy.greater\_equal
- 12.4.3 numpy.equal
- 12.4.4 numpy.not\_equal
- 12.4.5 numpy.less

- 12.4.6 numpy.less\_equal
- 12.4.7 numpy.isclose
- 12.4.8 numpy.allclose

## 13 排序,搜索和计数

- 13.1 排序
- 13.2 搜索
- 13.3 计数

## 14 集合操作

- 14.1 构造集合
- 14.2 布尔运算
- 14.2.1 求两个集合的交集:
- 14.2.2 求两个集合的并集:
- 14.2.3 求两个集合的差集:
- 14.2.4 求两个集合的异或:

# 1 常量

# 1.1 numpy.nan

1. 表示空值。

```
1 nan = NaN = NAN
```

【例】两个 numpy.nan 是不相等的。

```
import numpy as np

print(np.nan == np.nan) # False
print(np.nan != np.nan) # True
```

1. numpy.isnan(x, \*args, \*\*kwargs) Test element-wise for NaN and return result as a boolean array.

【例】

```
import numpy as np
 2
 3
    x = np.array([1, 1, 8, np.nan, 10])
    # [ 1. 1. 8. nan 10.]
 6
    y = np.isnan(x)
 8
    print(y)
    # [False False True False]
9
10
11
    z = np.count_nonzero(y)
    print(z) # 1
12
```

# 1.2 numpy.inf

1. 表示正无穷大。

```
1 Inf = inf = infty = Infinity = PINF
```

# 1.3 numpy.pi

1. 表示圆周率

1 pi = 3.1415926535897932384626433...

# 1.4 numpy.e

1. 表示自然常数

e = 2.71828182845904523536028747135266249775724709369995...

# 2 数据类型

# 2.1 常见数据类型

Python 原生的数据类型相对较少, bool、int、float、str等。这在不需要关心数据在计算机中表示的所有方式的应用中是方便的。然而,对于科学计算,通常需要更多的控制。为了加以区分 numpy 在这些类型名称末尾都加了"\_"。

下表列举了常用 numpy 基本类型。

ŧ	类型	备注	说明
ŧ	bool_ = bool8	8位	布尔类型
ŧ	int8 = byte	8位	整型
ŧ	int16 = short	16位	整型
ŧ	int32 = intc	32位	整型
ŧ	$int_{=} int64 = long = int0 = intp$	64位	整型
ŧ	uint8 = ubyte	8位	无符号整型
ŧ	uint16 = ushort	16位	无符号整型
ŧ	uint32 = uintc	32位	无符号整型
ŧ	uint64 = uintp = uint0 = uint	64位	无符号整型
ŧ	float16 = half	16位	浮点型
ŧ	float32 = single	32位	浮点型
ŧ	float_ = float64 = double	64位	浮点型
ŧ	str_ = unicode_ = str0 = unicode		Unicode 字符串
ŧ	datetime64		日期时间类型
ŧ	timedelta64		表示两个时间之间的间隔

# 2.2 创建数据类型

numpy 的数值类型实际上是 dtype 对象的实例。

```
1 class dtype(object):
2   def __init__(self, obj, align=False, copy=False):
3   pass
```

ŧ	字符	对应类型	备注
ŧ	b	boolean	'b1'
ŧ	i	signed integer	'i1', 'i2', 'i4', 'i8'
ŧ	u	unsigned integer	'u1', 'u2' ,'u4' ,'u8'
ŧ	f	floating-point	'f2', 'f4', 'f8'
ŧ	с	complex floating-point	
ŧ	m	timedelta64	表示两个时间之间的间隔
ŧ	M	datetime64	日期时间类型
ŧ	0	object	
ŧ	S	(byte-)string	S3表示长度为3的字符串
ŧ	U	Unicode	Unicode 字符串
ŧ	v	void	

```
1
     import numpy as np
2
3
     a = np.dtype('b1')
4
     print(a.type) # <class 'numpy.bool_'>
5
     print(a.itemsize) # 1
6
7
     a = np.dtype('i1')
     print(a.type) # <class 'numpy.int8'>
8
9
     print(a.itemsize) # 1
     a = np.dtype('i2')
10
     print(a.type) # <class 'numpy.int16'>
11
12
     print(a.itemsize) # 2
     a = np.dtype('i4')
13
     print(a.type) # <class 'numpy.int32'>
14
15
     print(a.itemsize) # 4
     a = np.dtype('i8')
16
17
     print(a.type) # <class 'numpy.int64'>
18
     print(a.itemsize) # 8
19
20
     a = np.dtype('u1')
21
     print(a.type) # <class 'numpy.uint8'>
22
     print(a.itemsize) # 1
     a = np.dtype('u2')
23
     print(a.type) # <class 'numpy.uint16'>
24
     print(a.itemsize) # 2
25
     a = np.dtype('u4')
26
     print(a.type) # <class 'numpy.uint32'>
27
28
     print(a.itemsize) # 4
29
     a = np.dtype('u8')
    print(a.type) # <class 'numpy.uint64'>
30
31
     print(a.itemsize) # 8
32
```

```
33
     a = np.dtype('f2')
34
     print(a.type) # <class 'numpy.float16'>
     print(a.itemsize) # 2
35
36
     a = np.dtype('f4')
     print(a.type) # <class 'numpy.float32'>
37
     print(a.itemsize) # 4
38
     a = np.dtype('f8')
39
40
     print(a.type) # <class 'numpy.float64'>
     print(a.itemsize) # 8
41
42
43
     a = np.dtype('S')
44
     print(a.type) # <class 'numpy.bytes_'>
     print(a.itemsize) # 0
45
     a = np.dtype('S3')
47
     print(a.type) # <class 'numpy.bytes_'>
     print(a.itemsize) # 3
48
49
50
     a = np.dtype('U3')
     print(a.type) # <class 'numpy.str_'>
51
     print(a.itemsize) # 12
52
```

# 2.3 数据类型信息

Python 的浮点数通常是64位浮点数,几乎等同于 np.float64。

NumPy和Python整数类型的行为在整数溢出方面存在显着差异,与 NumPy 不同,Python 的 **int** 是灵活的。这意味着Python整数可以扩展以容纳任何整数并且不会溢出。

Machine limits for integer types.

```
1  class iinfo(object):
2   def __init__(self, int_type):
3     pass
4   def min(self):
5     pass
6   def max(self):
7   pass
```

#### 【例】

```
import numpy as np

ii16 = np.iinfo(np.int16)
print(ii16.min) # -32768
print(ii16.max) # 32767

ii32 = np.iinfo(np.int32)
print(ii32.min) # -2147483648
print(ii32.max) # 2147483647
```

Machine limits for floating point types.

```
class finfo(object):
def _init(self, dtype):
```

```
import numpy as np
 2
 3
    ff16 = np.finfo(np.float16)
    print(ff16.bits) # 16
 4
    print(ff16.min) # -65500.0
 5
    print(ff16.max) # 65500.0
 6
    print(ff16.eps) # 0.000977
 7
8
9
    ff32 = np.finfo(np.float32)
    print(ff32.bits) # 32
10
    print(ff32.min) # -3.4028235e+38
11
12
     print(ff32.max) # 3.4028235e+38
     print(ff32.eps) # 1.1920929e-07
13
```

# 3 时间日期和时间增量

## 3.1 datetime64 基础

在 numpy 中,我们很方便的将字符串转换成时间日期类型 datetime64 (datetime 已被 python 包含的日期时间库所占用)。

datatime64 是带单位的日期时间类型,其单位如下:

ŧ	日期单位	代码含义	时间单位	代码含义
ŧ	Y	年	h	小时
ŧ	M	月	m	分钟
ŧ	W	周	S	秒
ŧ	D	天	ms	毫秒
ŧ	-	-	us	微秒
ŧ	-	-	ns	纳秒
ŧ	-	-	ps	皮秒
ŧ	-	-	fs	飞秒
ŧ	-	-	as	阿托秒

## 注意:

- 1. 1秒 = 1000 毫秒 (milliseconds)
- 2. 1毫秒 = 1000 微秒 (microseconds)

【例】从字符串创建 datetime64 类型时,默认情况下,numpy 会根据字符串自动选择对应的单位。

```
1
     import numpy as np
3
    a = np.datetime64('2020-03-01')
     print(a, a.dtype) # 2020-03-01 datetime64[D]
4
    a = np.datetime64('2020-03')
6
7
     print(a, a.dtype) # 2020-03 datetime64[M]
8
9
     a = np.datetime64('2020-03-08 20:00:05')
     print(a, a.dtype) # 2020-03-08T20:00:05 datetime64[s]
10
11
12
     a = np.datetime64('2020-03-08 20:00')
13
     print(a, a.dtype) # 2020-03-08T20:00 datetime64[m]
14
15
     a = np.datetime64('2020-03-08 20')
16
     print(a, a.dtype) # 2020-03-08T20 datetime64[h]
```

【例】从字符串创建 datetime64 类型时,可以强制指定使用的单位。

```
import numpy as np

a = np.datetime64('2020-03', 'D')

print(a, a.dtype) # 2020-03-01 datetime64[D]

a = np.datetime64('2020-03', 'Y')

print(a, a.dtype) # 2020 datetime64[Y]

print(np.datetime64('2020-03') == np.datetime64('2020-03-01')) # True

print(np.datetime64('2020-03') == np.datetime64('2020-03-02')) #False
```

由上例可以看出,2019-03 和2019-03-01 所表示的其实是同一个时间。

事实上,如果两个 datetime64 对象具有不同的单位,它们可能仍然代表相同的时刻。并且从较大的单位(如月份)转换为较小的单位(如天数)是安全的。

【例】从字符串创建 datetime64 数组时,如果单位不统一,则一律转化成其中最小的单位。

```
import numpy as np

a = np.array(['2020-03', '2020-03-08', '2020-03-08 20:00'], dtype='datetime64')

print(a, a.dtype)

# ['2020-03-01T00:00' '2020-03-08T00:00' '2020-03-08T20:00'] datetime64[m]
```

【例】使用 arange() 创建 datetime64 数组,用于生成日期范围。

```
1
     import numpy as np
2
     a = np.arange('2020-08-01', '2020-08-10', dtype=np.datetime64)
3
4
     print(a)
     # ['2020-08-01' '2020-08-02' '2020-08-03' '2020-08-04' '2020-08-05'
     # '2020-08-06' '2020-08-07' '2020-08-08' '2020-08-09']
6
7
     print(a.dtype) # datetime64[D]
8
9
     a = np.arange('2020-08-01 20:00', '2020-08-10', dtype=np.datetime64)
10
     print(a)
11
     # ['2020-08-01T20:00' '2020-08-01T20:01' '2020-08-01T20:02' ...
     # '2020-08-09T23:57' '2020-08-09T23:58' '2020-08-09T23:59']
12
13
     print(a.dtype) # datetime64[m]
14
     a = np.arange('2020-05', '2020-12', dtype=np.datetime64)
15
16
     print(a)
     # ['2020-05' '2020-06' '2020-07' '2020-08' '2020-09' '2020-10' '2020-11']
17
     print(a.dtype) # datetime64[M]
```

# 3.2 datetime64 和 timedelta64 运算

【例】timedelta64表示两个 datetime64 之间的差。timedelta64 也是带单位的,并且和相减运算中的两个 datetime64 中的较小的单位保持一致。

```
1
     import numpy as np
 2
3
     a = np.datetime64('2020-03-08') - np.datetime64('2020-03-07')
4
     b = np.datetime64('2020-03-08') - np.datetime64('202-03-07 08:00')
     c = np.datetime64('2020-03-08') - np.datetime64('2020-03-07 23:00', 'D')
5
6
     print(a, a.dtype) # 1 days timedelta64[D]
7
     print(b, b.dtype) # 956178240 minutes timedelta64[m]
8
9
     print(c, c.dtype) # 1 days timedelta64[D]
10
11
     a = np.datetime64('2020-03') + np.timedelta64(20, 'D')
12
     b = np.datetime64('2020-06-15 00:00') + np.timedelta64(12, 'h')
     print(a, a.dtype) # 2020-03-21 datetime64[D]
13
14
     print(b, b.dtype) # 2020-06-15T12:00 datetime64[m]
```

【例】生成 timedelta64时,要注意年('Y')和月('M')这两个单位无法和其它单位进行运算(一年有几天?一个月有几个小时?这些都是不确定的)。

```
1
     import numpy as np
 2
3
     a = np.timedelta64(1, 'Y')
4
     b = np.timedelta64(a, 'M')
5
     print(a) # 1 years
     print(b) # 12 months
6
7
8
     c = np.timedelta64(1, 'h')
9
     d = np.timedelta64(c, 'm')
     print(c) # 1 hours
10
11
     print(d) # 60 minutes
12
13
     print(np.timedelta64(a, 'D'))
14
     # TypeError: Cannot cast NumPy timedelta64 scalar from metadata [Y] to [D] according to the rule 'same_kind'
15
16
     print(np.timedelta64(b, 'D'))
17
     # TypeError: Cannot cast NumPy timedelta64 scalar from metadata [M] to [D] according to the rule 'same_kind'
```

【例】timedelta64的运算。

```
1
     import numpy as np
2
     a = np.timedelta64(1, 'Y')
3
4
     b = np.timedelta64(6, 'M')
5
     c = np.timedelta64(1, 'W')
     d = np.timedelta64(1, 'D')
6
7
     e = np.timedelta64(10, 'D')
8
9
     print(a) # 1 years
    print(b) # 6 months
10
11
     print(a + b) # 18 months
12
     print(a - b) # 6 months
     print(2 * a) # 2 years
13
14
     print(a / b) # 2.0
15
     print(c / d) # 7.0
16
     print(c % e) # 7 days
```

## 【例】numpy.datetime64 与 datetime.datetime 相互转换

```
1
     import numpy as np
 2
     import datetime
 3
4
     dt = datetime.datetime(year=2020, month=6, day=1, hour=20, minute=5, second=30)
5
     dt64 = np.datetime64(dt, 's')
     print(dt64, dt64.dtype)
6
     # 2020-06-01T20:05:30 datetime64[s]
7
8
9
     dt2 = dt64.astype(datetime.datetime)
10
     print(dt2, type(dt2))
     # 2020-06-01 20:05:30 <class 'datetime.datetime'>
11
```

# 3.3 datetime64 的应用

为了允许在只有一周中某些日子有效的上下文中使用日期时间,NumPy包含一组"busday"(工作日)功能。

1. numpy.busday\_offset(dates, offsets, roll='raise', weekmask='1111100', holidays=None, busdaycal=None, out=None)

First adjusts the date to fall on a valid day according to the roll rule, then applies offsets to the given dates counted in valid days.

参数 roll: {'raise', 'nat', 'forward', 'following', 'backward', 'preceding', 'modifiedfollowing', 'modifiedpreceding'}

- 1. 'raise' means to raise an exception for an invalid day.
- 2. 'nat' means to return a NaT (not-a-time) for an invalid day.
- 3. 'forward' and 'following' mean to take the first valid day later in time.
- 4. 'backward' and 'preceding' mean to take the first valid day earlier in time.

【例】将指定的偏移量应用于工作日,单位天('D')。计算下一个工作日,如果当前日期为非工作日,默认报错。可以指定 forward 或backward 规则来避免报错。(一个是向前取第一个有效的工作日,一个是向后取第一个有效的工作日)

```
1
     import numpy as np
 2
3
     # 2020-07-10 星期五
4
     a = np.busday_offset('2020-07-10', offsets=1)
5
     print(a) # 2020-07-13
6
7
     a = np.busday_offset('2020-07-11', offsets=1)
8
9
     # ValueError: Non-business day date in busday_offset
10
11
     a = np.busday_offset('2020-07-11', offsets=0, roll='forward')
     b = np.busday_offset('2020-07-11', offsets=0, roll='backward')
12
13
     print(a) # 2020-07-13
14
     print(b) # 2020-07-10
15
     a = np.busday_offset('2020-07-11', offsets=1, roll='forward')
16
     b = np.busday_offset('2020-07-11', offsets=1, roll='backward')
17
     print(a) # 2020-07-14
18
```

```
19 print(b) # 2020-07-13
```

可以指定偏移量为0来获取当前日期向前或向后最近的工作日,当然,如果当前日期本身就是工作日,则直接返回当前日期。

1. numpy.is\_busday(dates, weekmask='1111100', holidays=None, busdaycal=None, out=None) Calculates which of the given dates are valid days, and which are not.

【例】返回指定日期是否是工作日。

```
1 import numpy as np
2
3 # 2020-07-10 星期五
4 a = np.is_busday('2020-07-10')
5 b = np.is_busday('2020-07-11')
6 print(a) # True
7 print(b) # False
```

【例】统计一个 datetime64[D] 数组中的工作日天数。

```
1
     import numpy as np
2
 3
     # 2020-07-10 星期五
4
     begindates = np.datetime64('2020-07-10')
     enddates = np.datetime64('2020-07-20')
5
6
     a = np.arange(begindates, enddates, dtype='datetime64')
7
     b = np.count_nonzero(np.is_busday(a))
8
     print(a)
9
     # ['2020-07-10' '2020-07-11' '2020-07-12' '2020-07-13' '2020-07-14'
     # '2020-07-15' '2020-07-16' '2020-07-17' '2020-07-18' '2020-07-19']
10
11
     print(b) # 6
```

【例】自定义周掩码值,即指定一周中哪些星期是工作日。

```
1 import numpy as np
2
3 # 2020-07-10 星期五
4 a = np.is_busday('2020-07-10', weekmask=[1, 1, 1, 1, 1, 0, 0])
5 b = np.is_busday('2020-07-10', weekmask=[1, 1, 1, 1, 0, 0, 1])
6 print(a) # True
7 print(b) # False
```

1. numpy.busday\_count(begindates, enddates, weekmask='1111100', holidays=[], busdaycal=None, out=None) Counts the number of valid days between begindates and enddates, not including the day of enddates.

【例】返回两个日期之间的工作日数量。

```
import numpy as np

# 2020-07-10 星期五

begindates = np.datetime64('2020-07-10')

enddates = np.datetime64('2020-07-20')

a = np.busday_count(begindates, enddates)

b = np.busday_count(enddates, begindates)

print(a) # 6

print(b) # -6
```

## 参考图文

- 1. https://www.jianshu.com/p/336cd77d9914
- 2. https://www.cnblogs.com/gl1573/p/10549547.html#h2datetime64
- 3. <a href="https://www.numpy.org.cn/reference/arrays/datetime.html#%E6%97%A5%E6%9C%9F%E6%97%B6%E9%97%B4%E5%8D%95%E4%BD%8D">https://www.numpy.org.cn/reference/arrays/datetime.html#%E6%97%A5%E6%9C%9F%E6%97%B6%E9%97%B4%E5%8D%95%E4%BD%8D</a>

# 4 数组的创建

导入 numpy。

```
1 import numpy as np
```

numpy 提供的最重要的数据结构是 ndarray, 它是 python 中 list 的扩展。

# 4.1 1. 依据现有数据来创建 ndarray

# 4.1.1 (a) 通过array()函数进行创建。

```
def array(p_object, dtype=None, copy=True, order='K', subok=False, ndmin=0):
```

```
1
     import numpy as np
     # 创建一维数组
3
    a = np.array([0, 1, 2, 3, 4])
     b = np.array((0, 1, 2, 3, 4))
5
     print(a, type(a))
     # [0 1 2 3 4] <class 'numpy.ndarray'>
     print(b, type(b))
8
     # [0 1 2 3 4] <class 'numpy.ndarray'>
9
10
11
     # 创建二维数组
     c = np.array([[11, 12, 13, 14, 15],
12
                   [16, 17, 18, 19, 20],
13
14
                   [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30],
15
                   [31, 32, 33, 34, 35]])
16
17
     print(c, type(c))
     # [[11 12 13 14 15]
18
     # [16 17 18 19 20]
19
     # [21 22 23 24 25]
21
     # [26 27 28 29 30]
     # [31 32 33 34 35]] <class 'numpy.ndarray'>
22
23
24
     # 创建三维数组
25
     d = np.array([[(1.5, 2, 3), (4, 5, 6)],
                   [(3, 2, 1), (4, 5, 6)]])
26
27
     print(d, type(d))
```

```
28 # [[[1.5 2. 3.]
29 # [4. 5. 6.]]
30 #
31 # [[3. 2. 1.]
32 # [4. 5. 6.]]] <class 'numpy.ndarray'>
```

# 4.1.2 (b) 通过asarray()函数进行创建

array()和 asarray()都可以将结构数据转化为 ndarray,但是 array()和 asarray()主要区别就是当数据源是ndarray时, array()仍然会 copy 出一个副本,占用新的内存,但不改变 dtype 时 asarray()不会。

```
def asarray(a, dtype=None, order=None):
    return array(a, dtype, copy=False, order=order)
```

【例】array()和 asarray()都可以将结构数据转化为 ndarray

```
1
     import numpy as np
3
    x = [[1, 1, 1], [1, 1, 1], [1, 1, 1]]
4
    y = np.array(x)
    z = np.asarray(x)
6
    x[1][2] = 2
     print(x,type(x))
7
8
     # [[1, 1, 1], [1, 1, 2], [1, 1, 1]] <class 'list'>
9
10
     print(y,type(y))
11
     # [[1 1 1]
12
     # [1 1 1]
     # [1 1 1]] <class 'numpy.ndarray'>
13
14
15
    print(z,type(z))
    # [[1 1 1]
16
17
     # [1 1 1]
     # [1 1 1]] <class 'numpy.ndarray'>
```

【例】array() 和 asarray() 的区别。(array() 和 asarray() 主要区别就是当数据源是**ndarray** 时, array() 仍然会 copy 出一个副本,占用新的内存,但不改变 dtype 时 asarray() 不会。)

```
1
     import numpy as np
2
3
    x = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
4
    y = np.array(x)
5
     z = np.asarray(x)
     w = np.asarray(x, dtype=np.int)
7
     x[1][2] = 2
     print(x,type(x),x.dtype)
8
     # [[1 1 1]
9
10
     # [1 1 2]
11
     # [1 1 1]] <class 'numpy.ndarray'> int32
12
13
     print(y,type(y),y.dtype)
14
     # [[1 1 1]
```

```
15
    # [1 1 1]
16
     # [1 1 1]] <class 'numpy.ndarray'> int32
17
18
    print(z,type(z),z.dtype)
    # [[1 1 1]
19
    # [1 1 2]
20
     # [1 1 1]] <class 'numpy.ndarray'> int32
21
    print(w,type(w),w.dtype)
23
    # [[1 1 1]
24
    # [1 1 2]
26
    # [1 1 1]] <class 'numpy.ndarray'> int32
```

【例】更改为较大的dtype时,其大小必须是array的最后一个axis的总大小(以字节为单位)的除数

```
1
     import numpy as np
2
    x = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
3
4
    print(x, x.dtype)
    # [[1 1 1]
    # [1 1 1]
6
    # [1 1 1]] int32
7
    x.dtype = np.float
8
9
10
    # ValueError: When changing to a larger dtype, its size must be a divisor of the total size in bytes of the last
     axis of the array.
```

# 4.1.3 (c) 通过fromfunction()函数进行创建

给函数绘图的时候可能会用到 fromfunction(),该函数可从函数中创建数组。

```
1 def fromfunction(function, shape, **kwargs):
```

【例】通过在每个坐标上执行一个函数来构造数组。

```
1
     import numpy as np
2
3
     def f(x, y):
        return 10 * x + y
4
5
    x = np.fromfunction(f, (5, 4), dtype=int)
6
    print(x)
7
    # [[ 0 1 2 3]
8
    # [10 11 12 13]
    # [20 21 22 23]
10
    # [30 31 32 33]
11
    # [40 41 42 43]]
12
13
    x = np.fromfunction(lambda i, j: i == j, (3, 3), dtype=int)
14
15
    print(x)
16
    # [[ True False False]
     # [False True False]
17
```

```
# [False False True]]

20  x = np.fromfunction(lambda i, j: i + j, (3, 3), dtype=int)
21  print(x)
22  # [[0 1 2]
23  # [1 2 3]
24  # [2 3 4]]
```

# 4.2 2. 依据 ones 和 zeros 填充方式

在机器学习任务中经常做的一件事就是初始化参数,需要用常数值或者随机值来创建一个固定大小的矩阵。

## 4.2.1 (a) 零数组

- 1. zeros()函数:返回给定形状和类型的零数组。
- 2. zeros\_like()函数:返回与给定数组形状和类型相同的零数组。

```
def zeros(shape, dtype=None, order='C'):
def zeros_like(a, dtype=None, order='K', subok=True, shape=None):
```

#### 【例】

```
import numpy as np
1
2
    x = np.zeros(5)
3
    print(x) # [0. 0. 0. 0. 0.]
5
    x = np.zeros([2, 3])
6
    print(x)
    # [[0. 0. 0.]
    # [0. 0. 0.]]
8
9
10
    x = np.array([[1, 2, 3], [4, 5, 6]])
    y = np.zeros_like(x)
11
12
    print(y)
    # [[0 0 0]
13
    # [0 0 0]]
```

## 4.2.2 (b) 1数组

- 1. ones()函数:返回给定形状和类型的1数组。
- 2. ones\_like() 函数: 返回与给定数组形状和类型相同的1数组。

```
def ones(shape, dtype=None, order='C'):
def ones_like(a, dtype=None, order='K', subok=True, shape=None):
```

## 【例】

```
import numpy as np
2
3
    x = np.ones(5)
4
     print(x) # [1. 1. 1. 1.]
     x = np.ones([2, 3])
5
    print(x)
6
     # [[1. 1. 1.]
     # [1. 1. 1.]]
8
9
     x = np.array([[1, 2, 3], [4, 5, 6]])
10
11
    y = np.ones_like(x)
12
    print(y)
     # [[1 1 1]
13
14
     # [1 1 1]]
```

## 4.2.3 (c) 空数组

- 1. empty() 函数:返回一个空数组,数组元素为随机数。
- 2. empty\_like 函数: 返回与给定数组具有相同形状和类型的新数组。

```
def empty(shape, dtype=None, order='C'):
def empty_like(prototype, dtype=None, order='K', subok=True, shape=None):
```

```
1
     import numpy as np
 2
3
    x = np.empty(5)
4
     print(x)
5
     # [1.95821574e-306 1.60219035e-306 1.37961506e-306
     # 9.34609790e-307 1.24610383e-306]
6
7
8
    x = np.empty((3, 2))
9
     print(x)
10
     # [[1.60220393e-306 9.34587382e-307]
     # [8.45599367e-307 7.56598449e-307]
11
     # [1.33509389e-306 3.59412896e-317]]
12
13
14
     x = np.array([[1, 2, 3], [4, 5, 6]])
     y = np.empty_like(x)
15
     print(y)
17
     # [[ 7209029 6422625
                             6619244]
     # [
18
             100 707539280
                                  504]]
```

## 4.2.4 (d) 单位数组

- 1. eye()函数:返回一个对角线上为1,其它地方为零的单位数组。
- 2. identity()函数:返回一个方的单位数组。

```
def eye(N, M=None, k=0, dtype=float, order='C'):
def identity(n, dtype=None):
```

## 【例】

```
import numpy as np
 2
 3
    x = np.eye(4)
 4
    print(x)
 5
    # [[1. 0. 0. 0.]
    # [0. 1. 0. 0.]
    # [0. 0. 1. 0.]
 7
 8
     # [0. 0. 0. 1.]]
 9
10
     x = np.eye(2, 3)
11
     print(x)
12
     # [[1. 0. 0.]
13
     # [0. 1. 0.]]
14
15
    x = np.identity(4)
16
    print(x)
    # [[1. 0. 0. 0.]
    # [0. 1. 0. 0.]
18
    # [0. 0. 1. 0.]
19
    # [0. 0. 0. 1.]]
20
```

# 4.2.5 (e) 对角数组

1. diag()函数:提取对角线或构造对角数组。

```
1 def diag(v, k=0):
```

```
1
    import numpy as np
2
3
    x = np.arange(9).reshape((3, 3))
4
    print(x)
    # [[0 1 2]
    # [3 4 5]
6
7
    # [6 7 8]]
8
     print(np.diag(x)) # [0 4 8]
9
     print(np.diag(x, k=1)) # [1 5]
     print(np.diag(x, k=-1)) # [3 7]
10
11
12
    v = [1, 3, 5, 7]
13
     x = np.diag(v)
```

```
14 print(x)

15 # [[1 0 0 0]

16 # [0 3 0 0]

17 # [0 0 5 0]

18 # [0 0 0 7]]
```

## 4.2.6 (f) 常数数组

- 1. full()函数:返回一个常数数组。
- 2. full\_like() 函数: 返回与给定数组具有相同形状和类型的常数数组。

```
def full(shape, fill_value, dtype=None, order='C'):
def full_like(a, fill_value, dtype=None, order='K', subok=True, shape=None):
```

## 【例】

```
1
     import numpy as np
3
    x = np.full((2,), 7)
4
     print(x)
     # [7 7]
6
7
     x = np.full(2, 7)
8
     print(x)
9
     # [7 7]
10
11
     x = np.full((2, 7), 7)
12
     print(x)
13
     # [[7 7 7 7 7 7 7]
     # [7 7 7 7 7 7 7]]
14
15
16
     x = np.array([[1, 2, 3], [4, 5, 6]])
     y = np.full_like(x, 7)
17
18
     print(y)
     # [[7 7 7]
20
    # [7 7 7]]
```

# 4.3 3. 利用数值范围来创建ndarray

- 1. arange() 函数: 返回给定间隔内的均匀间隔的值。
- 2. linspace()函数:返回指定间隔内的等间隔数字。
- 3. logspace()函数:返回数以对数刻度均匀分布。
- 4. numpy.random.rand() 返回一个由[0,1)内的随机数组成的数组。

【例】

```
1
    import numpy as np
2
3
    x = np.arange(5)
4
    print(x) # [0 1 2 3 4]
5
6
    x = np.arange(3, 7, 2)
7
    print(x) # [3 5]
8
9
    x = np.linspace(start=0, stop=2, num=9)
10
    # [0. 0.25 0.5 0.75 1. 1.25 1.5 1.75 2. ]
11
12
13
    x = np.logspace(0, 1, 5)
14
    print(np.around(x, 2))
15
    # [ 1. 1.78 3.16 5.62 10. ]
16
                                      #np.around 返回四舍五入后的值, 可指定精度。
17
                                     # around(a, decimals=0, out=None)
18
                                     # a 输入数组
                                     # decimals 要舍入的小数位数。 默认值为0。 如果为负,整数将四舍五入到小数点左侧的位置
19
20
21
22
    x = np.linspace(start=0, stop=1, num=5)
     x = [10 ** i for i in x]
23
24
    print(np.around(x, 2))
    # [ 1. 1.78 3.16 5.62 10. ]
25
26
27
    x = np.random.random(5)
28
    print(x)
    # [0.41768753 0.16315577 0.80167915 0.99690199 0.11812291]
29
    x = np.random.random([2, 3])
31
32
    print(x)
33
    # [[0.41151858 0.93785153 0.57031309]
    # [0.13482333 0.20583516 0.45429181]]
```

# 4.4 4. 结构数组的创建

## 4.4.1 (a) 利用字典来定义结构

#### 【例】

```
1
     import numpy as np
2
3
     personType = np.dtype({
         'names': ['name', 'age', 'weight'],
4
         'formats': ['U30', 'i8', 'f8']})
5
6
7
     a = np.array([('Liming', 24, 63.9), ('Mike', 15, 67.), ('Jan', 34, 45.8)],
8
                  dtype=personType)
9
     print(a, type(a))
10
    # [('Liming', 24, 63.9) ('Mike', 15, 67.) ('Jan', 34, 45.8)]
     # <class 'numpy.ndarray'>
11
```

# 4.4.2 (b) 利用包含多个元组的列表来定义结构

```
1
    import numpy as np
2
    personType = np.dtype([('name', 'U30'), ('age', 'i8'), ('weight', 'f8')])
3
    a = np.array([('Liming', 24, 63.9), ('Mike', 15, 67.), ('Jan', 34, 45.8)],
4
5
                 dtype=personType)
6
    print(a, type(a))
7
    # [('Liming', 24, 63.9) ('Mike', 15, 67.) ('Jan', 34, 45.8)]
    # <class 'numpy.ndarray'>
8
9
10
    # 结构数组的取值方式和一般数组差不多,可以通过下标取得元素:
11
    print(a[0])
    # ('Liming', 24, 63.9)
12
13
14
    print(a[-2:])
    # [('Mike', 15, 67. ) ('Jan', 34, 45.8)]
15
16
    # 我们可以使用字段名作为下标获取对应的值
17
18
    print(a['name'])
    # ['Liming' 'Mike' 'Jan']
19
20
    print(a['age'])
    # [24 15 34]
21
    print(a['weight'])
22
    # [63.9 67. 45.8]
```

# 5数组的属性

在使用 numpy 时,你会想知道数组的某些信息。很幸运,在这个包里边包含了很多便捷的方法,可以给你想要的信息。

- 1. numpy.ndarray.ndim 用于返回数组的维数(轴的个数)也称为秩,一维数组的秩为1,二维数组的秩为2,以此类推。
- 2. numpy.ndarray.shape 表示数组的维度,返回一个元组,这个元组的长度就是维度的数目,即 ndim 属性(秩)。
- 3. numpy.ndarray.size 数组中所有元素的总量,相当于数组的 shape 中所有元素的乘积,例如矩阵的元素总量为行与列的乘积。
- 4. numpy.ndarray.dtype ndarray 对象的元素类型。
- 5. numpy.ndarray.itemsize 以字节的形式返回数组中每一个元素的大小。

```
class ndarray(object):

shape = property(lambda self: object(), lambda self, v: None, lambda self: None)

dtype = property(lambda self: object(), lambda self, v: None, lambda self: None)

size = property(lambda self: object(), lambda self, v: None, lambda self: None)

ndim = property(lambda self: object(), lambda self, v: None, lambda self: None)

itemsize = property(lambda self: object(), lambda self, v: None, lambda self: None)
```

#### 【例】

```
1
     import numpy as np
3
     a = np.array([1, 2, 3, 4, 5])
     print(a.shape) # (5,)
4
     print(a.dtype) # int32
     print(a.size) # 5
6
7
     print(a.ndim) # 1
8
     print(a.itemsize) # 4
9
10
     b = np.array([[1, 2, 3], [4, 5, 6.0]])
     print(b.shape) # (2, 3)
11
     print(b.dtype) # float64
12
     print(b.size) # 6
13
     print(b.ndim) # 2
14
     print(b.itemsize) # 8
```

在 ndarray 中所有元素必须是同一类型,否则会自动向下转换, int->float->str。

```
import numpy as np

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

print(a) # [1 2 3 4 5]

b = np.array([1, 2, 3, 4, '5'])

print(b) # ['1' '2' '3' '4' '5']

c = np.array([1, 2, 3, 4, 5.0])

print(c) # [1. 2. 3. 4. 5.]
```

# 6 副本与视图

在 Numpy 中,尤其是在做数组运算或数组操作时,返回结果不是数组的 副本 就是 视图。

在 Numpy 中, 所有赋值运算不会为数组和数组中的任何元素创建副本。

1. numpy.ndarray.copy() 函数创建一个副本。 对副本数据进行修改,不会影响到原始数据,它们物理内存不在同一位置。

#### 【例】

```
1
    import numpy as np
2
3
    x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
4
    y = x
5
    y[0] = -1
6
    print(x)
7
    #[-1 2 3 4 5 6 7 8]
8
    print(y)
    # [-1 2 3 4 5 6 7 8]
9
10
    x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
11
12
    y = x.copy()
    y[0] = -1
13
14
    print(x)
    # [1 2 3 4 5 6 7 8]
15
16
    print(y)
17
    #[-1 2 3 4 5 6 7 8]
```

【例】数组切片操作返回的对象只是原数组的视图。

```
1
     import numpy as np
2
3
    x = np.array([[11, 12, 13, 14, 15],
                  [16, 17, 18, 19, 20],
4
5
                   [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30],
6
7
                  [31, 32, 33, 34, 35]])
8
    y = x
    y[::2, :3:2] = -1
9
10
    print(x)
    # [[-1 12 -1 14 15]
11
    # [16 17 18 19 20]
12
    # [-1 22 -1 24 25]
13
14
    # [26 27 28 29 30]
    # [-1 32 -1 34 35]]
15
    print(y)
```

```
17 # [[-1 12 -1 14 15]
18
    # [16 17 18 19 20]
    # [-1 22 -1 24 25]
19
    # [26 27 28 29 30]
20
21
    # [-1 32 -1 34 35]]
22
23
    x = np.array([[11, 12, 13, 14, 15],
                  [16, 17, 18, 19, 20],
24
25
                  [21, 22, 23, 24, 25],
                  [26, 27, 28, 29, 30],
26
27
                  [31, 32, 33, 34, 35]])
28
    y = x.copy()
    y[::2, :3:2] = -1
29
30
    print(x)
31
    # [[11 12 13 14 15]
    # [16 17 18 19 20]
32
    # [21 22 23 24 25]
33
34
    # [26 27 28 29 30]
    # [31 32 33 34 35]]
35
    print(y)
36
37
    # [[-1 12 -1 14 15]
38
    # [16 17 18 19 20]
    # [-1 22 -1 24 25]
39
40
    # [26 27 28 29 30]
41 # [-1 32 -1 34 35]]
```

# 7索引与切片

数组索引机制指的是用方括号([])加序号的形式引用单个数组元素,它的用处很多,比如抽取元素,选取数组的几个元素,甚至为其赋一个新值。

# 7.1 整数索引

【例】要获取数组的单个元素,指定元素的索引即可。

```
import numpy as np
2
3
     x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
4
     print(x[2]) # 3
5
    x = np.array([[11, 12, 13, 14, 15],
6
7
                   [16, 17, 18, 19, 20],
8
                   [21, 22, 23, 24, 25],
9
                   [26, 27, 28, 29, 30],
                   [31, 32, 33, 34, 35]])
10
11
    print(x[2]) # [21 22 23 24 25]
12
     print(x[2][1]) # 22
13
     print(x[2, 1]) # 22
```

# 7.2 切片索引

切片操作是指抽取数组的一部分元素生成新数组。对 python **列表**进行切片操作得到的数组是原数组的**副本**,而对 **Numpy** 数据进行切片操作得到的数组则是指向相同缓冲区的**视图**。

如果想抽取(或查看)数组的一部分,必须使用切片语法,也就是,把几个用冒号( start:stop:step )隔开的数字置于方括号内。

为了更好地理解切片语法,还应该了解不明确指明起始和结束位置的情况。如省去第一个数字,numpy 会认为第一个数字是0;如省去第二个数字,numpy则会认为第二个数字是数组的最大索引值;如省去最后一个数字,它将会被理解为1,也就是抽取所有元素而不再考虑间隔。

#### 【例】对一维数组的切片

```
1
     import numpy as np
2
3
     x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
4
     print(x[0:2]) # [1 2]
     print(x[1:5:2]) # [2 4]
5
6
     print(x[2:]) # [3 4 5 6 7 8]
     print(x[:2]) # [1 2]
7
     print(x[-2:]) # [7 8]
8
9
     print(x[:-2]) # [1 2 3 4 5 6]
     print(x[:]) # [1 2 3 4 5 6 7 8]
10
     print(x[::-1]) # [8 7 6 5 4 3 2 1]
11
```

## 【例】对二维数组切片

```
import numpy as np
2
3
     x = np.array([[11, 12, 13, 14, 15],
                   [16, 17, 18, 19, 20],
4
5
                   [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30],
6
7
                   [31, 32, 33, 34, 35]])
8
     print(x[0:2])
9
     # [[11 12 13 14 15]
10
     # [16 17 18 19 20]]
11
12
     print(x[1:5:2])
13
     # [[16 17 18 19 20]
     # [26 27 28 29 30]]
14
15
     print(x[2:])
16
     # [[21 22 23 24 25]
17
     # [26 27 28 29 30]
18
     # [31 32 33 34 35]]
19
20
21
     print(x[:2])
22
     # [[11 12 13 14 15]
     # [16 17 18 19 20]]
23
24
25
     print(x[-2:])
     # [[26 27 28 29 30]
26
     # [31 32 33 34 35]]
27
28
29
     print(x[:-2])
     # [[11 12 13 14 15]
30
     # [16 17 18 19 20]
     # [21 22 23 24 25]]
32
33
34
     print(x[:])
    # [[11 12 13 14 15]
35
     # [16 17 18 19 20]
36
     # [21 22 23 24 25]
37
38
     # [26 27 28 29 30]
39
     # [31 32 33 34 35]]
```

```
40
41
     print(x[2, :]) # [21 22 23 24 25]
     print(x[:, 2]) # [13 18 23 28 33]
42
43
     print(x[0, 1:4]) # [12 13 14]
     print(x[1:4, 0]) # [16 21 26]
44
     print(x[1:3, 2:4])
45
46
     # [[18 19]
47
     # [23 24]]
48
49
     print(x[:, :])
     # [[11 12 13 14 15]
50
     # [16 17 18 19 20]
51
     # [21 22 23 24 25]
52
53
     # [26 27 28 29 30]
54
     # [31 32 33 34 35]]
55
56
     print(x[::2, ::2])
57
     # [[11 13 15]
     # [21 23 25]
58
     # [31 33 35]]
59
60
     print(x[::-1, :])
61
     # [[31 32 33 34 35]
62
     # [26 27 28 29 30]
63
     # [21 22 23 24 25]
64
     # [16 17 18 19 20]
65
     # [11 12 13 14 15]]
67
     print(x[:, ::-1])
68
69
     # [[15 14 13 12 11]
70
     # [20 19 18 17 16]
     # [25 24 23 22 21]
71
72
     # [30 29 28 27 26]
     # [35 34 33 32 31]]
```

通过对每个以逗号分隔的维度执行单独的切片,你可以对多维数组进行切片。因此,对于二维数组,我们的第一片定义了行的切片,第二片定义了列的切片。

```
1
     import numpy as np
2
3
     x = np.array([[11, 12, 13, 14, 15],
4
                   [16, 17, 18, 19, 20],
5
                   [21, 22, 23, 24, 25],
6
                   [26, 27, 28, 29, 30],
7
                   [31, 32, 33, 34, 35]])
8
     print(x)
     # [[11 12 13 14 15]
9
     # [16 17 18 19 20]
10
     # [21 22 23 24 25]
11
     # [26 27 28 29 30]
12
13
     # [31 32 33 34 35]]
14
```

```
15  x[0::2, 1::3] = 0

16  print(x)

17  # [[11  0  13  14  0]

18  # [16  17  18  19  20]

19  # [21  0  23  24  0]

20  # [26  27  28  29  30]

21  # [31  0  33  34  0]]
```

# 7.3 dots 索引

NumPy 允许使用 ... 表示足够多的冒号来构建完整的索引列表。

比如,如果 x 是5维数组:

```
1. x[1,2,...] 等于 x[1,2,:,:,:]
2. x[...,3] 等于 x[:,:,:,3]
3. x[4,...,5,:] 等于 x[4,:,:,5,:]
```

## 【例】

```
1
    import numpy as np
 2
 3
    x = np.random.randint(1, 100, [2, 2, 3])
    print(x)
 4
 5
    # [[[ 5 64 75]
    # [57 27 31]]
 6
 8
    # [[68 85 3]
    # [93 26 25]]]
9
10
11
    print(x[1, ...])
    # [[68 85 3]
12
    # [93 26 25]]
13
14
15
    print(x[..., 2])
    # [[75 31]
16
17
    # [ 3 25]]
```

# 7.4 整数数组索引

【例】方括号内传入多个索引值,可以同时选择多个元素。

```
1 import numpy as np
2
```

```
3
     x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
 4
     r = [0, 1, 2]
 5
     print(x[r])
 6
     # [1 2 3]
 8
     r = [0, 1, -1]
 9
     print(x[r])
10
     # [1 2 8]
11
     x = np.array([[11, 12, 13, 14, 15],
12
13
                   [16, 17, 18, 19, 20],
14
                   [21, 22, 23, 24, 25],
15
                   [26, 27, 28, 29, 30],
16
                   [31, 32, 33, 34, 35]])
17
     r = [0, 1, 2]
18
19
     print(x[r])
     # [[11 12 13 14 15]
20
     # [16 17 18 19 20]
21
     # [21 22 23 24 25]]
22
23
24
     r = [0, 1, -1]
     print(x[r])
25
26
     # [[11 12 13 14 15]
27
28
     # [16 17 18 19 20]
29
     # [31 32 33 34 35]]
30
31
     r = [0, 1, 2]
32
     c = [2, 3, 4]
     y = x[r, c]
33
34
     print(y)
35
     # [13 19 25]
```

```
1
     import numpy as np
 2
 3
     x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
 4
     r = np.array([[0, 1], [3, 4]])
 5
     print(x[r])
     # [[1 2]
 6
     # [4 5]]
 7
 8
 9
     x = np.array([[11, 12, 13, 14, 15],
                   [16, 17, 18, 19, 20],
10
11
                   [21, 22, 23, 24, 25],
12
                   [26, 27, 28, 29, 30],
13
                   [31, 32, 33, 34, 35]])
14
15
     r = np.array([[0, 1], [3, 4]])
16
     print(x[r])
     # [[[11 12 13 14 15]
17
```

```
18
       [16 17 18 19 20]]
19
    # [[26 27 28 29 30]
20
21
    # [31 32 33 34 35]]]
22
    # 获取了 5X5 数组中的四个角的元素。
23
    # 行索引是 [0,0] 和 [4,4], 而列索引是 [0,4] 和 [0,4]。
24
    r = np.array([[0, 0], [4, 4]])
25
    c = np.array([[0, 4], [0, 4]])
26
    y = x[r, c]
27
    print(y)
28
29
    # [[11 15]
    # [31 35]]
30
```

【例】可以借助切片:与整数数组组合。

```
1
     import numpy as np
2
3
     x = np.array([[11, 12, 13, 14, 15],
                   [16, 17, 18, 19, 20],
4
5
                   [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30],
6
                   [31, 32, 33, 34, 35]])
7
8
9
     y = x[0:3, [1, 2, 2]]
10
     print(y)
11
     # [[12 13 13]
     # [17 18 18]
12
13
     # [22 23 23]]
```

1. numpy. take(a, indices, axis=None, out=None, mode='raise') Take elements from an array along an axis.

```
import numpy as np
1
2
3
     x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
4
     r = [0, 1, 2]
     print(np.take(x, r))
     # [1 2 3]
6
7
     r = [0, 1, -1]
8
9
     print(np.take(x, r))
     # [1 2 8]
10
11
12
     x = np.array([[11, 12, 13, 14, 15],
                   [16, 17, 18, 19, 20],
13
                    [21, 22, 23, 24, 25],
14
                   [26, 27, 28, 29, 30],
15
                   [31, 32, 33, 34, 35]])
16
17
18
     r = [0, 1, 2]
19
     print(np.take(x, r, axis=0))
     # [[11 12 13 14 15]
20
```

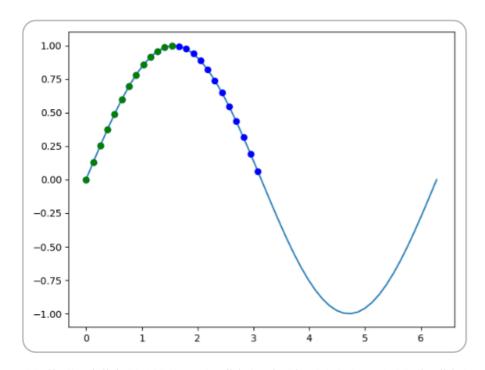
```
# [16 17 18 19 20]
21
     # [21 22 23 24 25]]
22
23
24
     r = [0, 1, -1]
25
     print(np.take(x, r, axis=0))
     # [[11 12 13 14 15]
26
     # [16 17 18 19 20]
27
28
     # [31 32 33 34 35]]
29
     r = [0, 1, 2]
30
31
     c = [2, 3, 4]
32
     y = np.take(x, [r, c])
     print(y)
33
34
     # [[11 12 13]
35
     # [13 14 15]]
```

# 7.5 布尔索引

我们可以通过一个布尔数组来索引目标数组。

```
1
    import numpy as np
2
3
    x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
4
    y = x > 5
5
    print(y)
    # [False False False False True True]
7
    print(x[x > 5])
    # [6 7 8]
8
9
    x = np.array([np.nan, 1, 2, np.nan, 3, 4, 5])
10
11
    y = np.logical_not(np.isnan(x))
12
    print(x[y])
13
    # [1. 2. 3. 4. 5.]
14
15
    x = np.array([[11, 12, 13, 14, 15],
                  [16, 17, 18, 19, 20],
16
17
                  [21, 22, 23, 24, 25],
                  [26, 27, 28, 29, 30],
18
                  [31, 32, 33, 34, 35]])
19
    y = x > 25
20
21
    print(y)
    # [[False False False False]
22
    # [False False False False]
23
    # [False False False False]
24
    # [ True True True True]
25
    # [ True True True True]]
26
27
    print(x[x > 25])
```

```
1
   import numpy as np
2
   import matplotlib.pyplot as plt
3
4
5
   x = np.linspace(0, 2 * np.pi, 50)
   y = np.sin(x)
6
7
   print(len(x)) # 50
8
   plt.plot(x, y)
9
10
   mask = y >= 0
   print(len(x[mask])) # 25
11
12
   print(mask)
13
   14
    15
    True False False False False False False False False False False
16
17
    False False
18
    False False]
    111
19
20
   plt.plot(x[mask], y[mask], 'bo')
21
22
   mask = np.logical_and(y >= 0, x <= np.pi / 2)
23
   print(mask)
24
   25
    True False False False False False False False False False False
26
27
    False False
    False False False False False False False False False False False
28
29
    False False]
30
31
   plt.plot(x[mask], y[mask], 'go')
32
33
   plt.show()
```



我们利用这些条件来选择图上的不同点。蓝色点(在图中还包括绿点,但绿点掩盖了蓝色点),显示值 大于0 的所有点。绿色点表示值 大于0 且 小于 $0.5\pi$  的所有点。

# 8数组迭代

除了for循环, Numpy 还提供另外一种更为优雅的遍历方法。

1. apply\_along\_axis(func1d, axis, arr) Apply a function to 1-D slices along the given axis.

```
1
     import numpy as np
 2
 3
     x = np.array([[11, 12, 13, 14, 15],
 4
                   [16, 17, 18, 19, 20],
 5
                   [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30],
 6
 7
                   [31, 32, 33, 34, 35]])
 8
 9
     y = np.apply_along_axis(np.sum, 0, x)
10
     print(y) # [105 110 115 120 125]
11
     y = np.apply_along_axis(np.sum, 1, x)
     print(y) # [ 65 90 115 140 165]
12
13
14
     y = np.apply_along_axis(np.mean, 0, x)
15
     print(y) # [21. 22. 23. 24. 25.]
16
     y = np.apply_along_axis(np.mean, 1, x)
     print(y) # [13. 18. 23. 28. 33.]
17
18
19
     def my_func(x):
20
21
         return (x[0] + x[-1]) * 0.5
22
23
24
     y = np.apply_along_axis(my_func, 0, x)
     print(y) # [21. 22. 23. 24. 25.]
25
     y = np.apply_along_axis(my_func, 1, x)
26
     print(y) # [13. 18. 23. 28. 33.]
27
```

# 9数组操作

## 9.1 更改形状

在对数组进行操作时,为了满足格式和计算的要求通常会改变其形状。

1. numpy.ndarray.shape 表示数组的维度,返回一个元组,这个元组的长度就是维度的数目,即 ndim 属性(秩)。

【例】通过修改 shap 属性来改变数组的形状。

```
import numpy as np

x = np.array([1, 2, 9, 4, 5, 6, 7, 8])
print(x.shape) # (8,)

x.shape = [2, 4]
print(x)
# [[1 2 9 4]
# [5 6 7 8]]
```

1. numpy.ndarray.flat 将数组转换为一维的迭代器,可以用for访问数组每一个元素。

### 【例】

```
1
     import numpy as np
2
3
     x = np.array([[11, 12, 13, 14, 15],
4
                   [16, 17, 18, 19, 20],
5
                   [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30],
6
                   [31, 32, 33, 34, 35]])
8
     y = x.flat
     print(y)
9
10
     # <numpy.flatiter object at 0x0000020F9BA10C60>
     for i in y:
11
         print(i, end=' ')
12
     # 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
13
14
     y[3] = 0
15
     print(end='\n')
16
17
     print(x)
    # [[11 12 13 0 15]
18
    # [16 17 18 19 20]
19
    # [21 22 23 24 25]
20
     # [26 27 28 29 30]
21
22
     # [31 32 33 34 35]]
```

1. numpy.ndarray.flatten([order='C']) 将数组的副本转换为一维数组,并返回。

a. order: 'C' -- 按行, 'F' -- 按列, 'A' -- 原顺序, 'k' -- 元素在内存中的出现顺序。(简记)

b. order: {'C/F, 'A, K}, 可选使用此索引顺序读取a的元素。'C'意味着以行大的C风格顺序对元素进行索引,最后一个轴索引会更改F表示以列大的Fortran样式顺序索引元素,其中第一个索引变化最快,最后一个索引变化最快。请注意,'C'和'F'选项不考虑基础数组的内存布局,仅引用轴索引的顺序.A'表示如果a为Fortran,则以类似Fortran的索引顺序读取元素在内存中连续,否则类似C的顺序。"K"表示按照步序在内存中的顺序读取元素,但步幅为负时反转数据除外。默认情况下,使用Cindex顺序。

### 【例】flatten()函数返回的是拷贝。

```
1
     import numpy as np
2
3
     x = np.array([[11, 12, 13, 14, 15],
4
                   [16, 17, 18, 19, 20],
                   [21, 22, 23, 24, 25],
5
                   [26, 27, 28, 29, 30],
6
7
                   [31, 32, 33, 34, 35]])
     y = x.flatten()
8
9
     print(y)
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
11
12
13
    y[3] = 0
     print(x)
14
     # [[11 12 13 14 15]
15
     # [16 17 18 19 20]
16
     # [21 22 23 24 25]
17
     # [26 27 28 29 30]
18
19
     # [31 32 33 34 35]]
20
21
     x = np.array([[11, 12, 13, 14, 15],
22
                   [16, 17, 18, 19, 20],
23
                   [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30],
24
25
                   [31, 32, 33, 34, 35]])
26
     y = x.flatten(order='F')
27
28
29
     # [11 16 21 26 31 12 17 22 27 32 13 18 23 28 33 14 19 24 29 34 15 20 25 30
30
     # 351
31
32
     y[3] = 0
33
     print(x)
    # [[11 12 13 14 15]
34
     # [16 17 18 19 20]
35
    # [21 22 23 24 25]
36
    # [26 27 28 29 30]
37
    # [31 32 33 34 35]]
38
```

1. numpy.ravel(a, order='C') Return a contiguous flattened array.

#### 【例】ravel()返回的是视图。

```
2
3
     x = np.array([[11, 12, 13, 14, 15],
                   [16, 17, 18, 19, 20],
4
5
                   [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30],
6
7
                   [31, 32, 33, 34, 35]])
8
     y = np.ravel(x)
9
     print(y)
     # [11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
10
11
12
13
     y[3] = 0
     print(x)
14
15
     # [[11 12 13 0 15]
16
     # [16 17 18 19 20]
     # [21 22 23 24 25]
17
     # [26 27 28 29 30]
18
     # [31 32 33 34 35]]
19
20
     【例】order=F 就是拷贝
21
22
23
     x = np.array([[11, 12, 13, 14, 15],
                   [16, 17, 18, 19, 20],
24
25
                   [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30],
26
27
                   [31, 32, 33, 34, 35]])
28
     y = np.ravel(x, order='F')
29
30
     print(y)
31
     # [11 16 21 26 31 12 17 22 27 32 13 18 23 28 33 14 19 24 29 34 15 20 25 30
32
33
34
     y[3] = 0
35
     print(x)
     # [[11 12 13 14 15]
36
37
     # [16 17 18 19 20]
38
     # [21 22 23 24 25]
     # [26 27 28 29 30]
39
40
     # [31 32 33 34 35]]
```

1. numpy.reshape(a, newshape[, order='C']) 在不更改数据的情况下为数组赋予新的形状。

【例】 reshape() 函数当参数 newshape = [rows,-1] 时,将根据行数自动确定列数。

```
1  import numpy as np
2
3  x = np.arange(12)
4  y = np.reshape(x, [3, 4])
5  print(y.dtype) # int32
6  print(y)
7  # [[ 0  1  2  3]
8  # [ 4  5  6  7]
9  # [ 8  9  10  11]]
```

```
10
11
    y = np.reshape(x, [3, -1])
12
    print(y)
13
    # [[ 0 1 2 3]
    # [4567]
14
    # [ 8 9 10 11]]
15
16
17
    y = np.reshape(x,[-1,3])
18
    print(y)
    # [[ 0 1 2]
19
    # [ 3 4 5]
    # [678]
21
    # [ 9 10 11]]
22
23
24
    y[0, 1] = 10
25
    print(x)
    # [ 0 10 2 3 4 5 6 7 8 9 10 11] (改变x去reshape后y中的值, x对应元素也改变)
```

【例】 reshape() 函数当参数 newshape = -1 时,表示将数组降为一维。

```
1
    import numpy as np
3
    x = np.random.randint(12, size=[2, 2, 3])
4
    print(x)
    # [[[11 9 1]
5
    # [ 1 10 3]]
6
7
    # [[ 0 6 1]
8
    # [ 4 11 3]]]
9
10
    y = np.reshape(x, -1)
11
    print(y)
12
    # [11 9 1 1 10 3 0 6 1 4 11 3]
```

# 9.2 数组转置

```
1. numpy.transpose(a, axes=None) Permute the dimensions of an array.
```

2. numpy.ndarray.T Same as self.transpose(), except that self is returned if self.ndim < 2.

```
1  import numpy as np
2
3  x = np.random.rand(5, 5) * 10
4  x = np.around(x, 2)
5  print(x)
6  # [[6.74 8.46 6.74 5.45 1.25]
7  # [3.54 3.49 8.62 1.94 9.92]
8  # [5.03 7.22 1.6 8.7 0.43]
9  # [7.5 7.31 5.69 9.67 7.65]
```

```
10
    # [1.8 9.52 2.78 5.87 4.14]]
    y = x.T
11
12
    print(y)
13
    # [[6.74 3.54 5.03 7.5 1.8 ]
    # [8.46 3.49 7.22 7.31 9.52]
14
    # [6.74 8.62 1.6 5.69 2.78]
15
    # [5.45 1.94 8.7 9.67 5.87]
16
    # [1.25 9.92 0.43 7.65 4.14]]
17
    y = np.transpose(x)
18
    print(y)
19
    # [[6.74 3.54 5.03 7.5 1.8]
21
    # [8.46 3.49 7.22 7.31 9.52]
    # [6.74 8.62 1.6 5.69 2.78]
22
23
    # [5.45 1.94 8.7 9.67 5.87]
24
   # [1.25 9.92 0.43 7.65 4.14]]
```

# 9.3 更改维度

当创建一个数组之后,还可以给它增加一个维度,这在矩阵计算中经常会用到。

1. numpy.newaxis = None None 的别名,对索引数组很有用。

【例】很多工具包在进行计算时都会先判断输入数据的维度是否满足要求,如果输入数据达不到指定的维度时,可以使用 newaxis 参数来增加一个维度。

```
1
     import numpy as np
2
    x = np.array([1, 2, 9, 4, 5, 6, 7, 8])
4
    print(x.shape) # (8,)
5
     print(x) # [1 2 9 4 5 6 7 8]
7
     y = x[np.newaxis, :]
     print(y.shape) # (1, 8)
8
     print(y) # [[1 2 9 4 5 6 7 8]]
9
10
     y = x[:, np.newaxis]
11
    print(y.shape) # (8, 1)
12
13
    print(y)
14
    # [[1]
    # [2]
15
    # [9]
17
    # [4]
    # [5]
18
19
     # [6]
20
    # [7]
21
     # [8]]
```

- 1. numpy.squeeze(a, axis=None) 从数组的形状中删除单维度条目,即把shape中为1的维度去掉。
  - a. a 表示输入的数组;

b. axis 用于指定需要删除的维度,但是指定的维度必须为单维度,否则将会报错;

在机器学习和深度学习中,通常算法的结果是可以表示向量的数组(即包含两对或以上的方括号形式[[]]),如果直接利用这个数组进行画图可能显示界面为空(见后面的示例)。我们可以利用 squeeze() 函数将表示向量的数组转换为秩为1的数组,这样利用 matplotlib 库函数画图时,就可以正常的显示结果了。

#### 【例】

```
import numpy as np

x = np.arange(10)
print(x.shape) # (10,)

x = x[np.newaxis, :]
print(x.shape) # (1, 10)

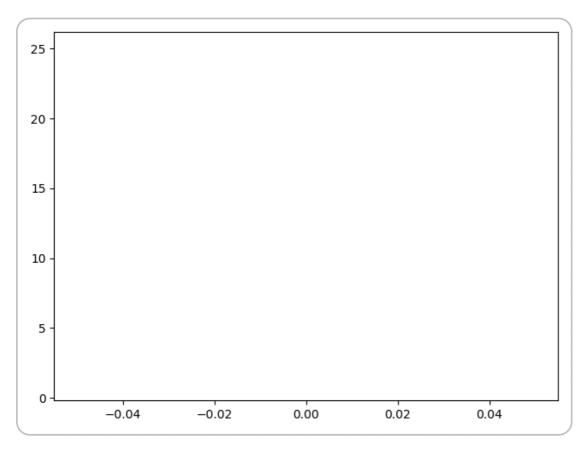
y = np.squeeze(x)
print(y.shape) # (10,)
```

### 【例】

```
1
     import numpy as np
3
    x = np.array([[[0], [1], [2]]])
    print(x.shape) # (1, 3, 1)
4
     print(x)
6
    # [[[0]
     # [1]
7
8
     # [2]]]
9
10
     y = np.squeeze(x)
11
     print(y.shape) # (3,)
     print(y) # [0 1 2]
12
13
14
     y = np.squeeze(x, axis=0)
     print(y.shape) # (3, 1)
15
     print(y)
16
17
     # [[0]
18
     # [1]
19
     # [2]]
20
21
     y = np.squeeze(x, axis=2)
22
     print(y.shape) # (1, 3)
23
     print(y) # [[0 1 2]]
24
25
     y = np.squeeze(x, axis=1)
26
     # ValueError: cannot select an axis to squeeze out which has size not equal to one
```

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([[1, 4, 9, 16, 25]])
print(x.shape) # (1, 5)
plt.plot(x)
plt.show()
```

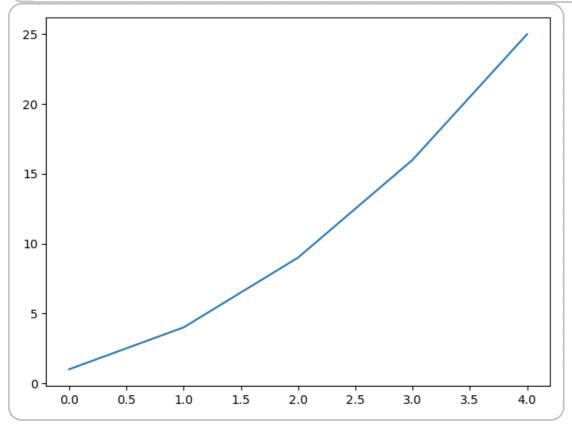


```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([[1, 4, 9, 16, 25]])

x = np.squeeze(x)
print(x.shape) # (5, )

plt.plot(x)
plt.show()
```



## 9.4 数组组合

如果要将两份数据组合到一起,就需要拼接操作。

1. numpy.concatenate((a1, a2, ...), axis=0, out=None) Join a sequence of arrays along an existing axis.

【例】连接沿现有轴的数组序列(原来x,y都是一维的,拼接后的结果也是一维的)。

```
import numpy as np
3
    x = np.array([1, 2, 3])
4
    y = np.array([7, 8, 9])
    z = np.concatenate([x, y])
6
    print(z)
7
    # [1 2 3 7 8 9]
8
9
    z = np.concatenate([x, y], axis=0)
10
     print(z)
11
    # [1 2 3 7 8 9]
```

【例】原来x,y都是二维的,拼接后的结果也是二维的。

```
import numpy as np
1
2
3
    x = np.array([1, 2, 3]).reshape(1, 3)
4
    y = np.array([7, 8, 9]).reshape(1, 3)
5
    z = np.concatenate([x, y])
6
    print(z)
    #[[1 2 3]
    # [ 7 8 9]]
8
9
    z = np.concatenate([x, y], axis=0)
10
    print(z)
    #[[1 2 3]
11
    # [ 7 8 9]]
12
    z = np.concatenate([x, y], axis=1)
13
14
    print(z)
   # [[ 1 2 3 7 8 9]]
15
```

【例】x,y在原来的维度上进行拼接。

```
import numpy as np
2
3
    x = np.array([[1, 2, 3], [4, 5, 6]])
    y = np.array([[7, 8, 9], [10, 11, 12]])
4
5
    z = np.concatenate([x, y])
    print(z)
6
7
    # [[ 1 2 3]
8
    # [ 4 5 6]
9
    # [7 8 9]
    # [10 11 12]]
10
    z = np.concatenate([x, y], axis=0)
11
12
    print(z)
    # [[ 1 2 3]
13
    # [ 4 5 6]
14
```

```
15 # [ 7 8 9]

16 # [10 11 12]]

17 z = np.concatenate([x, y], axis=1)

18 print(z)

19 # [[ 1 2 3 7 8 9]

20 # [ 4 5 6 10 11 12]]
```

1. numpy.stack(arrays, axis=0, out=None) Join a sequence of arrays along a new axis.

【例】沿着新的轴加入一系列数组(stack为增加维度的拼接)。

```
1
     import numpy as np
2
3
    x = np.array([1, 2, 3])
     y = np.array([7, 8, 9])
4
    z = np.stack([x, y])
    print(z.shape) # (2, 3)
6
7
     print(z)
8
     # [[1 2 3]
9
     # [7 8 9]]
10
     z = np.stack([x, y], axis=1)
11
     print(z.shape) # (3, 2)
12
13
     print(z)
     # [[1 7]
14
15
     # [28]
    # [3 9]]
16
```

```
1
     import numpy as np
2
3
    x = np.array([1, 2, 3]).reshape(1, 3)
    y = np.array([7, 8, 9]).reshape(1, 3)
4
5
    z = np.stack([x, y])
6
     print(z.shape) # (2, 1, 3)
7
     print(z)
8
     # [[[1 2 3]]
9
     # [[7 8 9]]]
10
11
12
     z = np.stack([x, y], axis=1)
     print(z.shape) # (1, 2, 3)
13
14
     print(z)
     # [[[1 2 3]
15
     # [7 8 9]]]
16
17
     z = np.stack([x, y], axis=2)
18
19
     print(z.shape) # (1, 3, 2)
20
     print(z)
     # [[[1 7]
21
22
     # [2 8]
23
    # [3 9]]]
```

```
1
    import numpy as np
2
3
    x = np.array([[1, 2, 3], [4, 5, 6]])
4
    y = np.array([[7, 8, 9], [10, 11, 12]])
    z = np.stack([x, y])
5
6
    print(z.shape) # (2, 2, 3)
7
    print(z)
    # [[[ 1 2 3]
8
9
       [ 4 5 6]]
10
11
    # [[ 7 8 9]
12
     # [10 11 12]]]
13
    z = np.stack([x, y], axis=1)
14
    print(z.shape) # (2, 2, 3)
15
    print(z)
16
17
    # [[[ 1 2 3]
     # [7 8 9]]
18
19
    # [[ 4 5 6]
20
    # [10 11 12]]]
21
22
23
    z = np.stack([x, y], axis=2)
    print(z.shape) # (2, 3, 2)
24
25
    print(z)
    # [[[ 1 7]
26
    # [28]
27
        [ 3 9]]
28
29
    # [[ 4 10]
30
31
        [ 5 11]
32
    # [6 12]]]
```

- 1. numpy.vstack(tup) Stack arrays in sequence vertically (row wise).
- 2. numpy.hstack(tup) Stack arrays in sequence horizontally (column wise).

#### 【例】一维的情况。

```
1
     import numpy as np
2
3
     x = np.array([1, 2, 3])
4
     y = np.array([7, 8, 9])
5
     z = np.vstack((x, y))
     print(z.shape) # (2, 3)
6
7
     print(z)
8
     # [[1 2 3]
     # [7 8 9]]
9
10
     z = np.stack([x, y])
11
12
     print(z.shape) # (2, 3)
13
     print(z)
     # [[1 2 3]
14
15
     # [7 8 9]]
```

```
16
17
     z = np.hstack((x, y))
18
     print(z.shape) # (6,)
19
     print(z)
     # [1 2 3 7 8 9]
20
21
22
     z = np.concatenate((x, y))
23
     print(z.shape) # (6,)
24
     print(z) # [1 2 3 7 8 9]
```

### 【例】二维的情况。

```
1
    import numpy as np
2
3
     x = np.array([1, 2, 3]).reshape(1, 3)
     y = np.array([7, 8, 9]).reshape(1, 3)
4
     z = np.vstack((x, y))
    print(z.shape) # (2, 3)
6
7
     print(z)
     # [[1 2 3]
8
9
     # [7 8 9]]
10
11
     z = np.concatenate((x, y), axis=0)
12
     print(z.shape) # (2, 3)
13
     print(z)
     # [[1 2 3]
14
15
     # [7 8 9]]
16
17
     z = np.hstack((x, y))
18
     print(z.shape) # (1, 6)
19
     print(z)
     # [[ 1 2 3 7 8 9]]
20
21
22
     z = np.concatenate((x, y), axis=1)
23
    print(z.shape) # (1, 6)
24
     print(z)
     # [[1 2 3 7 8 9]]
```

### 【例】二维的情况。

```
1
     import numpy as np
2
     x = np.array([[1, 2, 3], [4, 5, 6]])
3
     y = np.array([[7, 8, 9], [10, 11, 12]])
4
     z = np.vstack((x, y))
5
     print(z.shape) # (4, 3)
6
7
     print(z)
8
     # [[ 1 2 3]
     # [ 4 5 6]
9
10
     # [789]
11
     # [10 11 12]]
12
13
     z = np.concatenate((x, y), axis=0)
```

```
14
    print(z.shape) # (4, 3)
15
    print(z)
    # [[ 1 2 3]
16
17
    # [ 4 5 6]
    # [7 8 9]
18
    # [10 11 12]]
19
20
21
    z = np.hstack((x, y))
    print(z.shape) # (2, 6)
22
    print(z)
23
    #[[1 2 3 7 8 9]
24
    # [ 4 5 6 10 11 12]]
25
26
27
    z = np.concatenate((x, y), axis=1)
28
    print(z.shape) # (2, 6)
    print(z)
29
    #[[123789]
30
    # [ 4 5 6 10 11 12]]
31
```

hstack(),vstack() 分别表示水平和竖直的拼接方式。在数据维度等于1时,比较特殊。而当维度大于或等于2时,它们的作用相当于 concatenate,用于在已有轴上进行操作。

#### 【例】

```
import numpy as np

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

print(a) # [1 2 3 4 5]

a = np.concatenate([np.array([1, 2, 3, 4]), 5])

print(a)

# all the input arrays must have same number of dimensions, but the array at index 0 has 1 dimension(s) and the array at index 1 has 0 dimension(s)
```

# 9.5 数组拆分

1. numpy.split(ary, indices\_or\_sections, axis=0) Split an array into multiple sub-arrays as views into ary.

### 【例】拆分数组。

```
import numpy as np
     x = np.array([[11, 12, 13, 14],
3
4
                   [16, 17, 18, 19],
5
                   [21, 22, 23, 24]])
    y = np.split(x, [1, 3])
6
7
     print(y)
8
     # [array([[11, 12, 13, 14]]), array([[16, 17, 18, 19],
9
             [21, 22, 23, 24]]), array([], shape=(0, 4), dtype=int32)]
10
```

```
11
     y = np.split(x, [1, 3], axis=1)
12
     print(y)
13
     # [array([[11],
14
              [16],
15
              [21]]), array([[12, 13],
16
              [17, 18],
17
              [22, 23]]), array([[14],
              [19],
18
19
              [24]])]
     #
```

1. numpy.vsplit(ary, indices\_or\_sections) Split an array into multiple sub-arrays vertically (row-wise).

#### 【例】垂直切分是把数组按照高度切分

```
1
     import numpy as np
 2
3
     x = np.array([[11, 12, 13, 14],
4
                   [16, 17, 18, 19],
5
                   [21, 22, 23, 24]])
6
     y = np.vsplit(x, 3)
7
     print(y)
8
     # [array([[11, 12, 13, 14]]), array([[16, 17, 18, 19]]), array([[21, 22, 23, 24]])]
9
     y = np.split(x, 3)
10
11
     print(y)
     # [array([[11, 12, 13, 14]]), array([[16, 17, 18, 19]]), array([[21, 22, 23, 24]])]
12
13
14
15
     y = np.vsplit(x, [1])
     print(y)
16
17
     # [array([[11, 12, 13, 14]]), array([[16, 17, 18, 19],
18
              [21, 22, 23, 24]])]
19
20
     y = np.split(x, [1])
21
     print(y)
22
     # [array([[11, 12, 13, 14]]), array([[16, 17, 18, 19],
              [21, 22, 23, 24]])]
23
24
25
     y = np.vsplit(x, [1, 3])
26
27
     print(y)
28
     # [array([[11, 12, 13, 14]]), array([[16, 17, 18, 19],
29
              [21, 22, 23, 24]]), array([], shape=(0, 4), dtype=int32)]
     y = np.split(x, [1, 3], axis=0)
30
31
     print(y)
     # [array([[11, 12, 13, 14]]), array([[16, 17, 18, 19],
32
33
              [21, 22, 23, 24]]), array([], shape=(0, 4), dtype=int32)]
```

1. numpy.hsplit(ary, indices\_or\_sections) Split an array into multiple sub-arrays horizontally (column-wise).

### 【例】水平切分是把数组按照宽度切分。

```
2
3
     x = np.array([[11, 12, 13, 14],
                   [16, 17, 18, 19],
4
5
                   [21, 22, 23, 24]])
6
     y = np.hsplit(x, 2)
7
     print(y)
     # [array([[11, 12],
8
9
              [16, 17],
             [21, 22]]), array([[13, 14],
10
             [18, 19],
11
12
             [23, 24]])]
13
     y = np.split(x, 2, axis=1)
14
15
     print(y)
16
     # [array([[11, 12],
             [16, 17],
17
             [21, 22]]), array([[13, 14],
18
             [18, 19],
19
             [23, 24]])]
20
21
22
     y = np.hsplit(x, [3])
23
     print(y)
     # [array([[11, 12, 13],
24
25
             [16, 17, 18],
             [21, 22, 23]]), array([[14],
26
27
             [19],
28
            [24]])]
29
30
     y = np.split(x, [3], axis=1)
31
     print(y)
32
     # [array([[11, 12, 13],
             [16, 17, 18],
33
34
             [21, 22, 23]]), array([[14],
35
             [19],
             [24]])]
36
37
38
     y = np.hsplit(x, [1, 3])
39
     print(y)
     # [array([[11],
40
41
             [16],
             [21]]), array([[12, 13],
42
             [17, 18],
43
             [22, 23]]), array([[14],
44
             [19],
45
             [24]])]
     #
46
47
48
     y = np.split(x, [1, 3], axis=1)
49
     print(y)
     # [array([[11],
50
51
             [16],
52
              [21]]), array([[12, 13],
              [17, 18],
53
```

```
54 # [22, 23]]), array([[14],

55 # [19],

56 # [24]])]
```

# 9.6 数组平铺

1. numpy.tile(A, reps) Construct an array by repeating A the number of times given by reps.

tile 是瓷砖的意思,顾名思义,这个函数就是把数组像瓷砖一样铺展开来。

【例】将原矩阵横向、纵向地复制。

```
1
     import numpy as np
3
    x = np.array([[1, 2], [3, 4]])
4
    print(x)
    # [[1 2]
    # [3 4]]
6
7
    y = np.tile(x, (1, 3))
8
9
    print(y)
    # [[1 2 1 2 1 2]
10
     # [3 4 3 4 3 4]]
11
12
     y = np.tile(x, (3, 1))
13
14
     print(y)
     # [[1 2]
    # [3 4]
16
17
     # [1 2]
18
    # [3 4]
19
    # [1 2]
    # [3 4]]
20
21
22
    y = np.tile(x, (3, 3))
     print(y)
23
24
    # [[1 2 1 2 1 2]
     # [3 4 3 4 3 4]
25
    # [1 2 1 2 1 2]
26
    # [3 4 3 4 3 4]
27
28
     # [1 2 1 2 1 2]
29
     # [3 4 3 4 3 4]]
```

- 1. numpy.repeat(a, repeats, axis=None) Repeat elements of an array.
  - a. axis=0,沿着y轴复制,实际上增加了行数。
  - b. axis=1,沿着x轴复制,实际上增加了列数。
  - c. repeats,可以为一个数,也可以为一个矩阵。

d. axis=None 时就会flatten当前矩阵,实际上就是变成了一个行向量。

#### 【例】重复数组的元素。

```
import numpy as np
1
2
3
     x = np.repeat(3, 4)
4
     print(x) # [3 3 3 3]
5
     x = np.array([[1, 2], [3, 4]])
7
     y = np.repeat(x, 2)
     print(y)
8
     # [1 1 2 2 3 3 4 4]
9
10
     y = np.repeat(x, 2, axis=0)
11
     print(y)
12
13
     # [[1 2]
     # [1 2]
14
     # [3 4]
15
     # [3 4]]
17
     y = np.repeat(x, 2, axis=1)
18
19
     print(y)
20
     # [[1 1 2 2]
     # [3 3 4 4]]
21
22
23
     y = np.repeat(x, [2, 3], axis=0)
24
     print(y)
25
     # [[1 2]
26
     # [1 2]
27
     # [3 4]
28
     # [3 4]
29
     # [3 4]]
30
     y = np.repeat(x, [2, 3], axis=1)
31
32
     print(y)
     # [[1 1 2 2 2]
33
34
     # [3 3 4 4 4]]
```

# 9.7 添加和删除元素

- 1. numpy.unique(ar, return\_index=False, return\_inverse=False, return\_counts=False, axis=None) Find the unique elements of an array.
  - a. return\_index: the indices of the input array that give the unique values
  - b. return\_inverse: the indices of the unique array that reconstruct the input array
  - c. return\_counts: the number of times each unique value comes up in the input array

## 【例】查找数组的唯一元素。

```
1  a=np.array([1,1,2,3,3,4,4])
2  b=np.unique(a,return_counts=True)
3  print(b[0][list(b[1]).index(1)])
4  #2
```

### 参考文献

1. https://blog.csdn.net/csdn15698845876/article/details/73380803

# 10 向量化和广播

向量化和广播这两个概念是 numpy 内部实现的基础。有了向量化,编写代码时无需使用显式循环。这些循环实际上不能省略,只不过是在内部实现,被代码中的其他结构代替。向量化的应用使得代码更简洁,可读性更强,也可以说使用了向量化方法的代码看上去更"Pythonic"。

广播(Broadcasting)机制描述了 numpy 如何在算术运算期间处理具有不同形状的数组,让较小的数组在较大的数组上"广播",以便它们具有兼容的形状。并不是所有的维度都要彼此兼容才符合广播机制的要求,但它们必须满足一定的条件。

若两个数组的各维度兼容,也就是两个数组的每一维等长,或其中一个数组为一维,那么广播机制就适用。如果这两个条件不满足,numpy就会抛出异常,说两个数组不兼容。

总结来说,广播的规则有三个:

- 1. 如果两个数组的维度数dim不相同,那么小维度数组的形状将会在左边补1。
- 2. 如果shape维度不匹配, 但是有维度是1, 那么可以扩展维度是1的维度匹配另一个数组;
- 3. 如果shape维度不匹配,但是没有任何一个维度是1,则匹配引发错误;

#### 【例】二维数组加一维数组

```
import numpy as np
2
3
    x = np.arange(4)
4
    y = np.ones((3, 4))
5
    print(x.shape) # (4,)
     print(y.shape) # (3, 4)
6
8
    print((x + y).shape) # (3, 4)
    print(x + y)
9
    # [[1. 2. 3. 4.]
10
11
    # [1. 2. 3. 4.]
   # [1. 2. 3. 4.]]
12
```

## 【例】两个数组均需要广播

```
1
     import numpy as np
2
3
    x = np.arange(4).reshape(4, 1)
4
    y = np.ones(5)
5
     print(x.shape) # (4, 1)
6
7
     print(y.shape) # (5,)
8
9
     print((x + y).shape) # (4, 5)
     print(x + y)
10
    # [[1. 1. 1. 1. 1.]
11
    # [2. 2. 2. 2. 2.]
12
13
    # [3. 3. 3. 3. 3.]
14
    # [4. 4. 4. 4. 4.]]
```

```
15
16
    x = np.array([0.0, 10.0, 20.0, 30.0])
    y = np.array([1.0, 2.0, 3.0])
17
    z = x[:, np.newaxis] + y
18
19
    print(z)
    # [[ 1. 2. 3.]
20
    # [11. 12. 13.]
21
    # [21. 22. 23.]
22
23 # [31. 32. 33.]]
```

## 【例】不匹配报错的例子

```
1
    import numpy as np
2
3
    x = np.arange(4)
4
    y = np.ones(5)
5
6
    print(x.shape) # (4,)
7
    print(y.shape) # (5,)
8
9
    print(x + y)
10
    # ValueError: operands could not be broadcast together with shapes (4,) (5,)
```

# 11 数学函数

## 11.1 算数运算

## 11.1.1 numpy.add

## 11.1.2 numpy.subtract

## 11.1.3 numpy.multiply

## 11.1.4 numpy.divide

# 11.1.5 numpy.floor\_divide

# 11.1.6 numpy.power

- 1. numpy.add(x1, x2, \*args, \*\*kwargs) Add arguments element-wise.
- 2. numpy.subtract(x1, x2, \*args, \*\*kwargs) Subtract arguments element-wise.
- 3. numpy.multiply(x1, x2, \*args, \*\*kwargs) Multiply arguments element-wise.
- 4. numpy.divide(x1, x2, \*args, \*\*kwargs) Returns a true division of the inputs, element-wise.
- 5. numpy.floor\_divide(x1, x2, \*args, \*\*kwargs) Return the largest integer smaller or equal to the division of the inputs.
- 6. numpy.power(x1, x2, \*args, \*\*kwargs) First array elements raised to powers from second array, element-wise.

在 numpy 中对以上函数进行了运算符的重载,且运算符为 **元素级**。也就是说,它们只用于位置相同的元素之间,所得到的运算结果组成一个新的数组。

### 【例】注意 numpy 的广播规则。

```
3
     x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
 4
     y = x + 1
 5
     print(y)
 6
     print(np.add(x, 1))
 7
     # [2 3 4 5 6 7 8 9]
 8
 9
     y = x - 1
10
     print(y)
     print(np.subtract(x, 1))
11
     # [0 1 2 3 4 5 6 7]
12
13
     y = x * 2
14
15
     print(y)
16
     print(np.multiply(x, 2))
17
     # [ 2 4 6 8 10 12 14 16]
18
19
     y = x / 2
20
     print(y)
     print(np.divide(x, 2))
21
     # [0.5 1. 1.5 2. 2.5 3. 3.5 4.]
22
23
24
     y = x // 2
     print(y)
25
26
     print(np.floor_divide(x, 2))
     # [0 1 1 2 2 3 3 4]
27
28
29
     y = x ** 2
30
     print(y)
31
     print(np.power(x, 2))
32
     # [ 1 4 9 16 25 36 49 64]
```

#### 【例】注意 numpy 的广播规则。

```
1
     import numpy as np
 2
 3
     x = np.array([[11, 12, 13, 14, 15],
 4
                   [16, 17, 18, 19, 20],
 5
                   [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30],
 6
 7
                   [31, 32, 33, 34, 35]])
 8
     y = x + 1
 9
     print(y)
10
     print(np.add(x, 1))
     # [[12 13 14 15 16]
11
     # [17 18 19 20 21]
12
     # [22 23 24 25 26]
13
14
     # [27 28 29 30 31]
     # [32 33 34 35 36]]
15
16
17
     y = x - 1
18
     print(y)
19
     print(np.subtract(x, 1))
```

```
20
    # [[10 11 12 13 14]
    # [15 16 17 18 19]
21
    # [20 21 22 23 24]
22
23
    # [25 26 27 28 29]
    # [30 31 32 33 34]]
24
25
26
    y = x * 2
27
    print(y)
    print(np.multiply(x, 2))
28
    # [[22 24 26 28 30]
29
30
    # [32 34 36 38 40]
31
    # [42 44 46 48 50]
    # [52 54 56 58 60]
32
33
    # [62 64 66 68 70]]
34
35
    y = x / 2
36
    print(y)
37
    print(np.divide(x, 2))
    # [[ 5.5 6. 6.5 7. 7.5]
38
     # [8. 8.5 9. 9.5 10.]
39
    # [10.5 11. 11.5 12. 12.5]
40
    # [13. 13.5 14. 14.5 15.]
41
    # [15.5 16. 16.5 17. 17.5]]
42
43
44
    y = x // 2
45
    print(y)
    print(np.floor_divide(x, 2))
    # [[ 5 6 6 7 7]
47
    # [889910]
48
49
    # [10 11 11 12 12]
50
    # [13 13 14 14 15]
51
    # [15 16 16 17 17]]
52
    y = x ** 2
53
54
    print(y)
55
    print(np.power(x, 2))
56
    # [[ 121 144 169 196 225]
    # [ 256 289 324 361 400]
57
58
    # [ 441 484 529 576 625]
59
     # [ 676 729 784 841 900]
    # [ 961 1024 1089 1156 1225]]
```

【例】注意 numpy 的广播规则。

```
1
    import numpy as np
2
3
    x = np.array([[11, 12, 13, 14, 15],
4
                  [16, 17, 18, 19, 20],
5
                  [21, 22, 23, 24, 25],
6
                  [26, 27, 28, 29, 30],
7
                  [31, 32, 33, 34, 35]])
8
9
    y = np.arange(1, 6)
```

```
10
    print(y)
    # [1 2 3 4 5]
11
12
13
    z = x + y
14
    print(z)
15
    print(np.add(x, y))
    # [[12 14 16 18 20]
17
    # [17 19 21 23 25]
    # [22 24 26 28 30]
18
    # [27 29 31 33 35]
19
20
    # [32 34 36 38 40]]
21
22
    z = x - y
23
    print(z)
24
    print(np.subtract(x, y))
    # [[10 10 10 10 10]
25
    # [15 15 15 15 15]
26
    # [20 20 20 20 20]
27
    # [25 25 25 25 25]
28
    # [30 30 30 30 30]]
29
30
31
    z = x * y
32
    print(z)
33
    print(np.multiply(x, y))
    # [[ 11 24 39 56 75]
34
35
    # [ 16 34 54 76 100]
    # [ 21 44 69 96 125]
37
    # [ 26 54 84 116 150]
38
    # [ 31 64 99 136 175]]
39
40
    z = x / y
41
    print(z)
42
    print(np.divide(x, y))
43
    # [[11.
                    6.
                               4.33333333 3.5
                                                       3.
                                                                ]
    # [16.
                    8.5
                                6.
                                           4.75
                                                       4.
                                                                ]
44
45
     # [21.
                    11.
                                7.66666667 6.
                                                       5.
                                                                ]
46
    # [26.
                    13.5
                                9.33333333 7.25
                                                       6.
                                                                ]
47
    # [31.
                    16.
                               11.
                                           8.5
                                                       7.
                                                                ]]
48
49
    z = x // y
50
    print(z)
    print(np.floor_divide(x, y))
51
    # [[11 6 4 3 3]
52
    # [16 8 6 4 4]
53
    # [21 11 7 6 5]
54
     # [26 13 9 7 6]
55
    # [31 16 11 8 7]]
56
57
    z = x ** np.full([1, 5], 2)
58
59
    print(z)
60
    print(np.power(x, np.full([5, 5], 2)))
    # [[ 121 144 169 196 225]
61
```

```
62 # [ 256 289 324 361 400]
63 # [ 441 484 529 576 625]
64 # [ 676 729 784 841 900]
65 # [ 961 1024 1089 1156 1225]]
```

```
1
     import numpy as np
2
     x = np.array([[11, 12, 13, 14, 15],
3
4
                   [16, 17, 18, 19, 20],
5
                   [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30],
6
 7
                   [31, 32, 33, 34, 35]])
8
9
     y = np.arange(1, 26).reshape([5, 5])
10
     # [[ 1 2 3 4 5]
11
     # [ 6 7 8 9 10]
12
     # [11 12 13 14 15]
13
     # [16 17 18 19 20]
14
     # [21 22 23 24 25]]
15
16
17
     z = x + y
18
     print(z)
19
     print(np.add(x, y))
     # [[12 14 16 18 20]
     # [22 24 26 28 30]
21
     # [32 34 36 38 40]
22
     # [42 44 46 48 50]
23
24
     # [52 54 56 58 60]]
25
26
     z = x - y
27
     print(z)
28
     print(np.subtract(x, y))
     # [[10 10 10 10 10]
29
30
     # [10 10 10 10 10]
31
     # [10 10 10 10 10]
     # [10 10 10 10 10]
32
33
     # [10 10 10 10 10]]
34
35
     z = x * y
36
     print(z)
37
     print(np.multiply(x, y))
     # [[ 11 24 39 56 75]
38
     # [ 96 119 144 171 200]
39
40
     # [231 264 299 336 375]
41
     # [416 459 504 551 600]
     # [651 704 759 816 875]]
42
43
44
     z = x / y
45
     print(z)
     print(np.divide(x, y))
```

```
47
    # [[11.
                       4.33333333 3.5
48
    # [ 2.66666667 2.42857143 2.25
                                     2.11111111 2.
49
    # [ 1.90909091 1.83333333 1.76923077 1.71428571 1.66666667]
50
    # [ 1.625
                 1.58823529 1.55555556 1.52631579 1.5
    51
                                                       11
52
53
    z = x // y
54
    print(z)
55
    print(np.floor_divide(x, y))
    # [[11 6 4 3 3]
56
    # [ 2 2 2 2 2]
58
    # [ 1 1 1 1 1]
    # [1 1 1 1 1]
59
60
    # [ 1 1 1 1 1]]
61
    z = x ** np.full([5, 5], 2)
62
63
    print(z)
64
    print(np.power(x, np.full([5, 5], 2)))
    # [[ 121 144 169 196 225]
65
66
    # [ 256 289 324 361 400]
    # [ 441 484 529 576 625]
67
   # [ 676 729 784 841 900]
68
    # [ 961 1024 1089 1156 1225]]
69
```

## 11.1.7 numpy.sqrt

## 11.1.8 numpy.square

```
1. numpy.sqrt(x, *args, **kwargs) Return the non-negative square-root of an array, element-wise.
```

2. numpy.square(x, \*args, \*\*kwargs) Return the element-wise square of the input.

```
import numpy as np
1
2
3
     x = np.arange(1, 5)
     print(x) # [1 2 3 4]
4
5
     y = np.sqrt(x)
6
7
     print(y)
     # [1.
8
                   1.41421356 1.73205081 2.
9
     print(np.power(x, 0.5))
10
     # [1.
                   1.41421356 1.73205081 2.
                                                   ]
11
12
     y = np.square(x)
13
     print(y)
     #[1 4 9 16]
14
```

```
15 print(np.power(x, 2))
16 # [ 1 4 9 16]
```

11.2 三角函数

11.2.1 numpy.sin

**11.2.2 numpy.cos** 

11.2.3 numpy.tan

11.2.4 numpy.arcsin

11.2.5 numpy.arccos

# 11.2.6 numpy.arctan

```
1. numpy.sin(x, *args, **kwargs) Trigonometric sine, element-wise.
```

- 2. numpy.cos(x, \*args, \*\*kwargs) Cosine element-wise.
- 3. numpy.tan(x, \*args, \*\*kwargs) Compute tangent element-wise.
- 4. numpy.arcsin(x, \*args, \*\*kwargs) Inverse sine, element-wise.
- 5. numpy.arccos(x, \*args, \*\*kwargs) Trigonometric inverse cosine, element-wise.
- 6. numpy.arctan(x, \*args, \*\*kwargs) Trigonometric inverse tangent, element-wise.

**通用函数**(universal function)通常叫作ufunc,它对数组中的各个元素逐一进行操作。这表明,通用函数分别处理输入数组的每个元素,生成的结果组成一个新的输出数组。输出数组的大小跟输入数组相同。

三角函数等很多数学运算符合通用函数的定义,例如,计算平方根的 sqrt() 函数、用来取对数的 log() 函数和求正弦值的 sin() 函数。

### 【例】

1 import numpy as np

```
x = np.linspace(start=0, stop=np.pi / 2, num=10)
4
     print(x)
                0.17453293 0.34906585 0.52359878 0.6981317 0.87266463
5
     # [0.
6
     # 1.04719755 1.22173048 1.3962634 1.57079633]
8
     y = np.sin(x)
9
     print(y)
                                            0.64278761 0.76604444
     # [0. 0.17364818 0.34202014 0.5
10
     # 0.8660254 0.93969262 0.98480775 1.
11
                                               ]
12
13
    z = np.arcsin(y)
14
     print(z)
                  0.17453293 0.34906585 0.52359878 0.6981317 0.87266463
15
     # [0.
     # 1.04719755 1.22173048 1.3962634 1.57079633]
17
     y = np.cos(x)
18
19
     print(y)
     # [1.00000000e+00 9.84807753e-01 9.39692621e-01 8.66025404e-01
20
     # 7.66044443e-01 6.42787610e-01 5.00000000e-01 3.42020143e-01
21
     # 1.73648178e-01 6.12323400e-17]
22
23
    z = np.arccos(y)
24
     print(z)
25
                0.17453293 0.34906585 0.52359878 0.6981317 0.87266463
26
     # 1.04719755 1.22173048 1.3962634 1.57079633]
27
28
29
    y = np.tan(x)
30
     print(y)
31
     # [0.00000000e+00 1.76326981e-01 3.63970234e-01 5.77350269e-01
32
     # 8.39099631e-01 1.19175359e+00 1.73205081e+00 2.74747742e+00
33
     # 5.67128182e+00 1.63312394e+16]
34
35
    z = np.arctan(y)
36
     print(z)
37
     # [0.
                0.17453293 0.34906585 0.52359878 0.6981317 0.87266463
38
     # 1.04719755 1.22173048 1.3962634 1.57079633]
```

# 11.3 指数和对数

## **11.3.1 numpy.exp**

# **11.3.2 numpy.log**

## 11.3.3 numpy.exp2

## 11.3.4 numpy.log2

## 11.3.5 numpy.log10

```
    numpy.exp(x, *args, **kwargs) Calculate the exponential of all elements in the input array.
    numpy.log(x, *args, **kwargs) Natural logarithm, element-wise.
    numpy.exp2(x, *args, **kwargs) Calculate 2**p for all p in the input array.
    numpy.log2(x, *args, **kwargs) Base-2 logarithm of x.
    numpy.log10(x, *args, **kwargs) Return the base 10 logarithm of the input array, element-wise.
```

[1] The natural logarithm log is the inverse of the exponential function, so that log(exp(x)) = x. The natural logarithm is logarithm in base e.

```
1
     import numpy as np
2
3
     x = np.arange(1, 5)
4
     print(x)
    # [1 2 3 4]
    y = np.exp(x)
7
     print(y)
     # [ 2.71828183 7.3890561 20.08553692 54.59815003]
     z = np.log(y)
9
     print(z)
10
     # [1. 2. 3. 4.]
11
```

# 11.4 加法函数、乘法函数

## **11.4.1 numpy.sum**

1. numpy.sum(a[, axis=None, dtype=None, out=None, ...]) Sum of array elements over a given axis.

通过不同的 axis, numpy 会沿着不同的方向进行操作:如果不设置,那么对所有的元素操作;如果 axis=0,则沿着纵轴进行操作; axis=1,则沿着横轴进行操作。但这只是简单的二位数组,如果是多维的呢?可以总结为一句话:设 axis=i,则 numpy 沿着第 i 个下标变化的方向进行操作。

## 11.4.2 numpy.cumsum

1. numpy.cumsum(a, axis=None, dtype=None, out=None) Return the cumulative sum of the elements along a given axis.

**聚合函数** 是指对一组值(比如一个数组)进行操作,返回一个单一值作为结果的函数。因而,求数组所有元素之和的函数就是聚合函数。 **ndarray** 类实现了多个这样的函数。

【例】返回给定轴上的数组元素的总和。

```
1
     import numpy as np
 2
3
     x = np.array([[11, 12, 13, 14, 15],
                   [16, 17, 18, 19, 20],
4
5
                   [21, 22, 23, 24, 25],
6
                   [26, 27, 28, 29, 30],
                   [31, 32, 33, 34, 35]])
7
8
     y = np.sum(x)
9
     print(y) # 575
10
11
     y = np.sum(x, axis=0)
12
     print(y) # [105 110 115 120 125]
13
     y = np.sum(x, axis=1)
14
     print(y) # [ 65 90 115 140 165]
15
```

【例】返回给定轴上的数组元素的累加和。

```
1
    import numpy as np
2
3
    x = np.array([[11, 12, 13, 14, 15],
                  [16, 17, 18, 19, 20],
4
5
                  [21, 22, 23, 24, 25],
6
                  [26, 27, 28, 29, 30],
                  [31, 32, 33, 34, 35]])
7
8
    y = np.cumsum(x)
    print(y)
9
10
    # [ 11 23 36 50 65 81 98 116 135 155 176 198 221 245 270 296 323 351
    # 380 410 441 473 506 540 575]
11
12
13
    y = np.cumsum(x, axis=0)
14
    print(y)
    # [[ 11 12 13 14 15]
15
    # [ 27 29 31 33 35]
17
    # [ 48 51 54 57 60]
    # [ 74 78 82 86 90]
18
    # [105 110 115 120 125]]
19
20
21
    y = np.cumsum(x, axis=1)
    print(y)
22
    # [[ 11 23 36 50 65]
24
    # [ 16 33 51 70 90]
    # [ 21 43 66 90 115]
25
    # [ 26 53 81 110 140]
```

## 11.4.3 numpy.prod 乘积

1. numpy.prod(a[, axis=None, dtype=None, out=None, ...]) Return the product of array elements over a given axis.

## 11.4.4 numpy.cumprod 累乘

1. numpy.cumprod(a, axis=None, dtype=None, out=None) Return the cumulative product of elements along a given axis.

【例】返回给定轴上数组元素的乘积。

```
1
     import numpy as np
2
3
     x = np.array([[11, 12, 13, 14, 15],
4
                   [16, 17, 18, 19, 20],
                   [21, 22, 23, 24, 25],
5
6
                   [26, 27, 28, 29, 30],
7
                   [31, 32, 33, 34, 35]])
8
     y = np.prod(x)
9
     print(y) # 788529152
10
11
     y = np.prod(x, axis=0)
12
13
     # [2978976 3877632 4972968 6294624 7875000]
14
15
     y = np.prod(x, axis=1)
16
     print(y)
     # [ 360360 1860480 6375600 17100720 38955840]
17
```

【例】返回给定轴上数组元素的累乘。

```
1
     import numpy as np
2
3
     x = np.array([[11, 12, 13, 14, 15],
4
                  [16, 17, 18, 19, 20],
                  [21, 22, 23, 24, 25],
5
                  [26, 27, 28, 29, 30],
6
7
                  [31, 32, 33, 34, 35]])
8
     y = np.cumprod(x)
9
     print(y)
10
                11
                           132
                                     1716
                                                24024
                                                          360360
                                                                     5765760
        98017920 1764322560 -837609728 427674624
                                                       391232512
                                                                   17180672
11
12
         395155456 893796352 870072320 1147043840
                                                       905412608 -418250752
         755630080 1194065920 -1638662144 -897581056 444596224 -2063597568
13
14
         788529152]
15
     y = np.cumprod(x, axis=0)
16
17
     print(y)
```

```
18
     # [[
             11
                    12
                           13
                                  14
                                          15]
19
            176
                    204
                           234
                                   266
                                          3001
20
           3696
                   4488
                           5382
                                  6384
                                          7500]
21
     # [ 96096 121176 150696 185136 225000]
22
    # [2978976 3877632 4972968 6294624 7875000]]
23
24
    y = np.cumprod(x, axis=1)
25
    print(y)
    # [[
                      132
                             1716
                                     24024 360360]
26
              11
    # [
27
              16
                      272
                             4896
                                     93024 1860480]
28
              21
                      462
                            10626
                                    255024 6375600]
29
     # [
              26
                      702
                            19656
                                    570024 17100720]
                            32736 1113024 38955840]]
    # [
              31
                      992
30
```

## 11.4.5 numpy.diff 差值

```
    numpy.diff(a, n=1, axis=-1, prepend=np._NoValue, append=np._NoValue)
    Calculate the n-th discrete difference along the given axis.
    a. a: 输入矩阵
    b. n: 可选,代表要执行几次差值
    c. axis: 默认是最后一个
```

The first difference is given by out[i] = a[i+1] - a[i] along the given axis, higher differences are calculated by using diff recursively.

【例】沿着指定轴计算第N维的离散差值。

```
1
     import numpy as np
2
3
     A = np.arange(2, 14).reshape((3, 4))
     A[1, 1] = 8
4
5
     print(A)
     # [[ 2 3 4 5]
     # [6 8 8 9]
7
    # [10 11 12 13]]
8
9
     print(np.diff(A))
    # [[1 1 1]
10
    # [2 0 1]
11
    # [1 1 1]]
12
     print(np.diff(A, axis=0))
13
     # [[4 5 4 4]
14
15
     # [4 3 4 4]]
```

# 11.5 四舍五入

## 11.5.1 numpy.around 舍入

1. numpy.around(a, decimals=0, out=None) Evenly round to the given number of decimals.

【例】将数组舍入到给定的小数位数。

```
import numpy as np
2
3
    x = np.random.rand(3, 3) * 10
4
    print(x)
    # [[6.59144457 3.78566113 8.15321227]
5
6
     # [1.68241475 3.78753332 7.68886328]
7
    # [2.84255822 9.58106727 7.86678037]]
8
9
    y = np.around(x)
10
    print(y)
    # [[ 7. 4. 8.]
11
     # [ 2. 4. 8.]
12
     # [ 3. 10. 8.]]
13
14
     y = np.around(x, decimals=2)
15
    print(y)
16
    # [[6.59 3.79 8.15]
17
     # [1.68 3.79 7.69]
18
19
    # [2.84 9.58 7.87]]
```

# 11.5.2 numpy.ceil 上限

# 11.5.3 numpy.floor 下限

```
1. numpy.ceil(x, *args, **kwargs) Return the ceiling of the input, element-wise.
```

2. numpy.floor(x, \*args, \*\*kwargs) Return the floor of the input, element-wise.

```
1
     import numpy as np
2
     x = np.random.rand(3, 3) * 10
3
     print(x)
5
     # [[0.67847795 1.33073923 4.53920122]
     # [7.55724676 5.88854047 2.65502046]
6
     # [8.67640444 8.80110812 5.97528726]]
8
     y = np.ceil(x)
9
10
     print(y)
11
     # [[1. 2. 5.]
12
     # [8. 6. 3.]
```

```
13 # [9. 9. 6.]]
14
15 y = np.floor(x)
16 print(y)
17 # [[0. 1. 4.]
18 # [7. 5. 2.]
19 # [8. 8. 5.]]
```

## 11.6 杂项

## 11.6.1 numpy.clip 裁剪

```
1. numpy.clip(a, a_min, a_max, out=None, **kwargs): Clip (limit) the values in an array.
```

Given an interval, values outside the interval are clipped to the interval edges. For example, if an interval of [0, 1] is specified, values smaller than 0 become 0, and values larger than 1 become 1.

【例】裁剪(限制)数组中的值。

```
1
     import numpy as np
3
     x = np.array([[11, 12, 13, 14, 15],
                   [16, 17, 18, 19, 20],
4
                   [21, 22, 23, 24, 25],
5
                   [26, 27, 28, 29, 30],
6
                   [31, 32, 33, 34, 35]])
7
     y = np.clip(x, a_min=20, a_max=30)
8
9
     print(y)
     # [[20 20 20 20 20]
10
     # [20 20 20 20 20]
11
12
     # [21 22 23 24 25]
13
     # [26 27 28 29 30]
     # [30 30 30 30 30]]
14
```

# 11.6.2 numpy.absolute 绝对值

# 11.6.3 numpy.abs

```
1. numpy.absolute(x, *args, **kwargs) Calculate the absolute value element-wise.
```

```
2. numpy.abs(x, *args, **kwargs) is a shorthand for this function.
```

```
1
     import numpy as np
2
3
    x = np.arange(-5, 5)
4
    print(x)
    # [-5 -4 -3 -2 -1 0 1 2 3 4]
5
7
    y = np.abs(x)
8
     print(y)
     # [5 4 3 2 1 0 1 2 3 4]
9
10
    y = np.absolute(x)
11
12
     print(y)
     # [5 4 3 2 1 0 1 2 3 4]
13
```

# 11.6.4 numpy.sign 返回数字符号的逐元素指示

1. numpy.sign(x, \*args, \*\*kwargs) Returns an element-wise indication of the sign of a number.

#### 【例】

```
1  x = np.arange(-5, 5)
2  print(x)
3  #[-5 -4 -3 -2 -1 0 1 2 3 4]
4  print(np.sign(x))
5  #[-1 -1 -1 -1 -1 0 1 1 1 1]
```

#### 参考文献

- 1. https://mp.weixin.qq.com/s/RWsGvvmw4ptf7d8zPIDEJw
- $2. \ \underline{\text{https://blog.csdn.net/hanshuobest/article/details/78558826?utm\_medium=distribute.pc\_relevant.none-task-blog-baidujs-1}\\$

# 12 逻辑函数

## 12.1 真值测试

## **12.1.1 numpy.all**

### **12.1.2 numpy.any**

```
1. numpy.all(a, axis=None, out=None, keepdims=np._NoValue) Test whether all array elements along a given axis evaluate to True.

2. numpy.any(a, axis=None, out=None, keepdims=np._NoValue) Test whether any array element along a given axis evaluates to True.
```

#### 【例】

```
1
     import numpy as np
2
3
     a = np.array([0, 4, 5])
4
     b = np.copy(a)
     print(np.all(a == b)) # True
5
     print(np.any(a == b)) # True
7
8
     b[0] = 1
9
     print(np.all(a == b)) # False
10
     print(np.any(a == b)) # True
11
     print(np.all([1.0, np.nan])) # True
12
13
     print(np.any([1.0, np.nan])) # True
14
15
     a = np.eye(3)
     print(np.all(a, axis=0)) # [False False False]
16
17
     print(np.any(a, axis=0)) # [ True True True]
```

# 12.2 数组内容

# 12.2.1 numpy.isnan

1. numpy.isnan(x, \*args, \*\*kwargs) Test element-wise for NaN and return result as a boolean array.

#### 【例】

```
1 a=np.array([1,2,np.nan])
2 print(np.isnan(a))
3 #[False False True]
4
```

## 12.3 逻辑运算

### 12.3.1 numpy.logical\_not

## 12.3.2 numpy.logical\_and

### 12.3.3 numpy.logical\_or

## 12.3.4 numpy.logical\_xor

```
1. numpy.logical_not(x, *args, **kwargs) Compute the truth value of NOT x element-wise.
```

- 2. numpy.logical\_and(x1, x2, \*args, \*\*kwargs) Compute the truth value of x1 AND x2 element-wise.
- 3. numpy.logical\_or(x1, x2, \*args, \*\*kwargs) Compute the truth value of x1 OR x2 element-wise.
- 4. numpy.logical\_xor(x1, x2, \*args, \*\*kwargs) Compute the truth value of x1 XOR x2, element-wise.

```
【例】计算非x元素的真值。
1
2
3
    import numpy as np
4
5
    print(np.logical_not(3))
6
7
    print(np.logical_not([True, False, 0, 1]))
    # [False True True False]
8
9
10
    x = np.arange(5)
    print(np.logical_not(x < 3))</pre>
11
    # [False False True True]
```

```
13
14
     【例】计算x1 AND x2元素的真值。
15
16
     print(np.logical_and(True, False))
17
     print(np.logical_and([True, False], [True, False]))
18
     # [ True False]
19
20
     print(np.logical_and(x > 1, x < 4))
     # [False False True True False]
21
22
     【例】逐元素计算x1 OR x2的真值。
23
24
25
26
     print(np.logical_or(True, False))
27
     print(np.logical_or([True, False], [False, False]))
28
     # [ True False]
29
     print(np.logical_or(x < 1, x > 3))
30
     # [ True False False False True]
31
32
33
     【例】计算x1 XOR x2的真值,按元素计算。
34
35
     print(np.logical_xor(True, False))
     print(np.logical_xor([True, True, False, False], [True, False, True, False]))
37
38
    # [False True True False]
    print(np.logical_xor(x < 1, x > 3))
40
    # [ True False False False True]
    print(np.logical_xor(0, np.eye(2)))
41
42
    # [[ True False]
    # [False True]]
```

# 12.4 对照

# 12.4.1 numpy.greater

# 12.4.2 numpy.greater\_equal

## 12.4.3 numpy.equal

### 12.4.4 numpy.not\_equal

## 12.4.5 numpy.less

### 12.4.6 numpy.less\_equal

```
    numpy.greater(x1, x2, *args, **kwargs) Return the truth value of (x1 > x2) element-wise.
    numpy.greater_equal(x1, x2, *args, **kwargs) Return the truth value of (x1 >= x2) element-wise.
    numpy.equal(x1, x2, *args, **kwargs) Return (x1 == x2) element-wise.
    numpy.not_equal(x1, x2, *args, **kwargs) Return (x1 != x2) element-wise.
    numpy.less(x1, x2, *args, **kwargs) Return the truth value of (x1 < x2) element-wise.</li>
    numpy.less_equal(x1, x2, *args, **kwargs) Return the truth value of (x1 =< x2) element-wise.</li>
```

【例】numpy对以上对照函数进行了运算符的重载。

```
1
     import numpy as np
3
    x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
4
5
    y = x > 2
6
    print(y)
7
    print(np.greater(x, 2))
8
    # [False False True True True True True]
9
    y = x >= 2
10
11
    print(y)
12
    print(np.greater_equal(x, 2))
13
    # [False True True True True True True]
14
    y = x == 2
15
    print(y)
16
    print(np.equal(x, 2))
17
     # [False True False False False False False]
19
20
    y = x != 2
21
    print(y)
22
    print(np.not_equal(x, 2))
23
    # [ True False True True True True True]
24
    y = x < 2
25
26
    print(y)
27
    print(np.less(x, 2))
    # [ True False False False False False False]
28
29
```

```
30  y = x <= 2
31  print(y)
32  print(np.less_equal(x, 2))
33  # [ True True False False False False False]</pre>
```

```
1
    import numpy as np
2
3
    x = np.array([[11, 12, 13, 14, 15],
4
                [16, 17, 18, 19, 20],
5
                [21, 22, 23, 24, 25],
                [26, 27, 28, 29, 30],
6
                [31, 32, 33, 34, 35]])
8
    y = x > 20
9
    print(y)
10
    print(np.greater(x, 20))
    # [[False False False False]
11
12
    # [False False False False]
13
    # [ True True True True]
14
    # [ True True True True]
    # [ True True True True]]
15
16
17
    y = x >= 20
18
    print(y)
19
    print(np.greater_equal(x, 20))
    # [[False False False False]
21
    # [False False False True]
    # [ True True True True]
22
23
    # [ True True True True]
24
    # [ True True True True]]
25
26
    y = x == 20
27
    print(y)
    print(np.equal(x, 20))
28
    # [[False False False False]
29
30
    # [False False False True]
31
    # [False False False False]
    # [False False False False]
32
33
    # [False False False False]]
34
    y = x != 20
35
36
    print(y)
37
    print(np.not_equal(x, 20))
    # [[ True True True True]
38
39
    # [ True True True False]
40
    # [ True True True True]
41
    # [ True True True True]
    # [ True True True True]]
42
43
44
45
    y = x < 20
    print(y)
```

```
47
    print(np.less(x, 20))
48
    # [[ True True True True]
49
    # [ True True True False]
50
    # [False False False False]
    # [False False False False]
51
    # [False False False False]]
52
53
54
    y = x <= 20
55
    print(y)
    print(np.less_equal(x, 20))
56
57
    # [[ True True True True]
    # [ True True True True]
58
    # [False False False False]
59
60
    # [False False False False]
61
    # [False False False False]]
```

```
1
    import numpy as np
2
3
    np.random.seed(20200611)
4
     x = np.array([[11, 12, 13, 14, 15],
                  [16, 17, 18, 19, 20],
5
6
                  [21, 22, 23, 24, 25],
 7
                  [26, 27, 28, 29, 30],
                  [31, 32, 33, 34, 35]])
8
9
10
    y = np.random.randint(10, 40, [5, 5])
    print(y)
11
    # [[32 28 31 33 37]
12
    # [23 37 37 30 29]
13
    # [32 24 10 33 15]
14
15
     # [27 17 10 36 16]
    # [25 32 23 39 34]]
16
17
18
    z = x > y
19
    print(z)
20
    print(np.greater(x, y))
    # [[False False False False]
21
22
     # [False False False False]
23
    # [False False True False True]
    # [False True True False True]
24
25
    # [ True False True False True]]
26
27
    z = x >= y
28
    print(z)
29
    print(np.greater_equal(x, y))
30
    # [[False False False False]
    # [False False False False]
31
32
     # [False False True False True]
33
    # [False True True False True]
    # [ True True True False True]]
34
35
```

```
36
    z = x == y
37
    print(z)
38
    print(np.equal(x, y))
39
    # [[False False False False]
    # [False False False False]
40
    # [False False False False]
41
42
    # [False False False False]
43
    # [False True False False False]]
44
45
    z = x != y
46
    print(z)
47
    print(np.not_equal(x, y))
    # [[ True True True True]
48
49
    # [ True True True True]
50
    # [ True True True True]
    # [ True True True True]
51
52
    # [ True False True True]]
53
54
    z = x < y
55
    print(z)
56
    print(np.less(x, y))
    # [[ True True True True]
57
58
    # [ True True True True]
59
    # [ True True False True False]
    # [ True False False True False]
60
    # [False False False True False]]
61
62
63
    z = x <= y
64
    print(z)
    print(np.less_equal(x, y))
    # [[ True True True True]
66
67
    # [ True True True True]
68
    # [ True True False True False]
69
    # [ True False False True False]
70
    # [False True False True False]]
```

【例】注意 numpy 的广播规则。

```
1
     import numpy as np
 2
3
     x = np.array([[11, 12, 13, 14, 15],
4
                   [16, 17, 18, 19, 20],
 5
                    [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30],
6
 7
                   [31, 32, 33, 34, 35]])
8
9
     np.random.seed(20200611)
10
     y = np.random.randint(10, 50, 5)
11
12
     print(y)
13
     # [32 37 30 24 10]
14
15
     z = x > y
```

```
16
    print(z)
17
    print(np.greater(x, y))
18
    # [[False False False True]
19
    # [False False False True]
    # [False False False True]
20
    # [False False False True True]
21
22
    # [False False True True]]
23
24
    z = x >= y
25
    print(z)
    print(np.greater_equal(x, y))
26
27
    # [[False False False True]
    # [False False False True]
28
29
    # [False False False True True]
30
    # [False False False True True]
    # [False False True True]]
31
32
33
    z = x == y
34
    print(z)
35
    print(np.equal(x, y))
    # [[False False False False]
36
    # [False False False False]
37
    # [False False False True False]
38
    # [False False False False]
39
    # [False False False False]]
40
41
42
    z = x != y
43
    print(z)
    print(np.not_equal(x, y))
44
45
    # [[ True True True True]
    # [ True True True True]
46
47
    # [ True
              True True False True]
48
    # [ True True True True]
49
    # [ True True True True]]
50
51
    z = x < y
52
    print(z)
    print(np.less(x, y))
53
54
    # [[ True True True False]
55
    # [ True True True False]
    # [ True True True False False]
56
    # [ True True True False False]
57
58
    # [ True True False False False]]
59
60
    z = x <= y
61
    print(z)
62
    print(np.less_equal(x, y))
    # [[ True True True False]
63
64
    # [ True True True False]
65
    # [ True
              True True True False]
    # [ True True True False False]
66
67
    # [ True True False False False]]
```

### 12.4.7 numpy.isclose

### 12.4.8 numpy.allclose

- 1. numpy.isclose(a, b, rtol=1.e-5, atol=1.e-8, equal\_nan=False) Returns a boolean array where two arrays are element-wise equal within a tolerance.
- 2. numpy.allclose(a, b, rtol=1.e-5, atol=1.e-8, equal\_nan=False) Returns True if two arrays are element-wise equal within a tolerance.

```
numpy.allclose() 等价于 numpy.all(isclose(a, b, rtol=rtol, atol=atol, equal_nan=equal_nan))。
```

The tolerance values are positive, typically very small numbers. The relative difference (rtol \* abs(b)) and the absolute difference atol are added together to compare against the absolute difference between a and b.

判断是否为True的计算依据:

```
1 np.absolute(a - b) <= (atol + rtol * absolute(b))
2
3 - atol: float, 绝对公差。
4 - rtol: float, 相对公差。
```

NaNs are treated as equal if they are in the same place and if equal\_nan=True. Infs are treated as equal if they are in the same place and of the same sign in both arrays.

#### 【例】比较两个数组是否可以认为相等。

```
import numpy as np
2
     x = np.isclose([1e10, 1e-7], [1.00001e10, 1e-8])
3
4
     print(x) # [ True False]
     x = np.allclose([1e10, 1e-7], [1.00001e10, 1e-8])
6
     print(x) # False
7
8
9
     x = np.isclose([1e10, 1e-8], [1.00001e10, 1e-9])
10
     print(x) # [ True True]
11
12
     x = np.allclose([1e10, 1e-8], [1.00001e10, 1e-9])
13
     print(x) # True
14
15
     x = np.isclose([1e10, 1e-8], [1.0001e10, 1e-9])
     print(x) # [False True]
16
17
18
     x = np.allclose([1e10, 1e-8], [1.0001e10, 1e-9])
19
     print(x) # False
20
21
     x = np.isclose([1.0, np.nan], [1.0, np.nan])
     print(x) # [ True False]
22
23
24
     x = np.allclose([1.0, np.nan], [1.0, np.nan])
25
     print(x) # False
```

```
26
27  x = np.isclose([1.0, np.nan], [1.0, np.nan], equal_nan=True)
28  print(x) # [ True True]
29
30  x = np.allclose([1.0, np.nan], [1.0, np.nan], equal_nan=True)
31  print(x) # True
```

# 13 排序, 搜索和计数

### 13.1 排序

```
1. numpy.sort(a[, axis=-1, kind='quicksort', order=None]) Return a sorted copy of an array.

a. axis: 排序沿数组的(轴)方向,0表示按行,1表示按列,None表示展开来排序,默认为-1,表示沿最后的轴排序。
b. kind: 排序的算法,提供了快排'quicksort'、混排'mergesort'、堆排'heapsort',默认为'quicksort'。
c. order: 排序的字段名,可指定字段排序,默认为None。
```

```
1
     import numpy as np
3
     np.random.seed(20200612)
4
     x = np.random.rand(5, 5) * 10
     x = np.around(x, 2)
     print(x)
6
     # [[2.32 7.54 9.78 1.73 6.22]
7
     # [6.93 5.17 9.28 9.76 8.25]
9
    # [0.01 4.23 0.19 1.73 9.27]
    # [7.99 4.97 0.88 7.32 4.29]
10
11
     # [9.05 0.07 8.95 7.9 6.99]]
12
13
     y = np.sort(x)
14
     print(y)
    # [[1.73 2.32 6.22 7.54 9.78]
15
    # [5.17 6.93 8.25 9.28 9.76]
16
     # [0.01 0.19 1.73 4.23 9.27]
17
     # [0.88 4.29 4.97 7.32 7.99]
18
     # [0.07 6.99 7.9 8.95 9.05]]
19
20
21
    y = np.sort(x, axis=0)
22
     print(y)
     # [[0.01 0.07 0.19 1.73 4.29]
23
     # [2.32 4.23 0.88 1.73 6.22]
     # [6.93 4.97 8.95 7.32 6.99]
25
     # [7.99 5.17 9.28 7.9 8.25]
26
27
     # [9.05 7.54 9.78 9.76 9.27]]
28
29
     y = np.sort(x, axis=1)
    print(y)
     # [[1.73 2.32 6.22 7.54 9.78]
31
    # [5.17 6.93 8.25 9.28 9.76]
32
    # [0.01 0.19 1.73 4.23 9.27]
33
34
     # [0.88 4.29 4.97 7.32 7.99]
```

```
35 # [0.07 6.99 7.9 8.95 9.05]]
```

#### 【例】

```
1
     import numpy as np
2
     dt = np.dtype([('name', 'S10'), ('age', np.int)])
3
     a = np.array([("Mike", 21), ("Nancy", 25), ("Bob", 17), ("Jane", 27)], dtype=dt)
4
5
     b = np.sort(a, order='name')
6
     print(b)
7
     # [(b'Bob', 17) (b'Jane', 27) (b'Mike', 21) (b'Nancy', 25)]
8
9
     b = np.sort(a, order='age')
10
     print(b)
     # [(b'Bob', 17) (b'Mike', 21) (b'Nancy', 25) (b'Jane', 27)]
11
```

如果排序后,想用元素的索引位置替代排序后的实际结果,该怎么办呢?

1. numpy.argsort(a[, axis=-1, kind='quicksort', order=None]) Returns the indices that would sort an array.

【例】对数组沿给定轴执行间接排序,并使用指定排序类型返回数据的索引数组。这个索引数组用于构造排序后的数组。

```
1
     import numpy as np
2
     np.random.seed(20200612)
3
     x = np.random.randint(0, 10, 10)
4
5
     print(x)
     # [6 1 8 5 5 4 1 2 9 1]
6
7
8
     y = np.argsort(x)
9
     print(y)
     # [1 6 9 7 5 3 4 0 2 8]
10
11
12
     print(x[y])
     # [1 1 1 2 4 5 5 6 8 9]
13
14
15
     y = np.argsort(-x)
16
     print(y)
17
     # [8 2 0 3 4 5 7 1 6 9]
18
19
     print(x[y])
20
     # [9 8 6 5 5 4 2 1 1 1]
```

```
1
     import numpy as np
2
3
     np.random.seed(20200612)
    x = np.random.rand(5, 5) * 10
4
5
     x = np.around(x, 2)
6
     print(x)
7
     # [[2.32 7.54 9.78 1.73 6.22]
     # [6.93 5.17 9.28 9.76 8.25]
8
9
     # [0.01 4.23 0.19 1.73 9.27]
10
    # [7.99 4.97 0.88 7.32 4.29]
```

```
11
     # [9.05 0.07 8.95 7.9 6.99]]
12
13
     y = np.argsort(x)
14
     print(y)
     # [[3 0 4 1 2]
15
     # [1 0 4 2 3]
16
17
     # [0 2 3 1 4]
     # [2 4 1 3 0]
18
     # [1 4 3 2 0]]
19
20
21
     y = np.argsort(x, axis=0)
22
     print(y)
     # [[2 4 2 0 3]
23
24
     # [0 2 3 2 0]
25
     # [1 3 4 3 4]
     # [3 1 1 4 1]
26
27
     # [40012]]
28
     y = np.argsort(x, axis=1)
29
30
     print(y)
     # [[3 0 4 1 2]
31
     # [10423]
32
     # [0 2 3 1 4]
33
     # [2 4 1 3 0]
34
     # [1 4 3 2 0]]
35
36
37
     y = np.array([np.take(x[i], np.argsort(x[i])) for i in range(5)])
     #numpy.take(a, indices, axis=None, out=None, mode='raise')沿轴从数组中获取元素。
38
39
     print(y)
40
     # [[1.73 2.32 6.22 7.54 9.78]
41
     # [5.17 6.93 8.25 9.28 9.76]
    # [0.01 0.19 1.73 4.23 9.27]
42
43
     # [0.88 4.29 4.97 7.32 7.99]
    # [0.07 6.99 7.9 8.95 9.05]]
```

如何将数据按照某一指标进行排序呢?

- 1. numpy.lexsort(keys[, axis=-1]) Perform an indirect stable sort using a sequence of keys. (使用键序列执行间接稳定排序。)
- 2. 给定多个可以在电子表格中解释为列的排序键,lexsort返回一个整数索引数组,该数组描述了按多个列排序的顺序。序列中的最后一个键用于主排序顺序,倒数第二个键用于辅助排序顺序,依此类推。keys参数必须是可以转换为相同形状的数组的对象序列。如果为keys参数提供了2D数组,则将其行解释为排序键,并根据最后一行,倒数第二行等进行排序。
- 【例】按照第一列的升序或者降序对整体数据进行排序。

```
import numpy as np

np.random.seed(20200612)

x = np.random.rand(5, 5) * 10

x = np.around(x, 2)

print(x)

# [[2.32 7.54 9.78 1.73 6.22]

# [6.93 5.17 9.28 9.76 8.25]
```

```
9
     # [0.01 4.23 0.19 1.73 9.27]
     # [7.99 4.97 0.88 7.32 4.29]
10
     # [9.05 0.07 8.95 7.9 6.99]]
11
12
13
     index = np.lexsort([x[:, 0]])
     print(index)
14
     # [2 0 1 3 4]
15
16
17
     y = x[index]
18
     print(y)
     # [[0.01 4.23 0.19 1.73 9.27]
19
     # [2.32 7.54 9.78 1.73 6.22]
20
     # [6.93 5.17 9.28 9.76 8.25]
21
22
     # [7.99 4.97 0.88 7.32 4.29]
23
     # [9.05 0.07 8.95 7.9 6.99]]
24
25
     index = np.lexsort([-1 * x[:, 0]])
26
     print(index)
     # [4 3 1 0 2]
27
28
29
     y = x[index]
30
     print(y)
     # [[9.05 0.07 8.95 7.9 6.99]
31
32
     # [7.99 4.97 0.88 7.32 4.29]
     # [6.93 5.17 9.28 9.76 8.25]
33
34
     # [2.32 7.54 9.78 1.73 6.22]
35
     # [0.01 4.23 0.19 1.73 9.27]]
```

```
1
     import numpy as np
2
3
     x = np.array([1, 5, 1, 4, 3, 4, 4])
4
     y = np.array([9, 4, 0, 4, 0, 2, 1])
5
     a = np.lexsort([x])
6
     b = np.lexsort([y])
     print(a)
8
     # [0 2 4 3 5 6 1]
     print(x[a])
9
     # [1 1 3 4 4 4 5]
10
11
12
     print(b)
     # [2 4 6 5 1 3 0]
13
14
     print(y[b])
15
     # [0 0 1 2 4 4 9]
16
17
     z = np.lexsort([y, x])
18
     print(z)
     # [2 0 4 6 5 3 1]
19
20
     print(x[z])
21
     # [1 1 3 4 4 4 5]
22
23
     z = np.lexsort([x, y])
```

```
24 print(z)
25 # [2 4 6 5 3 1 0]
26 print(y[z])
27 # [0 0 1 2 4 4 9]
```

1. numpy.partition(a, kth, axis=-1, kind='introselect', order=None) Return a partitioned copy of an array.

Creates a copy of the array with its elements rearranged in such a way that the value of the element in k-th position is in the position it would be in a sorted array. All elements smaller than the k-th element are moved before this element and all equal or greater are moved behind it. The ordering of the elements in the two partitions is undefined.

【例】以索引是 kth 的元素为基准,将元素分成两部分,即大于该元素的放在其后面,小于该元素的放在其前面,这里有点类似于快排。

```
1
     import numpy as np
 2
3
     np.random.seed(100)
4
     x = np.random.randint(1, 30, [8, 3])
     print(x)
5
     # [[ 9 25 4]
6
7
     # [ 8 24 16]
8
     # [17 11 21]
     # [ 3 22 3]
9
10
     # [ 3 15 3]
11
     # [18 17 25]
     # [16 5 12]
12
     # [29 27 17]]
13
14
     y = np.sort(x, axis=0)
15
     print(y)
16
17
     # [[ 3 5 3]
     # [ 3 11 3]
18
     # [ 8 15 4]
19
20
     # [ 9 17 12]
21
     # [16 22 16]
     # [17 24 17]
22
23
     # [18 25 21]
     # [29 27 25]]
24
25
     z = np.partition(x, kth=2, axis=0)
26
27
     print(z)
     # [[ 3 5 3]
28
     # [ 3 11 3]
29
30
     # [ 8 15 4]
31
     # [ 9 22 21]
     # [17 24 16]
32
33
     # [18 17 25]
     # [16 25 12]
34
35
     # [29 27 17]]
```

#### 【例】选取每一列第三小的数

```
import numpy as np

np.random.seed(100)
```

```
4
     x = np.random.randint(1, 30, [8, 3])
5
     print(x)
     # [[ 9 25 4]
6
7
     # [ 8 24 16]
     # [17 11 21]
8
9
     # [ 3 22 3]
     # [ 3 15 3]
10
     # [18 17 25]
11
    # [16 5 12]
12
     # [29 27 17]]
13
     z = np.partition(x, kth=2, axis=0)
14
15
     print(z[2])
     # [ 8 15 4]
16
```

#### 【例】选取每一列第三大的数据

```
1
     import numpy as np
2
3
     np.random.seed(100)
     x = np.random.randint(1, 30, [8, 3])
4
5
     print(x)
     # [[ 9 25 4]
6
     # [ 8 24 16]
7
     # [17 11 21]
     # [ 3 22 3]
9
     # [ 3 15 3]
10
11
     # [18 17 25]
    # [16 5 12]
12
     # [29 27 17]]
13
14
     z = np.partition(x, kth=-3, axis=0)
15
     print(z[-3])
     # [17 24 17]
16
```

#### 1. numpy.argpartition(a, kth, axis=-1, kind='introselect', order=None)

Perform an indirect partition along the given axis using the algorithm specified by the kind keyword. It returns an array of indices of the same shape as a that index data along the given axis in partitioned order.

```
import numpy as np
3
     np.random.seed(100)
4
     x = np.random.randint(1, 30, [8, 3])
5
     print(x)
     # [[ 9 25 4]
6
    # [ 8 24 16]
7
8
     # [17 11 21]
     # [ 3 22 3]
9
    # [ 3 15 3]
10
     # [18 17 25]
11
     # [16 5 12]
12
     # [29 27 17]]
13
14
```

```
15
    y = np.argsort(x, axis=0)
16
    print(y)
17
    # [[3 6 3]
    # [4 2 4]
    # [1 4 0]
19
    # [0 5 6]
20
    # [6 3 1]
21
22
    # [2 1 7]
    # [5 0 2]
23
    # [7 7 5]]
24
25
    z = np.argpartition(x, kth=2, axis=0)
26
27
    print(z)
28
    # [[3 6 3]
29
    # [4 2 4]
    # [1 4 0]
30
31
    # [0 3 2]
32
    # [2 1 1]
    # [5 5 5]
33
    # [6 0 6]
34
35
    # [7 7 7]]
```

#### 【例】选取每一列第三小的数的索引

```
1
     import numpy as np
 2
 3
     np.random.seed(100)
 4
     x = np.random.randint(1, 30, [8, 3])
 5
     print(x)
     # [[ 9 25 4]
 7
     # [ 8 24 16]
     # [17 11 21]
 8
 9
     # [ 3 22 3]
10
     # [ 3 15 3]
     # [18 17 25]
11
     # [16 5 12]
12
     # [29 27 17]]
13
14
15
     z = np.argpartition(x, kth=2, axis=0)
16
     print(z[2])
17
     # [1 4 0]
```

#### 【例】选取每一列第三大的数的索引

```
1
     import numpy as np
2
3
     np.random.seed(100)
4
     x = np.random.randint(1, 30, [8, 3])
5
     print(x)
     # [[ 9 25 4]
     # [ 8 24 16]
7
    # [17 11 21]
8
9
     # [ 3 22 3]
10
     # [ 3 15 3]
```

```
11 # [18 17 25]
12 # [16 5 12]
13 # [29 27 17]]
14
15 z = np.argpartition(x, kth=-3, axis=0)
16 print(z[-3])
17 # [2 1 7]
```

## 13.2 搜索

1. numpy.argmax(a[, axis=None, out=None]) Returns the indices of the maximum values along an axis.

#### 【例】

```
import numpy as np
2
3
     np.random.seed(20200612)
     x = np.random.rand(5, 5) * 10
4
5
     x = np.around(x, 2)
     print(x)
6
7
     # [[2.32 7.54 9.78 1.73 6.22]
8
     # [6.93 5.17 9.28 9.76 8.25]
9
     # [0.01 4.23 0.19 1.73 9.27]
10
     # [7.99 4.97 0.88 7.32 4.29]
11
     # [9.05 0.07 8.95 7.9 6.99]]
12
13
     y = np.argmax(x)
14
     print(y) # 2
15
     y = np.argmax(x, axis=0)
16
17
     print(y)
18
     # [4 0 0 1 2]
19
20
     y = np.argmax(x, axis=1)
21
     print(y)
22
     # [2 3 4 0 0]
```

1. numpy.argmin(a[, axis=None, out=None]) Returns the indices of the minimum values along an axis.

```
import numpy as np

np.random.seed(20200612)

x = np.random.rand(5, 5) * 10

x = np.around(x, 2)

print(x)

# [[2.32 7.54 9.78 1.73 6.22]

# [6.93 5.17 9.28 9.76 8.25]
```

```
9
    # [0.01 4.23 0.19 1.73 9.27]
10
     # [7.99 4.97 0.88 7.32 4.29]
11
     # [9.05 0.07 8.95 7.9 6.99]]
12
13
     y = np.argmin(x)
     print(y) # 10
14
15
     y = np.argmin(x, axis=0)
16
17
     print(y)
     # [2 4 2 0 3]
18
19
20
     y = np.argmin(x, axis=1)
21
     print(y)
22
    # [3 1 0 2 1]
```

- 1. numppy.nonzero(a) Return the indices of the elements that are non-zero.
- , 其值为非零元素的下标在对应轴上的值。
  - 1. 只有 a 中非零元素才会有索引值, 那些零值元素没有索引值。
  - 2. 返回一个长度为 a.ndim 的元组(tuple),元组的每个元素都是一个整数数组(array)。
  - 3. 每一个array均是从一个维度上来描述其索引值。比如,如果 a 是一个二维数组,则tuple包含两个array,第一个array从行维度来描述索引值,第二个array从列维度来描述索引值。
  - 4. 该 np.transpose(np.nonzero(x)) 函数能够描述出每一个非零元素在不同维度的索引值。
  - 5. 通过 a [nonzero(a)] 得到所有 a 中的非零值。

#### 【例】一维数组

```
import numpy as np
1
3
     x = np.array([0, 2, 3])
4
    print(x) # [0 2 3]
     print(x.shape) # (3,)
5
     print(x.ndim) # 1
6
7
8
    y = np.nonzero(x)
9
     print(y) # (array([1, 2], dtype=int64),)
10
     print(np.array(y)) # [[1 2]]
11
     print(np.array(y).shape) # (1, 2)
12
     print(np.array(y).ndim) # 2
13
     print(np.transpose(y))
14
    # [[1]
    # [2]]
15
16
     print(x[np.nonzero(x)])
17
    #[2, 3]
```

#### 【例】二维数组

```
import numpy as np

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

print(x)

# [[3 0 0]

# [0 4 0]
```

```
# [5 6 0]]
7
8
     print(x.shape) # (3, 3)
9
     print(x.ndim) # 2
10
11
     y = np.nonzero(x)
12
     print(y)
13
     # (array([0, 1, 2, 2], dtype=int64), array([0, 1, 0, 1], dtype=int64))
14
     print(np.array(y))
     # [[0 1 2 2]
15
     # [0 1 0 1]]
16
17
     print(np.array(y).shape) # (2, 4)
18
     print(np.array(y).ndim) # 2
19
20
     y = x[np.nonzero(x)]
21
     print(y) # [3 4 5 6]
22
23
     y = np.transpose(np.nonzero(x))
24
     print(y)
    # [[0 0]
25
26
     # [1 1]
27
     # [20]
28
    # [2 1]]
```

#### 【例】三维数组

```
1
    import numpy as np
2
    x = np.array([[[0, 1], [1, 0]], [[0, 1], [1, 0]], [[0, 0], [1, 0]]])
3
4
    print(x)
5
    # [[[0 1]
    # [1 0]]
6
7
8
    # [[0 1]
9
    # [1 0]]
10
    # [[0 0]
11
12
    # [1 0]]]
13
    print(np.shape(x)) # (3, 2, 2)
14
    print(x.ndim) # 3
15
16
    y = np.nonzero(x)
17
    print(np.array(y))
    # [[0 0 1 1 2]
18
19
    # [0 1 0 1 1]
    # [10100]]
20
21
    print(np.array(y).shape) # (3, 5)
22
    print(np.array(y).ndim) # 2
23
    print(y)
    24
25
    print(x[np.nonzero(x)])
26
   #[1 1 1 1 1]
```

【例】 nonzero() 将布尔数组转换成整数数组进行操作。

```
1
     import numpy as np
2
3
     x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
4
     # [[1 2 3]
5
     # [4 5 6]
6
     # [7 8 9]]
7
8
9
     y = x > 3
     print(y)
10
     # [[False False False]
11
12
     # [ True True True]
     # [ True True True]]
13
14
15
     y = np.nonzero(x > 3)
     print(y)
16
17
     # (array([1, 1, 1, 2, 2, 2], dtype=int64), array([0, 1, 2, 0, 1, 2], dtype=int64))
18
     y = x[np.nonzero(x > 3)]
19
20
     print(y)
21
     # [4 5 6 7 8 9]
22
     y = x[x > 3]
23
24
     print(y)
25
     # [4 5 6 7 8 9]
```

1. numpy.where(condition, [x=None, y=None]) Return elements chosen from x or y depending on condition.

【例】满足条件 condition,输出 x,不满足输出 y。

```
1
     import numpy as np
 2
 3
    x = np.arange(10)
 4
     print(x)
     # [0 1 2 3 4 5 6 7 8 9]
 5
 6
 7
    y = np.where(x < 5, x, 10 * x)
 8
     print(y)
     # [ 0 1 2 3 4 50 60 70 80 90]
 9
10
11
     x = np.array([[0, 1, 2],
                  [0, 2, 4],
12
13
                  [0, 3, 6]])
     y = np.where(x < 4, x, -1)
14
15
     print(y)
16
    #[[0 1 2]
17
    # [02-1]
    # [ 0 3 -1]]
18
```

【例】只有 condition ,没有 x 和 y ,则输出满足条件 (即非0) 元素的坐标 (等价于 numpy.nonzero )。这里的坐标以tuple的形式给出,通常原数组有多少维,输出的tuple中就包含几个数组,分别对应符合条件元素的各维坐标。

```
1 import numpy as np
2
```

```
3
     x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
4
     y = np.where(x > 5)
5
     print(y)
6
     # (array([5, 6, 7], dtype=int64),)
7
     print(x[y])
     # [6 7 8]
8
9
     y = np.nonzero(x > 5)
10
11
     print(y)
12
     # (array([5, 6, 7], dtype=int64),)
13
     print(x[y])
14
     # [6 7 8]
15
16
     x = np.array([[11, 12, 13, 14, 15],
17
                   [16, 17, 18, 19, 20],
                   [21, 22, 23, 24, 25],
18
19
                   [26, 27, 28, 29, 30],
20
                   [31, 32, 33, 34, 35]])
     y = np.where(x > 25)
21
22
     print(y)
23
     # (array([3, 3, 3, 3, 3, 4, 4, 4, 4, 4], dtype=int64), array([0, 1, 2, 3, 4, 0, 1, 2, 3, 4], dtype=int64))
24
25
     print(x[y])
     # [26 27 28 29 30 31 32 33 34 35]
26
27
     y = np.nonzero(x > 25)
28
29
     # (array([3, 3, 3, 3, 3, 4, 4, 4, 4, 4], dtype=int64), array([0, 1, 2, 3, 4, 0, 1, 2, 3, 4], dtype=int64))
30
31
     print(x[y])
32
     # [26 27 28 29 30 31 32 33 34 35]
```

1. numpy.searchsorted(a, v[, side='left', sorter=None]) Find indices where elements should be inserted to maintain order.

a. a: 一维输入数组。当 sorter 参数为 None 的时候, a 必须为升序数组; 否则, sorter 不能为空,存放 a 中元素的 index ,用于 反映 a 数组的升序排列方式。

b. v: 插入 a 数组的值,可以为单个元素, list 或者 ndarray。

c. side:查询方向,当为 left 时,将返回第一个符合条件的元素下标;当为 right 时,将返回最后一个符合条件的元素下标。

d. sorter: 一维数组存放 a 数组元素的 index, index 对应元素为升序。

```
1
     import numpy as np
2
3
     x = np.array([0, 1, 5, 9, 11, 18, 26, 33])
4
     y = np.searchsorted(x, 15)
5
     print(y) # 5
6
7
     y = np.searchsorted(x, 15, side='right')
8
     print(y) # 5
9
10
     y = np.searchsorted(x, -1)
```

```
11
     print(y) # 0
12
     y = np.searchsorted(x, -1, side='right')
13
14
     print(y) # 0
15
16
     y = np.searchsorted(x, 35)
17
     print(y) # 8
18
     y = np.searchsorted(x, 35, side='right')
19
20
     print(y) # 8
21
22
     y = np.searchsorted(x, 11)
23
     print(y) # 4
24
25
     y = np.searchsorted(x, 11, side='right')
     print(y) # 5
26
27
28
     y = np.searchsorted(x, 0)
     print(y) # 0
29
30
31
     y = np.searchsorted(x, 0, side='right')
     print(y) # 1
32
33
34
     y = np.searchsorted(x, 33)
     print(y) # 7
35
36
37
     y = np.searchsorted(x, 33, side='right')
38
     print(y) # 8
```

#### 【例】

```
import numpy as np

x = np.array([0, 1, 5, 9, 11, 18, 26, 33])

y = np.searchsorted(x, [-1, 0, 11, 15, 33, 35])

print(y) # [0 0 4 5 7 8]

y = np.searchsorted(x, [-1, 0, 11, 15, 33, 35], side='right')

print(y) # [0 1 5 5 8 8]
```

```
1
     import numpy as np
2
3
     x = np.array([0, 1, 5, 9, 11, 18, 26, 33])
     np.random.shuffle(x)
4
5
     print(x) # [33 1 9 18 11 26 0 5]
6
7
     x_{sort} = np.argsort(x)
8
     print(x_sort) # [6 1 7 2 4 3 5 0]
9
     y = np.searchsorted(x, [-1, 0, 11, 15, 33, 35], sorter=x_sort)
10
     print(y) # [0 0 4 5 7 8]
11
12
```

```
13  y = np.searchsorted(x, [-1, 0, 11, 15, 33, 35], side='right', sorter=x_sort)
14  print(y) # [0 1 5 5 8 8]
```

# 13.3 计数

1. numpy.count\_nonzero(a, axis=None) Counts the number of non-zero values in the array a.

【例】返回数组中的非0元素个数。

```
1
     import numpy as np
 2
    x = np.count_nonzero(np.eye(4))
 3
 4
     print(x) # 4
 5
    x = np.count_nonzero([[0, 1, 7, 0, 0], [3, 0, 0, 2, 19]])
 6
 7
     print(x) # 5
 8
 9
     x = np.count_nonzero([[0, 1, 7, 0, 0], [3, 0, 0, 2, 19]], axis=0)
10
     print(x) # [1 1 1 1]
11
12
     x = np.count\_nonzero([[0, 1, 7, 0, 0], [3, 0, 0, 2, 19]], axis=1)
13
     print(x) # [2 3]
```

#### 参考文献

- 1. https://blog.csdn.net/u013698770/article/details/54632047
- 2. https://www.cnblogs.com/massquantity/p/8908859.html

# 14 集合操作

## 14.1 构造集合

- numpy.unique(ar, return\_index=False, return\_inverse=False, return\_counts=False, axis=None)
   Find the unique elements of an array.
  - a. return\_index=True 表示返回新列表元素在旧列表中的位置。
  - b. return\_inverse=True 表示返回旧列表元素在新列表中的位置。
  - c. return\_counts=True 表示返回新列表元素在旧列表中出现的次数。

#### 【例】找出数组中的唯一值并返回已排序的结果。

```
1
     import numpy as np
3
     x = np.unique([1, 1, 3, 2, 3, 3])
4
     print(x) # [1 2 3]
    x = sorted(set([1, 1, 3, 2, 3, 3]))
6
7
     print(x) # [1, 2, 3]
8
9
     x = np.array([[1, 1], [2, 3]])
     u = np.unique(x)
10
     print(u) # [1 2 3]
11
12
13
     x = np.array([[1, 0, 0], [1, 0, 0], [2, 3, 4]])
14
     y = np.unique(x, axis=0)
15
     print(y)
16
     # [[1 0 0]
     # [2 3 4]]
17
18
     x = np.array(['a', 'b', 'b', 'c', 'a'])
19
20
     u, index = np.unique(x, return_index=True)
     print(u) # ['a' 'b' 'c']
21
22
     print(index) # [0 1 3]
     print(x[index]) # ['a' 'b' 'c']
23
24
25
     x = np.array([1, 2, 6, 4, 2, 3, 2])
     u, index = np.unique(x, return_inverse=True)
26
     print(u) # [1 2 3 4 6]
27
     print(index) # [0 1 4 3 1 2 1]
28
     print(u[index]) # [1 2 6 4 2 3 2]
29
30
31
     u, count = np.unique(x, return_counts=True)
32
     print(u) # [1 2 3 4 6]
```

## 14.2 布尔运算

1. numpy.in1d(ar1, ar2, assume\_unique=False, invert=False) Test whether each element of a 1-D array is also present in a second

Returns a boolean array the same length as ar1 that is True where an element of ar1 is in ar2 and False otherwise.

【例】前面的数组是否包含于后面的数组,返回布尔值。返回的值是针对第一个参数的数组的,所以维数和第一个参数一致,布尔值与数组的 元素位置也一一对应。

```
import numpy as np
2
3
    test = np.array([0, 1, 2, 5, 0])
4
    states = [0, 2]
     mask = np.in1d(test, states)
5
     print(mask) # [ True False True False True]
6
7
     print(test[mask]) # [0 2 0]
8
9
     mask = np.in1d(test, states, invert=True)
10
     print(mask) # [False True False True False]
11
     print(test[mask]) # [1 5]
```

### 14.2.1 求两个集合的交集:

1. numpy.intersect1d(ar1, ar2, assume\_unique=False, return\_indices=False) Find the intersection of two arrays.

Return the sorted, unique values that are in both of the input arrays.

【例】求两个数组的唯一化+求交集+排序函数。

```
import numpy as np
2
     from functools import reduce
3
4
     x = np.intersect1d([1, 3, 4, 3], [3, 1, 2, 1])
     print(x) # [1 3]
5
6
7
     x = np.array([1, 1, 2, 3, 4])
8
     y = np.array([2, 1, 4, 6])
    xy, x_ind, y_ind = np.intersect1d(x, y, return_indices=True)
9
     print(x_ind) # [0 2 4]
10
11
     print(y_ind) # [1 0 2]
12
     print(xy) # [1 2 4]
     print(x[x_ind]) # [1 2 4]
13
14
     print(y[y_ind]) # [1 2 4]
15
```

```
16  x = reduce(np.intersect1d, ([1, 3, 4, 3], [3, 1, 2, 1], [6, 3, 4, 2]))

17  print(x) # [3]
```

### 14.2.2 求两个集合的并集:

1. numpy.union1d(ar1, ar2) Find the union of two arrays.

Return the unique, sorted array of values that are in either of the two input arrays.

【例】计算两个集合的并集,唯一化并排序。

```
import numpy as np
2
    from functools import reduce
3
4
    x = np.union1d([-1, 0, 1], [-2, 0, 2])
5
    print(x) # [-2 -1 0 1 2]
    x = reduce(np.union1d, ([1, 3, 4, 3], [3, 1, 2, 1], [6, 3, 4, 2]))
6
    print(x) # [1 2 3 4 6]
8
    functools.reduce(function, iterable[, initializer])
9
    将两个参数的 function 从左至右积累地应用到 iterable 的条目,以便将该可迭代对象缩减为单一的值。 例如,reduce(lambda x, y:
10
    x+y, [1, 2, 3, 4, 5]) 是计算((((1+2)+3)+4)+5)的值。 左边的参数 x 是积累值而右边的参数 y 则是来自 iterable 的更新值。 如
    果存在可选项 initializer, 它会被放在参与计算的可迭代对象的条目之前, 并在可迭代对象为空时作为默认值。 如果没有给出 initializer
    并且 iterable 仅包含一个条目,则将返回第一项。
11
12
    大致相当于:
    def reduce(function, iterable, initializer=None):
13
14
        it = iter(iterable)
        if initializer is None:
15
           value = next(it)
16
17
        else:
           value = initializer
18
        for element in it:
19
20
           value = function(value, element)
        return value
21
22
```

## 14.2.3 求两个集合的差集:

1. numpy.setdiff1d(ar1, ar2, assume\_unique=False) Find the set difference of two arrays.

Return the unique values in ar1 that are not in ar2.

【例】集合的差,即元素存在于第一个函数不存在于第二个函数中。

```
import numpy as np

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

x = np.setdiff1d(a, b)
print(x) # [1 2]
```

# 14.2.4 求两个集合的异或:

1. setxor1d(ar1, ar2, assume\_unique=False) Find the set exclusive-or of two arrays.

【例】集合的对称差,即两个集合的交集的补集。简言之,就是两个数组中各自独自拥有的元素的集合。

```
import numpy as np

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

x = np.setxor1d(a, b)
print(x) # [1 2 5 6]
```

#### 参考文献

1. https://www.jianshu.com/p/3bfe21aa1adb