Datawhale

Numpy教程

Datawhale 出品

1 常量

1.1 numpy.nan

1. 表示空值。

```
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

x = np.array([1, 1, 8, np.nan, 10])
print(x)
# [ 1.  1.  8. nan 10.]

y = np.isnan(x)
print(y)
# [False False False  True False]

z = np.count_nonzero(y)
print(z)  # 1
```

1.2 numpy.inf

1. 表示正无穷大。

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

【例】

```

```

1.3 numpy.pi

1. 表示圆周率

```
pi = 3.14159265358979323846264433...
```

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。

类型	备注	说明
bool_ = bool8	8位	布尔类型
int8 = byte	8位	整型
int16 = short	16位	整型
int32 = intc	32位	整型
int_ = int64 = long = int0 = intp	64位	整型
uint8 = ubyte	8位	无符号整型
uint16 = ushort	16位	无符号整型
uint32 = uintc	32位	无符号整型
uint64 = uintp = uint0 = uint	64位	无符号整型
float16 = half	16位	浮点型
float32 = single	32位	浮点型
float_ = float64 = double	64位	浮点型
str_ = unicode_ = str0 = unicode		Unicode 字符串
datetime64		日期时间类型
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。

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

每个内建类型都有一个唯一定义它的字符代码，如下：

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

【例】

```
import numpy as np

a = np.dtype('b1')
print(a.type)  # <class 'numpy.bool_'>
print(a.itemsize)  # 1

a = np.dtype('i1')
print(a.type)  # <class 'numpy.int8'>
print(a.itemsize)  # 1
a = np.dtype('i2')
print(a.type)  # <class 'numpy.int16'>
print(a.itemsize)  # 2
a = np.dtype('i4')
print(a.type)  # <class 'numpy.int32'>
print(a.itemsize)  # 4
a = np.dtype('i8')
print(a.type)  # <class 'numpy.int64'>
print(a.itemsize)  # 8

a = np.dtype('u1')
print(a.type)  # <class 'numpy.uint8'>
print(a.itemsize)  # 1
a = np.dtype('u2')
print(a.type)  # <class 'numpy.uint16'>
print(a.itemsize)  # 2
a = np.dtype('u4')
print(a.type)  # <class 'numpy.uint32'>
print(a.itemsize)  # 4
a = np.dtype('u8')
print(a.type)  # <class 'numpy.uint64'>
print(a.itemsize)  # 8

```

```
a = np.dtype('f2')
print(a.type)  # <class 'numpy.float16'>
print(a.itemsize)  # 2
a = np.dtype('f4')
print(a.type)  # <class 'numpy.float32'>
print(a.itemsize)  # 4
a = np.dtype('f8')
print(a.type)  # <class 'numpy.float64'>
print(a.itemsize)  # 8

a = np.dtype('S')
print(a.type)  # <class 'numpy.bytes_'>
print(a.itemsize)  # 0
a = np.dtype('S3')
print(a.type)  # <class 'numpy.bytes_'>
print(a.itemsize)  # 3

a = np.dtype('U3')
print(a.type)  # <class 'numpy.str_'>
print(a.itemsize)  # 12
```

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.

```
class iinfo(object):
    def __init__(self, int_type):
        pass
    def min(self):
        pass
    def max(self):
        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):
```

【例】

```
import numpy as np

ff16 = np.finfo(np.float16)
print(ff16.bits)  # 16
print(ff16.min)  # -65500.0
print(ff16.max)  # 65500.0
print(ff16.eps)  # 0.000977

ff32 = np.finfo(np.float32)
print(ff32.bits)  # 32
print(ff32.min)  # -3.4028235e+38
print(ff32.max)  # 3.4028235e+38
print(ff32.eps)  # 1.1920929e-07
```

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：

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

```
import numpy as np

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

a = np.datetime64('2020-03')
print(a, a.dtype)  # 2020-03 datetime64[M]

a = np.datetime64('2020-03-08 20:00:05')
print(a, a.dtype)  # 2020-03-08T20:00:05 datetime64[s]

a = np.datetime64('2020-03-08 20:00')
print(a, a.dtype)  # 2020-03-08T20:00 datetime64[m]

a = np.datetime64('2020-03-08 20')
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。

```
import numpy as np

a = np.arange('2020-08-01', '2020-08-10', dtype=np.datetime64)
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']
print(a.dtype)  # datetime64[D]

a = np.arange('2020-08-01 20:00', '2020-08-10', dtype=np.datetime64)
print(a)
# ['2020-08-01T20:00' '2020-08-01T20:01' '2020-08-01T20:02' ...
#  '2020-08-09T23:57' '2020-08-09T23:58' '2020-08-09T23:59']
print(a.dtype)  # datetime64[m]

a = np.arange('2020-05', '2020-12', dtype=np.datetime64)
print(a)
# ['2020-05' '2020-06' '2020-07' '2020-08' '2020-09' '2020-10' '2020-11']
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。

```python
import numpy as np

a = np.datetime64('2020-03-08') - np.datetime64('2020-03-07')
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')

print(a, a.dtype)  # 1 days timedelta64[D]
print(b, b.dtype)  # 956178240 minutes timedelta64[m]
print(c, c.dtype)  # 1 days timedelta64[D]

a = np.datetime64('2020-03') + np.timedelta64(20, 'D')
b = np.datetime64('2020-06-15 00:00') + np.timedelta64(12, 'h')
print(a, a.dtype)  # 2020-03-21 datetime64[D]
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）。

```python
import numpy as np

a = np.timedelta64(1, 'Y')
b = np.timedelta64(a, 'M')
print(a)  # 1 years
print(b)  # 12 months

c = np.timedelta64(1, 'h')
d = np.timedelta64(c, 'm')
print(c)  # 1 hours
print(d)  # 60 minutes

print(np.timedelta64(a, 'D'))
# TypeError: Cannot cast NumPy timedelta64 scalar from metadata [Y] to [D] according to the rule 'same_kind'

print(np.timedelta64(b, 'D'))
# TypeError: Cannot cast NumPy timedelta64 scalar from metadata [M] to [D] according to the rule 'same_kind'
```

【例】timedelta64 的运算。

```python
import numpy as np

a = np.timedelta64(1, 'Y')
b = np.timedelta64(6, 'M')
c = np.timedelta64(1, 'W')
d = np.timedelta64(1, 'D')
e = np.timedelta64(10, 'D')

print(a)  # 1 years
print(b)  # 6 months
print(a + b)  # 18 months
print(a - b)  # 6 months
print(2 * a)  # 2 years
print(a / b)  # 2.0
print(c / d)  # 7.0
print(c % e)  # 7 days
```

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

```
import numpy as np
import datetime

dt = datetime.datetime(year=2020, month=6, day=1, hour=20, minute=5, second=30)
dt64 = np.datetime64(dt, 's')
print(dt64, dt64.dtype)
# 2020-06-01T20:05:30 datetime64[s]

dt2 = dt64.astype(datetime.datetime)
print(dt2, type(dt2))
# 2020-06-01 20:05:30 <class 'datetime.datetime'>
```

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）

```
import numpy as np

# 2020-07-10 星期五
a = np.busday_offset('2020-07-10', offsets=1)
print(a)  # 2020-07-13

a = np.busday_offset('2020-07-11', offsets=1)
print(a)
# ValueError: Non-business day date in busday_offset

a = np.busday_offset('2020-07-11', offsets=0, roll='forward')
b = np.busday_offset('2020-07-11', offsets=0, roll='backward')
print(a)  # 2020-07-13
print(b)  # 2020-07-10

a = np.busday_offset('2020-07-11', offsets=1, roll='forward')
b = np.busday_offset('2020-07-11', offsets=1, roll='backward')
print(a)  # 2020-07-14
```

```
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。

```
import numpy as np

# 2020-07-10 星期五
a = np.is_busday('2020-07-10')
b = np.is_busday('2020-07-11')
print(a)  # True
print(b)  # False
```

【例】统计一个 `datetime64[D]` 数组中的工作日天数。

```
import numpy as np

# 2020-07-10 星期五
begindates = np.datetime64('2020-07-10')
enddates = np.datetime64('2020-07-20')
a = np.arange(begindates, enddates, dtype='datetime64')
b = np.count_nonzero(np.is_busday(a))
print(a)
# ['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']
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。

```
import numpy as np

# 2020-07-10 星期五
a = np.is_busday('2020-07-10', weekmask=[1, 1, 1, 1, 1, 0, 0])
b = np.is_busday('2020-07-10', weekmask=[1, 1, 1, 1, 0, 0, 1])
print(a)  # True
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. 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

4 数组的创建

导入 numpy。

```
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):
```

【例】

```
import numpy as np

# 创建一维数组
a = np.array([0, 1, 2, 3, 4])
b = np.array((0, 1, 2, 3, 4))
print(a, type(a))
# [0 1 2 3 4] <class 'numpy.ndarray'>
print(b, type(b))
# [0 1 2 3 4] <class 'numpy.ndarray'>

# 创建二维数组
c = np.array([[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]])
print(c, type(c))
# [[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]] <class 'numpy.ndarray'>

# 创建三维数组
d = np.array([[(1.5, 2, 3), (4, 5, 6)],
              [(3, 2, 1), (4, 5, 6)]])
print(d, type(d))
```

```
# [[[1.5 2.  3. ]
#   [4.  5.  6. ]]
#
#  [[3.  2.  1. ]
#   [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

```
import numpy as np

x = [[1, 1, 1], [1, 1, 1], [1, 1, 1]]
y = np.array(x)
z = np.asarray(x)
x[1][2] = 2
print(x,type(x))
# [[1, 1, 1], [1, 1, 2], [1, 1, 1]] <class 'list'>

print(y,type(y))
# [[1 1 1]
#  [1 1 1]
#  [1 1 1]] <class 'numpy.ndarray'>

print(z,type(z))
# [[1 1 1]
#  [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。)

```
import numpy as np

x = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
y = np.array(x)
z = np.asarray(x)
w = np.asarray(x, dtype=np.int)
x[1][2] = 2
print(x,type(x),x.dtype)
# [[1 1 1]
#  [1 1 2]
#  [1 1 1]] <class 'numpy.ndarray'> int32

print(y,type(y),y.dtype)
# [[1 1 1]
```

```
#  [1 1 1]
#  [1 1 1]] <class 'numpy.ndarray'> int32

print(z,type(z),z.dtype)
# [[1 1 1]
#  [1 1 2]
#  [1 1 1]] <class 'numpy.ndarray'> int32

print(w,type(w),w.dtype)
# [[1 1 1]
#  [1 1 2]
#  [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

```
import numpy as np

x = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
print(x, x.dtype)
# [[1 1 1]
#  [1 1 1]
#  [1 1 1]] int32
x.dtype = np.float

# 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。

```
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。

```
import numpy as np

def f(x, y):
    return 10 * x + y

x = np.fromfunction(f, (5, 4), dtype=int)
print(x)
# [[ 0  1  2  3]
#  [10 11 12 13]
#  [20 21 22 23]
#  [30 31 32 33]
#  [40 41 42 43]]

x = np.fromfunction(lambda i, j: i == j, (3, 3), dtype=int)
print(x)
# [[ True False False]
#  [False  True False]
```

```
#  [False False  True]]

x = np.fromfunction(lambda i, j: i + j, (3, 3), dtype=int)
print(x)
# [[0 1 2]
#  [1 2 3]
#  [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

x = np.zeros(5)
print(x)  # [0. 0. 0. 0. 0.]
x = np.zeros([2, 3])
print(x)
# [[0. 0. 0.]
#  [0. 0. 0.]]

x = np.array([[1, 2, 3], [4, 5, 6]])
y = np.zeros_like(x)
print(y)
# [[0 0 0]
#  [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

x = np.ones(5)
print(x)  # [1. 1. 1. 1. 1.]
x = np.ones([2, 3])
print(x)
# [[1. 1. 1.]
#  [1. 1. 1.]]

x = np.array([[1, 2, 3], [4, 5, 6]])
y = np.ones_like(x)
print(y)
# [[1 1 1]
#  [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):
```

【例】

```
import numpy as np

x = np.empty(5)
print(x)
# [1.95821574e-306 1.60219035e-306 1.37961506e-306
#  9.34609790e-307 1.24610383e-306]

x = np.empty((3, 2))
print(x)
# [[1.60220393e-306 9.34587382e-307]
#  [8.45599367e-307 7.56598449e-307]
#  [1.33509389e-306 3.59412896e-317]]

x = np.array([[1, 2, 3], [4, 5, 6]])
y = np.empty_like(x)
print(y)
# [[  7209029   6422625   6619244]
#  [      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

x = np.eye(4)
print(x)
# [[1. 0. 0. 0.]
#  [0. 1. 0. 0.]
#  [0. 0. 1. 0.]
#  [0. 0. 0. 1.]]

x = np.eye(2, 3)
print(x)
# [[1. 0. 0.]
#  [0. 1. 0.]]

x = np.identity(4)
print(x)
# [[1. 0. 0. 0.]
#  [0. 1. 0. 0.]
#  [0. 0. 1. 0.]
#  [0. 0. 0. 1.]]
```

4.2.5 （e）对角数组

1. `diag()` 函数：提取对角线或构造对角数组。

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

【例】

```
import numpy as np

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

v = [1, 3, 5, 7]
x = np.diag(v)
```

```
print(x)
# [[1 0 0 0]
#  [0 3 0 0]
#  [0 0 5 0]
#  [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】

```
import numpy as np

x = np.full((2,), 7)
print(x)
# [7 7]

x = np.full(2, 7)
print(x)
# [7 7]

x = np.full((2, 7), 7)
print(x)
# [[7 7 7 7 7 7 7]
#  [7 7 7 7 7 7 7]]

x = np.array([[1, 2, 3], [4, 5, 6]])
y = np.full_like(x, 7)
print(y)
# [[7 7 7]
#  [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。

```
def arange([start,] stop[, step,], dtype=None):
def linspace(start, stop, num=50, endpoint=True, retstep=False,
             dtype=None, axis=0):
def logspace(start, stop, num=50, endpoint=True, base=10.0,
             dtype=None, axis=0):
def rand(d0, d1, ..., dn):
```

【例】

```
import numpy as np

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

x = np.arange(3, 7, 2)
print(x)  # [3 5]

x = np.linspace(start=0, stop=2, num=9)
print(x)
# [0.   0.25 0.5  0.75 1.   1.25 1.5  1.75 2.  ]

x = np.logspace(0, 1, 5)
print(np.around(x, 2))
# [ 1.    1.78  3.16  5.62 10.  ]
                                  #np.around 返回四舍五入后的值，可指定精度。
                                 # around(a, decimals=0, out=None)
                                 # 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


x = np.linspace(start=0, stop=1, num=5)
x = [10 ** i for i in x]
print(np.around(x, 2))
# [ 1.    1.78  3.16  5.62 10.  ]

x = np.random.random(5)
print(x)
# [0.41768753 0.16315577 0.80167915 0.99690199 0.11812291]

x = np.random.random([2, 3])
print(x)
# [[0.41151858 0.93785153 0.57031309]
#  [0.13482333 0.20583516 0.45429181]]
```

4.4 4. 结构数组的创建

结构数组，首先需要定义结构，然后利用`np.array()`来创建数组，其参数`dtype`为定义的结构。

4.4.1 （a）利用字典来定义结构

【例】

```
import numpy as np

personType = np.dtype({
    'names': ['name', 'age', 'weight'],
    'formats': ['U30', 'i8', 'f8']})

a = np.array([('Liming', 24, 63.9), ('Mike', 15, 67.), ('Jan', 34, 45.8)],
             dtype=personType)
print(a, type(a))
# [('Liming', 24, 63.9) ('Mike', 15, 67. ) ('Jan', 34, 45.8)]
# <class 'numpy.ndarray'>
```

4.4.2 （b）利用包含多个元组的列表来定义结构

【例】

```
import numpy as np

personType = np.dtype([('name', 'U30'), ('age', 'i8'), ('weight', 'f8')])
a = np.array([('Liming', 24, 63.9), ('Mike', 15, 67.), ('Jan', 34, 45.8)],
             dtype=personType)
print(a, type(a))
# [('Liming', 24, 63.9) ('Mike', 15, 67. ) ('Jan', 34, 45.8)]
# <class 'numpy.ndarray'>

# 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：
print(a[0])
# ('Liming', 24, 63.9)

print(a[-2:])
# [('Mike', 15, 67. ) ('Jan', 34, 45.8)]

# 我们可以使用字段名作为下标获取对应的值
print(a['name'])
# ['Liming' 'Mike' 'Jan']
print(a['age'])
# [24 15 34]
print(a['weight'])
# [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】

```
import numpy as np

a = np.array([1, 2, 3, 4, 5])
print(a.shape)  # (5,)
print(a.dtype)  # int32
print(a.size)  # 5
print(a.ndim)  # 1
print(a.itemsize)  # 4

b = np.array([[1, 2, 3], [4, 5, 6.0]])
print(b.shape)  # (2, 3)
print(b.dtype)  # float64
print(b.size)  # 6
print(b.ndim)  # 2
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` 。

【例】

```python
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】

```
import numpy as np

x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = x
y[0] = -1
print(x)
# [-1  2  3  4  5  6  7  8]
print(y)
# [-1  2  3  4  5  6  7  8]

x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = x.copy()
y[0] = -1
print(x)
# [1 2 3 4 5 6 7 8]
print(y)
# [-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。

```
import numpy as np

x = np.array([[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]])
y = x
y[::2, :3:2] = -1
print(x)
# [[-1 12 -1 14 15]
#  [16 17 18 19 20]
#  [-1 22 -1 24 25]
#  [26 27 28 29 30]
#  [-1 32 -1 34 35]]
print(y)
```

```
# [[-1 12 -1 14 15]
#  [16 17 18 19 20]
#  [-1 22 -1 24 25]
#  [26 27 28 29 30]
#  [-1 32 -1 34 35]]

x = np.array([[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]])
y = x.copy()
y[::2, :3:2] = -1
print(x)
# [[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]]
print(y)
# [[-1 12 -1 14 15]
#  [16 17 18 19 20]
#  [-1 22 -1 24 25]
#  [26 27 28 29 30]
#  [-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

x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
print(x[2])  # 3

x = np.array([[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]])
print(x[2])  # [21 22 23 24 25]
print(x[2][1])  # 22
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

```python
import numpy as np

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

【例】对二维数组切片

```python
import numpy as np

x = np.array([[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]])
print(x[0:2])
# [[11 12 13 14 15]
#  [16 17 18 19 20]]

print(x[1:5:2])
# [[16 17 18 19 20]
#  [26 27 28 29 30]]

print(x[2:])
# [[21 22 23 24 25]
#  [26 27 28 29 30]
#  [31 32 33 34 35]]

print(x[:2])
# [[11 12 13 14 15]
#  [16 17 18 19 20]]

print(x[-2:])
# [[26 27 28 29 30]
#  [31 32 33 34 35]]

print(x[:-2])
# [[11 12 13 14 15]
#  [16 17 18 19 20]
#  [21 22 23 24 25]]

print(x[:])
# [[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]]
```

```
print(x[2, :])  # [21 22 23 24 25]
print(x[:, 2])  # [13 18 23 28 33]
print(x[0, 1:4])  # [12 13 14]
print(x[1:4, 0])  # [16 21 26]
print(x[1:3, 2:4])
# [[18 19]
#  [23 24]]

print(x[:, :])
# [[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]]

print(x[::2, ::2])
# [[11 13 15]
#  [21 23 25]
#  [31 33 35]]

print(x[::-1, :])
# [[31 32 33 34 35]
#  [26 27 28 29 30]
#  [21 22 23 24 25]
#  [16 17 18 19 20]
#  [11 12 13 14 15]]

print(x[:, ::-1])
# [[15 14 13 12 11]
#  [20 19 18 17 16]
#  [25 24 23 22 21]
#  [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。

【例】

```
import numpy as np

x = np.array([[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]])
print(x)
# [[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]]

```

```
x[0::2, 1::3] = 0
print(x)
# [[11  0 13 14  0]
#  [16 17 18 19 20]
#  [21  0 23 24  0]
#  [26 27 28 29 30]
#  [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】

```
import numpy as np

x = np.random.randint(1, 100, [2, 2, 3])
print(x)
# [[[ 5 64 75]
#   [57 27 31]]
#
#  [[68 85  3]
#   [93 26 25]]]

print(x[1, ...])
# [[68 85  3]
#  [93 26 25]]

print(x[..., 2])
# [[75 31]
#  [ 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。

```
import numpy as np

```

```
x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
r = [0, 1, 2]
print(x[r])
# [1 2 3]

r = [0, 1, -1]
print(x[r])
# [1 2 8]

x = np.array([[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]])

r = [0, 1, 2]
print(x[r])
# [[11 12 13 14 15]
#  [16 17 18 19 20]
#  [21 22 23 24 25]]

r = [0, 1, -1]
print(x[r])

# [[11 12 13 14 15]
#  [16 17 18 19 20]
#  [31 32 33 34 35]]

r = [0, 1, 2]
c = [2, 3, 4]
y = x[r, c]
print(y)
# [13 19 25]
```

【例】

```
import numpy as np

x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
r = np.array([[0, 1], [3, 4]])
print(x[r])
# [[1 2]
#  [4 5]]

x = np.array([[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]])

r = np.array([[0, 1], [3, 4]])
print(x[r])
# [[[11 12 13 14 15]
```

```
#   [16 17 18 19 20]]
#
#  [[26 27 28 29 30]
#   [31 32 33 34 35]]]

# 获取了 5X5 数组中的四个角的元素。
# 行索引是 [0,0] 和 [4,4]，而列索引是 [0,4] 和 [0,4]。
r = np.array([[0, 0], [4, 4]])
c = np.array([[0, 4], [0, 4]])
y = x[r, c]
print(y)
# [[11 15]
#  [31 35]]
```

【例】可以借助切片`:`与整数数组组合。

```
import numpy as np

x = np.array([[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]])

y = x[0:3, [1, 2, 2]]
print(y)
# [[12 13 13]
#  [17 18 18]
#  [22 23 23]]
```

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

【例】

```
import numpy as np

x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
r = [0, 1, 2]
print(np.take(x, r))
# [1 2 3]

r = [0, 1, -1]
print(np.take(x, r))
# [1 2 8]

x = np.array([[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]])

r = [0, 1, 2]
print(np.take(x, r, axis=0))
# [[11 12 13 14 15]
```

```
#  [16 17 18 19 20]
#  [21 22 23 24 25]]

r = [0, 1, -1]
print(np.take(x, r, axis=0))
# [[11 12 13 14 15]
#  [16 17 18 19 20]
#  [31 32 33 34 35]]

r = [0, 1, 2]
c = [2, 3, 4]
y = np.take(x, [r, c])
print(y)
# [[11 12 13]
#  [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。

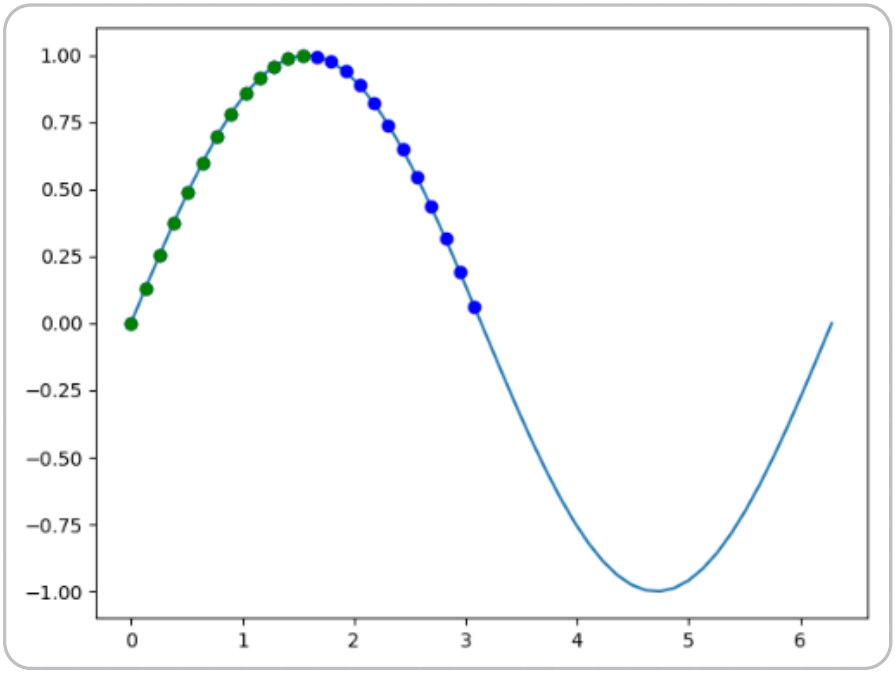
【例】

```
import numpy as np

x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = x > 5
print(y)
# [False False False False False  True  True  True]
print(x[x > 5])
# [6 7 8]

x = np.array([np.nan, 1, 2, np.nan, 3, 4, 5])
y = np.logical_not(np.isnan(x))
print(x[y])
# [1. 2. 3. 4. 5.]

x = np.array([[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]])
y = x > 25
print(y)
# [[False False False False False]
#  [False False False False False]
#  [False False False False False]
#  [ True  True  True  True  True]
#  [ True  True  True  True  True]]
print(x[x > 25])
```

```
# [26 27 28 29 30 31 32 33 34 35]
```

【例】

```
import numpy as np

import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 50)
y = np.sin(x)
print(len(x))  # 50
plt.plot(x, y)

mask = y >= 0
print(len(x[mask]))  # 25
print(mask)
'''
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True False False False False False False False False False False False
 False False False False False False False False False False False False
 False False]
'''
plt.plot(x[mask], y[mask], 'bo')

mask = np.logical_and(y >= 0, x <= np.pi / 2)
print(mask)
'''
[ True  True  True  True  True  True  True  True  True  True  True  True
  True False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False]
'''

plt.plot(x[mask], y[mask], 'go')
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0.5π 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.

【例】

```
import numpy as np

x = np.array([[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]])

y = np.apply_along_axis(np.sum, 0, x)
print(y)  # [105 110 115 120 125]
y = np.apply_along_axis(np.sum, 1, x)
print(y)  # [ 65  90 115 140 165]

y = np.apply_along_axis(np.mean, 0, x)
print(y)  # [21. 22. 23. 24. 25.]
y = np.apply_along_axis(np.mean, 1, x)
print(y)  # [13. 18. 23. 28. 33.]


def my_func(x):
    return (x[0] + x[-1]) * 0.5


y = np.apply_along_axis(my_func, 0, x)
print(y)  # [21. 22. 23. 24. 25.]
y = np.apply_along_axis(my_func, 1, x)
print(y)  # [13. 18. 23. 28. 33.]
```

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】

```
import numpy as np

x = np.array([[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]])
y = x.flat
print(y)
# <numpy.flatiter object at 0x0000020F9BA10C60>
for i in y:
    print(i, end=' ')
# 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

y[3] = 0
print(end='\n')
print(x)
# [[11 12 13  0 15]
#  [16 17 18 19 20]
#  [21 22 23 24 25]
#  [26 27 28 29 30]
#  [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。

```
import numpy as np

x = np.array([[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]])
y = x.flatten()
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
#  35]

y[3] = 0
print(x)
# [[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]]

x = np.array([[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]])

y = x.flatten(order='F')
print(y)
# [11 16 21 26 31 12 17 22 27 32 13 18 23 28 33 14 19 24 29 34 15 20 25 30
#  35]

y[3] = 0
print(x)
# [[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]]
```

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

【例】`ravel()`返回的是视图。

```
import numpy as np
```

```
x = np.array([[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]])
y = np.ravel(x)
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
#  35]

y[3] = 0
print(x)
# [[11 12 13  0 15]
#  [16 17 18 19 20]
#  [21 22 23 24 25]
#  [26 27 28 29 30]
#  [31 32 33 34 35]]
```

【例】order=F 就是拷贝

```
x = np.array([[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]])

y = np.ravel(x, order='F')
print(y)
# [11 16 21 26 31 12 17 22 27 32 13 18 23 28 33 14 19 24 29 34 15 20 25 30
#  35]

y[3] = 0
print(x)
# [[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]]
```

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。

```
import numpy as np

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

```
10
11  y = np.reshape(x, [3, -1])
12  print(y)
13  # [[ 0  1  2  3]
14  #  [ 4  5  6  7]
15  #  [ 8  9 10 11]]
16
17  y = np.reshape(x,[-1,3])
18  print(y)
19  # [[ 0  1  2]
20  #  [ 3  4  5]
21  #  [ 6  7  8]
22  #  [ 9 10 11]]
23
24  y[0, 1] = 10
25  print(x)
26  # [ 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
2
3   x = np.random.randint(12, size=[2, 2, 3])
4   print(x)
5   # [[[11  9  1]
6   #   [ 1 10  3]]
7   #
8   #  [[ 0  6  1]
9   #   [ 4 11  3]]]
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`.

【例】

```
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]
```

```
#  [1.8  9.52 2.78 5.87 4.14]]
y = x.T
print(y)
# [[6.74 3.54 5.03 7.5  1.8 ]
#  [8.46 3.49 7.22 7.31 9.52]
#  [6.74 8.62 1.6  5.69 2.78]
#  [5.45 1.94 8.7  9.67 5.87]
#  [1.25 9.92 0.43 7.65 4.14]]
y = np.transpose(x)
print(y)
# [[6.74 3.54 5.03 7.5  1.8 ]
#  [8.46 3.49 7.22 7.31 9.52]
#  [6.74 8.62 1.6  5.69 2.78]
#  [5.45 1.94 8.7  9.67 5.87]
#  [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。

```
import numpy as np

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

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

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

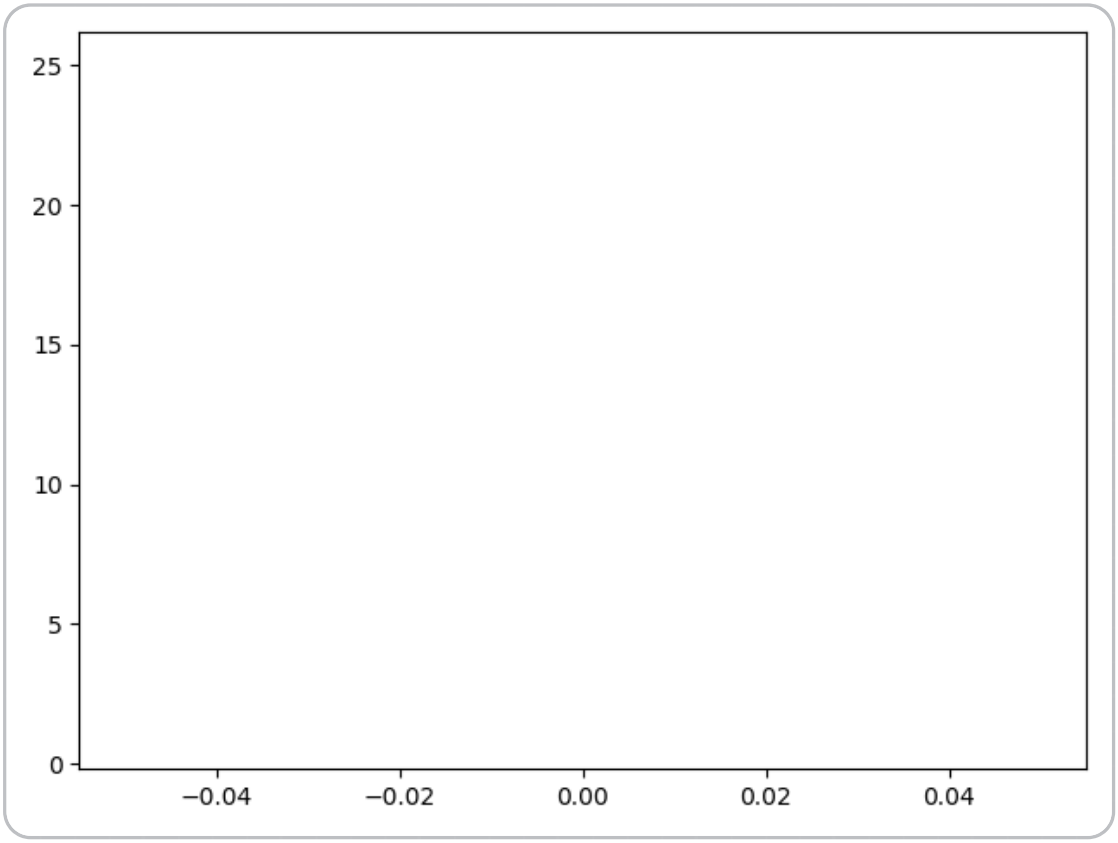
```
import numpy as np

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】

```
import numpy as np

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

y = np.squeeze(x)
print(y.shape)  # (3,)
print(y)  # [0 1 2]

y = np.squeeze(x, axis=0)
print(y.shape)  # (3, 1)
print(y)
# [[0]
#  [1]
#  [2]]

y = np.squeeze(x, axis=2)
print(y.shape)  # (1, 3)
print(y)  # [[0 1 2]]

y = np.squeeze(x, axis=1)
# ValueError: cannot select an axis to squeeze out which has size not equal to one
```

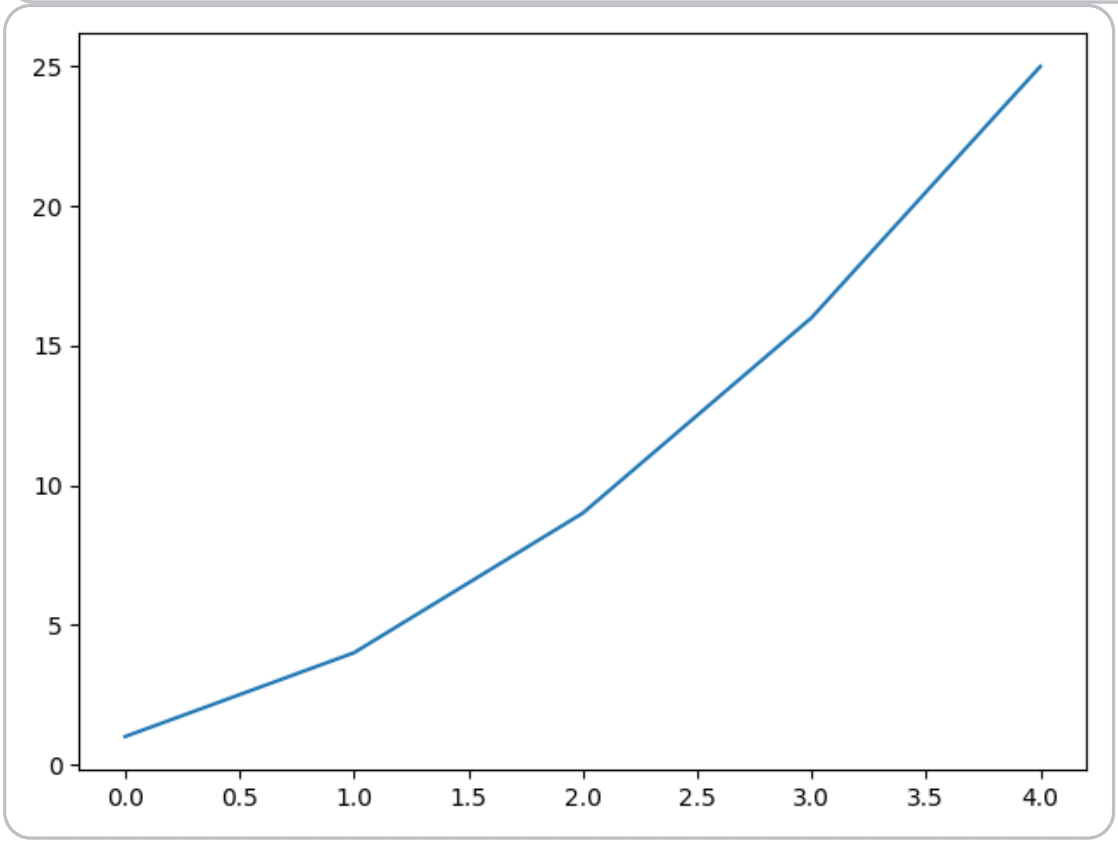
【例】

```
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()
```



【例】

```python
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

x = np.array([1, 2, 3])
y = np.array([7, 8, 9])
z = np.concatenate([x, y])
print(z)
# [1 2 3 7 8 9]

z = np.concatenate([x, y], axis=0)
print(z)
# [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

x = np.array([1, 2, 3]).reshape(1, 3)
y = np.array([7, 8, 9]).reshape(1, 3)
z = np.concatenate([x, y])
print(z)
# [[ 1  2  3]
#  [ 7  8  9]]
z = np.concatenate([x, y], axis=0)
print(z)
# [[ 1  2  3]
#  [ 7  8  9]]
z = np.concatenate([x, y], axis=1)
print(z)
# [[ 1  2  3  7  8  9]]
```

【例】x，y在原来的维度上进行拼接。

```
import numpy as np

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

```
#  [ 7  8  9]
#  [10 11 12]]
z = np.concatenate([x, y], axis=1)
print(z)
# [[ 1  2  3  7  8  9]
#  [ 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）。

```
import numpy as np

x = np.array([1, 2, 3])
y = np.array([7, 8, 9])
z = np.stack([x, y])
print(z.shape)  # (2, 3)
print(z)
# [[1 2 3]
#  [7 8 9]]

z = np.stack([x, y], axis=1)
print(z.shape)  # (3, 2)
print(z)
# [[1 7]
#  [2 8]
#  [3 9]]
```

【例】

```
import numpy as np

x = np.array([1, 2, 3]).reshape(1, 3)
y = np.array([7, 8, 9]).reshape(1, 3)
z = np.stack([x, y])
print(z.shape)  # (2, 1, 3)
print(z)
# [[[1 2 3]]
#
#  [[7 8 9]]]

z = np.stack([x, y], axis=1)
print(z.shape)  # (1, 2, 3)
print(z)
# [[[1 2 3]
#   [7 8 9]]]

z = np.stack([x, y], axis=2)
print(z.shape)  # (1, 3, 2)
print(z)
# [[[1 7]
#   [2 8]
#   [3 9]]]
```

【例】

```python
import numpy as np

x = np.array([[1, 2, 3], [4, 5, 6]])
y = np.array([[7, 8, 9], [10, 11, 12]])
z = np.stack([x, y])
print(z.shape)  # (2, 2, 3)
print(z)
# [[[ 1  2  3]
#   [ 4  5  6]]
#
#  [[ 7  8  9]
#   [10 11 12]]]

z = np.stack([x, y], axis=1)
print(z.shape)  # (2, 2, 3)
print(z)
# [[[ 1  2  3]
#   [ 7  8  9]]
#
#  [[ 4  5  6]
#   [10 11 12]]]

z = np.stack([x, y], axis=2)
print(z.shape)  # (2, 3, 2)
print(z)
# [[[ 1  7]
#   [ 2  8]
#   [ 3  9]]
#
#  [[ 4 10]
#   [ 5 11]
#   [ 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。

```python
import numpy as np

x = np.array([1, 2, 3])
y = np.array([7, 8, 9])
z = np.vstack((x, y))
print(z.shape)  # (2, 3)
print(z)
# [[1 2 3]
#  [7 8 9]]

z = np.stack([x, y])
print(z.shape)  # (2, 3)
print(z)
# [[1 2 3]
#  [7 8 9]]
```

```
16
17  z = np.hstack((x, y))
18  print(z.shape)  # (6,)
19  print(z)
20  # [1  2  3  7  8  9]
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)
4   y = np.array([7, 8, 9]).reshape(1, 3)
5   z = np.vstack((x, y))
6   print(z.shape)  # (2, 3)
7   print(z)
8   # [[1 2 3]
9   #  [7 8 9]]
10
11  z = np.concatenate((x, y), axis=0)
12  print(z.shape)  # (2, 3)
13  print(z)
14  # [[1 2 3]
15  #  [7 8 9]]
16
17  z = np.hstack((x, y))
18  print(z.shape)  # (1, 6)
19  print(z)
20  # [[ 1  2  3  7  8  9]]
21
22  z = np.concatenate((x, y), axis=1)
23  print(z.shape)  # (1, 6)
24  print(z)
25  # [[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], [4, 5, 6]])
4   y = np.array([[7, 8, 9], [10, 11, 12]])
5   z = np.vstack((x, y))
6   print(z.shape)  # (4, 3)
7   print(z)
8   # [[ 1  2  3]
9   #  [ 4  5  6]
10  #  [ 7  8  9]
11  #  [10 11 12]]
12
13  z = np.concatenate((x, y), axis=0)
```

```
print(z.shape)  # (4, 3)
print(z)
# [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]

z = np.hstack((x, y))
print(z.shape)  # (2, 6)
print(z)
# [[ 1  2  3  7  8  9]
#  [ 4  5  6 10 11 12]]

z = np.concatenate((x, y), axis=1)
print(z.shape)  # (2, 6)
print(z)
# [[ 1  2  3  7  8  9]
#  [ 4  5  6 10 11 12]]
```

`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],
              [16, 17, 18, 19],
              [21, 22, 23, 24]])
y = np.split(x, [1, 3])
print(y)
# [array([[11, 12, 13, 14]]), array([[16, 17, 18, 19],
#        [21, 22, 23, 24]]), array([], shape=(0, 4), dtype=int32)]

```

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

```
import numpy as np

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

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


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

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


y = np.vsplit(x, [1, 3])
print(y)
# [array([[11, 12, 13, 14]]), array([[16, 17, 18, 19],
#        [21, 22, 23, 24]]), array([], shape=(0, 4), dtype=int32)]
y = np.split(x, [1, 3], axis=0)
print(y)
# [array([[11, 12, 13, 14]]), array([[16, 17, 18, 19],
#        [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。

```
import numpy as np
```

```
x = np.array([[11, 12, 13, 14],
              [16, 17, 18, 19],
              [21, 22, 23, 24]])
y = np.hsplit(x, 2)
print(y)
# [array([[11, 12],
#        [16, 17],
#        [21, 22]]), array([[13, 14],
#        [18, 19],
#        [23, 24]])]

y = np.split(x, 2, axis=1)
print(y)
# [array([[11, 12],
#        [16, 17],
#        [21, 22]]), array([[13, 14],
#        [18, 19],
#        [23, 24]])]

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

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

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

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

```
#        [22, 23]]), array([[14],
#        [19],
#        [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。

```
import numpy as np

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

y = np.tile(x, (1, 3))
print(y)
# [[1 2 1 2 1 2]
#  [3 4 3 4 3 4]]

y = np.tile(x, (3, 1))
print(y)
# [[1 2]
#  [3 4]
#  [1 2]
#  [3 4]
#  [1 2]
#  [3 4]]

y = np.tile(x, (3, 3))
print(y)
# [[1 2 1 2 1 2]
#  [3 4 3 4 3 4]
#  [1 2 1 2 1 2]
#  [3 4 3 4 3 4]
#  [1 2 1 2 1 2]
#  [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

x = np.repeat(3, 4)
print(x)  # [3 3 3 3]

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

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

y = np.repeat(x, 2, axis=1)
print(y)
# [[1 1 2 2]
#  [3 3 4 4]]

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

y = np.repeat(x, [2, 3], axis=1)
print(y)
# [[1 1 2 2 2]
#  [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。

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

x = np.arange(4)
y = np.ones((3, 4))
print(x.shape)  # (4,)
print(y.shape)  # (3, 4)

print((x + y).shape)  # (3, 4)
print(x + y)
# [[1. 2. 3. 4.]
#  [1. 2. 3. 4.]
#  [1. 2. 3. 4.]]
```

【例】两个数组均需要广播

```
import numpy as np

x = np.arange(4).reshape(4, 1)
y = np.ones(5)

print(x.shape)  # (4, 1)
print(y.shape)  # (5,)

print((x + y).shape)  # (4, 5)
print(x + y)
# [[1. 1. 1. 1. 1.]
#  [2. 2. 2. 2. 2.]
#  [3. 3. 3. 3. 3.]
#  [4. 4. 4. 4. 4.]]
```

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

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

```
x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = x + 1
print(y)
print(np.add(x, 1))
# [2 3 4 5 6 7 8 9]

y = x - 1
print(y)
print(np.subtract(x, 1))
# [0 1 2 3 4 5 6 7]

y = x * 2
print(y)
print(np.multiply(x, 2))
# [ 2  4  6  8 10 12 14 16]

y = x / 2
print(y)
print(np.divide(x, 2))
# [0.5 1.  1.5 2.  2.5 3.  3.5 4. ]

y = x // 2
print(y)
print(np.floor_divide(x, 2))
# [0 1 1 2 2 3 3 4]

y = x ** 2
print(y)
print(np.power(x, 2))
# [ 1  4  9 16 25 36 49 64]
```

【例】注意 numpy 的广播规则。

```
import numpy as np

x = np.array([[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]])
y = x + 1
print(y)
print(np.add(x, 1))
# [[12 13 14 15 16]
#  [17 18 19 20 21]
#  [22 23 24 25 26]
#  [27 28 29 30 31]
#  [32 33 34 35 36]]

y = x - 1
print(y)
print(np.subtract(x, 1))
```

```
# [[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]]

y = x * 2
print(y)
print(np.multiply(x, 2))
# [[22 24 26 28 30]
#  [32 34 36 38 40]
#  [42 44 46 48 50]
#  [52 54 56 58 60]
#  [62 64 66 68 70]]

y = x / 2
print(y)
print(np.divide(x, 2))
# [[ 5.5  6.   6.5  7.   7.5]
#  [ 8.   8.5  9.   9.5 10. ]
#  [10.5 11.  11.5 12.  12.5]
#  [13.  13.5 14.  14.5 15. ]
#  [15.5 16.  16.5 17.  17.5]]

y = x // 2
print(y)
print(np.floor_divide(x, 2))
# [[ 5  6  6  7  7]
#  [ 8  8  9  9 10]
#  [10 11 11 12 12]
#  [13 13 14 14 15]
#  [15 16 16 17 17]]

y = x ** 2
print(y)
print(np.power(x, 2))
# [[ 121  144  169  196  225]
#  [ 256  289  324  361  400]
#  [ 441  484  529  576  625]
#  [ 676  729  784  841  900]
#  [ 961 1024 1089 1156 1225]]
```

【例】注意 numpy 的广播规则。

```
import numpy as np

x = np.array([[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]])

y = np.arange(1, 6)
```

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

z = x + y
print(z)
print(np.add(x, y))
# [[12 14 16 18 20]
#  [17 19 21 23 25]
#  [22 24 26 28 30]
#  [27 29 31 33 35]
#  [32 34 36 38 40]]

z = x - y
print(z)
print(np.subtract(x, y))
# [[10 10 10 10 10]
#  [15 15 15 15 15]
#  [20 20 20 20 20]
#  [25 25 25 25 25]
#  [30 30 30 30 30]]

z = x * y
print(z)
print(np.multiply(x, y))
# [[ 11  24  39  56  75]
#  [ 16  34  54  76 100]
#  [ 21  44  69  96 125]
#  [ 26  54  84 116 150]
#  [ 31  64  99 136 175]]

z = x / y
print(z)
print(np.divide(x, y))
# [[11.          6.          4.33333333  3.5         3.        ]
#  [16.          8.5         6.          4.75        4.        ]
#  [21.         11.          7.66666667  6.          5.        ]
#  [26.         13.5         9.33333333  7.25        6.        ]
#  [31.         16.         11.          8.5         7.        ]]

z = x // y
print(z)
print(np.floor_divide(x, y))
# [[11  6  4  3  3]
#  [16  8  6  4  4]
#  [21 11  7  6  5]
#  [26 13  9  7  6]
#  [31 16 11  8  7]]

z = x ** np.full([1, 5], 2)
print(z)
print(np.power(x, np.full([5, 5], 2)))
# [[ 121  144  169  196  225]
```

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

【例】

```
import numpy as np

x = np.array([[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]])

y = np.arange(1, 26).reshape([5, 5])
print(y)
# [[ 1  2  3  4  5]
#  [ 6  7  8  9 10]
#  [11 12 13 14 15]
#  [16 17 18 19 20]
#  [21 22 23 24 25]]

z = x + y
print(z)
print(np.add(x, y))
# [[12 14 16 18 20]
#  [22 24 26 28 30]
#  [32 34 36 38 40]
#  [42 44 46 48 50]
#  [52 54 56 58 60]]

z = x - y
print(z)
print(np.subtract(x, y))
# [[10 10 10 10 10]
#  [10 10 10 10 10]
#  [10 10 10 10 10]
#  [10 10 10 10 10]
#  [10 10 10 10 10]]

z = x * y
print(z)
print(np.multiply(x, y))
# [[ 11  24  39  56  75]
#  [ 96 119 144 171 200]
#  [231 264 299 336 375]
#  [416 459 504 551 600]
#  [651 704 759 816 875]]

z = x / y
print(z)
print(np.divide(x, y))
```

```
# [[11.          6.          4.33333333  3.5         3.        ]
#  [ 2.66666667  2.42857143  2.25        2.11111111  2.        ]
#  [ 1.90909091  1.83333333  1.76923077  1.71428571  1.66666667]
#  [ 1.625       1.58823529  1.55555556  1.52631579  1.5       ]
#  [ 1.47619048  1.45454545  1.43478261  1.41666667  1.4       ]]

z = x // y
print(z)
print(np.floor_divide(x, y))
# [[11  6  4  3  3]
#  [ 2  2  2  2  2]
#  [ 1  1  1  1  1]
#  [ 1  1  1  1  1]
#  [ 1  1  1  1  1]]

z = x ** np.full([5, 5], 2)
print(z)
print(np.power(x, np.full([5, 5], 2)))
# [[ 121  144  169  196  225]
#  [ 256  289  324  361  400]
#  [ 441  484  529  576  625]
#  [ 676  729  784  841  900]
#  [ 961 1024 1089 1156 1225]]
```

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.

【例】

```
import numpy as np

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

y = np.sqrt(x)
print(y)
# [1.         1.41421356 1.73205081 2.        ]
print(np.power(x, 0.5))
# [1.         1.41421356 1.73205081 2.        ]

y = np.square(x)
print(y)
# [ 1  4  9 16]
```

```
print(np.power(x, 2))
# [ 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】

```
import numpy as np

```

```
x = np.linspace(start=0, stop=np.pi / 2, num=10)
print(x)
# [0.         0.17453293 0.34906585 0.52359878 0.6981317  0.87266463
#  1.04719755 1.22173048 1.3962634  1.57079633]

y = np.sin(x)
print(y)
# [0.         0.17364818 0.34202014 0.5        0.64278761 0.76604444
#  0.8660254  0.93969262 0.98480775 1.        ]

z = np.arcsin(y)
print(z)
# [0.         0.17453293 0.34906585 0.52359878 0.6981317  0.87266463
#  1.04719755 1.22173048 1.3962634  1.57079633]

y = np.cos(x)
print(y)
# [1.00000000e+00 9.84807753e-01 9.39692621e-01 8.66025404e-01
#  7.66044443e-01 6.42787610e-01 5.00000000e-01 3.42020143e-01
#  1.73648178e-01 6.12323400e-17]

z = np.arccos(y)
print(z)
# [0.         0.17453293 0.34906585 0.52359878 0.6981317  0.87266463
#  1.04719755 1.22173048 1.3962634  1.57079633]

y = np.tan(x)
print(y)
# [0.00000000e+00 1.76326981e-01 3.63970234e-01 5.77350269e-01
#  8.39099631e-01 1.19175359e+00 1.73205081e+00 2.74747742e+00
#  5.67128182e+00 1.63312394e+16]

z = np.arctan(y)
print(z)
# [0.         0.17453293 0.34906585 0.52359878 0.6981317  0.87266463
#  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

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

【例】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`.

```
import numpy as np

x = np.arange(1, 5)
print(x)
# [1 2 3 4]
y = np.exp(x)
print(y)
# [ 2.71828183  7.3890561  20.08553692 54.59815003]
z = np.log(y)
print(z)
# [1. 2. 3. 4.]
```

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。

```
import numpy as np

x = np.array([[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]])
y = np.sum(x)
print(y)  # 575

y = np.sum(x, axis=0)
print(y)  # [105 110 115 120 125]

y = np.sum(x, axis=1)
print(y)  # [ 65  90 115 140 165]
```

【例】返回给定轴上的数组元素的累加和。

```
import numpy as np

x = np.array([[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]])
y = np.cumsum(x)
print(y)
# [ 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]

y = np.cumsum(x, axis=0)
print(y)
# [[ 11  12  13  14  15]
#  [ 27  29  31  33  35]
#  [ 48  51  54  57  60]
#  [ 74  78  82  86  90]
#  [105 110 115 120 125]]

y = np.cumsum(x, axis=1)
print(y)
# [[ 11  23  36  50  65]
#  [ 16  33  51  70  90]
#  [ 21  43  66  90 115]
#  [ 26  53  81 110 140]
```

```
#  [ 31  63  96 130 165]]
```

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。

```
import numpy as np

x = np.array([[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]])
y = np.prod(x)
print(y)  # 788529152

y = np.prod(x, axis=0)
print(y)
# [2978976 3877632 4972968 6294624 7875000]

y = np.prod(x, axis=1)
print(y)
# [  360360  1860480  6375600 17100720 38955840]
```

【例】返回给定轴上数组元素的累乘。

```
import numpy as np

x = np.array([[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]])
y = np.cumprod(x)
print(y)
# [         11         132        1716       24024      360360     5765760
#     98017920  1764322560  -837609728   427674624   391232512    17180672
#    395155456   893796352   870072320  1147043840   905412608  -418250752
#    755630080  1194065920 -1638662144  -897581056   444596224 -2063597568
#    788529152]

y = np.cumprod(x, axis=0)
print(y)
```

```
# [[     11      12      13      14      15]
#  [    176     204     234     266     300]
#  [   3696    4488    5382    6384    7500]
#  [  96096  121176  150696  185136  225000]
#  [2978976 3877632 4972968 6294624 7875000]]

y = np.cumprod(x, axis=1)
print(y)
# [[      11      132     1716    24024   360360]
#  [      16      272     4896    93024  1860480]
#  [      21      462    10626   255024  6375600]
#  [      26      702    19656   570024 17100720]
#  [      31      992    32736  1113024 38955840]]
```

11.4.5 numpy.diff 差值

1. `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。

```
import numpy as np

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

x = np.random.rand(3, 3) * 10
print(x)
# [[6.59144457 3.78566113 8.15321227]
#  [1.68241475 3.78753332 7.68886328]
#  [2.84255822 9.58106727 7.86678037]]

y = np.around(x)
print(y)
# [[ 7.  4.  8.]
#  [ 2.  4.  8.]
#  [ 3. 10.  8.]]

y = np.around(x, decimals=2)
print(y)
# [[6.59 3.79 8.15]
#  [1.68 3.79 7.69]
#  [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.

【例】

```
import numpy as np

x = np.random.rand(3, 3) * 10
print(x)
# [[0.67847795 1.33073923 4.53920122]
#  [7.55724676 5.88854047 2.65502046]
#  [8.67640444 8.80110812 5.97528726]]

y = np.ceil(x)
print(y)
# [[1. 2. 5.]
#  [8. 6. 3.]
```

```
#  [9. 9. 6.]]

y = np.floor(x)
print(y)
# [[0. 1. 4.]
#  [7. 5. 2.]
#  [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。

```
import numpy as np

x = np.array([[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]])
y = np.clip(x, a_min=20, a_max=30)
print(y)
# [[20 20 20 20 20]
#  [20 20 20 20 20]
#  [21 22 23 24 25]
#  [26 27 28 29 30]
#  [30 30 30 30 30]]
```

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.

【例】

```python
import numpy as np

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

y = np.abs(x)
print(y)
# [5 4 3 2 1 0 1 2 3 4]

y = np.absolute(x)
print(y)
# [5 4 3 2 1 0 1 2 3 4]
```

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】

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

参考文献

1. https://mp.weixin.qq.com/s/RWsGvvmw4ptf7d8zPIDEJw
2. 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】

```
import numpy as np

a = np.array([0, 4, 5])
b = np.copy(a)
print(np.all(a == b))  # True
print(np.any(a == b))  # True

b[0] = 1
print(np.all(a == b))  # False
print(np.any(a == b))  # True

print(np.all([1.0, np.nan]))  # True
print(np.any([1.0, np.nan]))  # True

a = np.eye(3)
print(np.all(a, axis=0))  # [False False False]
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】

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

```

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。

import numpy as np

print(np.logical_not(3))
# False
print(np.logical_not([True, False, 0, 1]))
# [False  True  True False]

x = np.arange(5)
print(np.logical_not(x < 3))
# [False 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  # False
18  print(np.logical_and([True, False], [True, False]))
19  # [ True False]
20  print(np.logical_and(x > 1, x < 4))
21  # [False False  True  True False]
22
23  【例】逐元素计算x1 OR x2的真值。
24
25
26  print(np.logical_or(True, False))
27  # True
28  print(np.logical_or([True, False], [False, False]))
29  # [ True False]
30  print(np.logical_or(x < 1, x > 3))
31  # [ True False False False  True]
32
33  【例】计算x1 XOR x2的真值，按元素计算。
34
35  print(np.logical_xor(True, False))
36  # True
37  print(np.logical_xor([True, True, False, False], [True, False, True, False]))
38  # [False  True  True False]
39  print(np.logical_xor(x < 1, x > 3))
40  # [ True False False False  True]
41  print(np.logical_xor(0, np.eye(2)))
42  # [[ True False]
43  #  [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

1. `numpy.greater(x1, x2, *args, **kwargs)` Return the truth value of (x1 > x2) element-wise.
2. `numpy.greater_equal(x1, x2, *args, **kwargs)` Return the truth value of (x1 >= x2) element-wise.
3. `numpy.equal(x1, x2, *args, **kwargs)` Return (x1 == x2) element-wise.
4. `numpy.not_equal(x1, x2, *args, **kwargs)` Return (x1 != x2) element-wise.
5. `numpy.less(x1, x2, *args, **kwargs)` Return the truth value of (x1 < x2) element-wise.
6. `numpy.less_equal(x1, x2, *args, **kwargs)` Return the truth value of (x1 =< x2) element-wise.

【例】numpy对以上对照函数进行了运算符的重载。

```
import numpy as np

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

y = x > 2
print(y)
print(np.greater(x, 2))
# [False False  True  True  True  True  True  True]

y = x >= 2
print(y)
print(np.greater_equal(x, 2))
# [False  True  True  True  True  True  True  True]

y = x == 2
print(y)
print(np.equal(x, 2))
# [False  True False False False False False False]

y = x != 2
print(y)
print(np.not_equal(x, 2))
# [ True False  True  True  True  True  True  True]

y = x < 2
print(y)
print(np.less(x, 2))
# [ True False False False False False False False]

```

```
y = x <= 2
print(y)
print(np.less_equal(x, 2))
# [ True  True False False False False False False]
```

【例】

```
import numpy as np

x = np.array([[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]])
y = x > 20
print(y)
print(np.greater(x, 20))
# [[False False False False False]
#  [False False False False False]
#  [ True  True  True  True  True]
#  [ True  True  True  True  True]
#  [ True  True  True  True  True]]

y = x >= 20
print(y)
print(np.greater_equal(x, 20))
# [[False False False False False]
#  [False False False False  True]
#  [ True  True  True  True  True]
#  [ True  True  True  True  True]
#  [ True  True  True  True  True]]

y = x == 20
print(y)
print(np.equal(x, 20))
# [[False False False False False]
#  [False False False False  True]
#  [False False False False False]
#  [False False False False False]
#  [False False False False False]]

y = x != 20
print(y)
print(np.not_equal(x, 20))
# [[ True  True  True  True  True]
#  [ True  True  True  True False]
#  [ True  True  True  True  True]
#  [ True  True  True  True  True]
#  [ True  True  True  True  True]]


y = x < 20
print(y)
```

```
print(np.less(x, 20))
# [[ True  True  True  True  True]
#  [ True  True  True  True False]
#  [False False False False False]
#  [False False False False False]
#  [False False False False False]]

y = x <= 20
print(y)
print(np.less_equal(x, 20))
# [[ True  True  True  True  True]
#  [ True  True  True  True  True]
#  [False False False False False]
#  [False False False False False]
#  [False False False False False]]
```

【例】

```
import numpy as np

np.random.seed(20200611)
x = np.array([[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]])

y = np.random.randint(10, 40, [5, 5])
print(y)
# [[32 28 31 33 37]
#  [23 37 37 30 29]
#  [32 24 10 33 15]
#  [27 17 10 36 16]
#  [25 32 23 39 34]]

z = x > y
print(z)
print(np.greater(x, y))
# [[False False False False False]
#  [False False False False False]
#  [False False  True False  True]
#  [False  True  True False  True]
#  [ True False  True False  True]]

z = x >= y
print(z)
print(np.greater_equal(x, y))
# [[False False False False False]
#  [False False False False False]
#  [False False  True False  True]
#  [False  True  True False  True]
#  [ True  True  True False  True]]

```

```
z = x == y
print(z)
print(np.equal(x, y))
# [[False False False False False]
#  [False False False False False]
#  [False False False False False]
#  [False False False False False]
#  [False  True False False False]]

z = x != y
print(z)
print(np.not_equal(x, y))
# [[ True  True  True  True  True]
#  [ True  True  True  True  True]
#  [ True  True  True  True  True]
#  [ True  True  True  True  True]
#  [ True False  True  True  True]]

z = x < y
print(z)
print(np.less(x, y))
# [[ True  True  True  True  True]
#  [ True  True  True  True  True]
#  [ True  True False  True False]
#  [ True False False  True False]
#  [False False False  True False]]

z = x <= y
print(z)
print(np.less_equal(x, y))
# [[ True  True  True  True  True]
#  [ True  True  True  True  True]
#  [ True  True False  True False]
#  [ True False False  True False]
#  [False  True False  True False]]
```

【例】注意 numpy 的广播规则。

```
import numpy as np

x = np.array([[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]])

np.random.seed(20200611)
y = np.random.randint(10, 50, 5)

print(y)
# [32 37 30 24 10]

z = x > y
```

```
print(z)
print(np.greater(x, y))
# [[False False False False  True]
#  [False False False False  True]
#  [False False False False  True]
#  [False False False  True  True]
#  [False False  True  True  True]]

z = x >= y
print(z)
print(np.greater_equal(x, y))
# [[False False False False  True]
#  [False False False False  True]
#  [False False False  True  True]
#  [False False False  True  True]
#  [False False  True  True  True]]

z = x == y
print(z)
print(np.equal(x, y))
# [[False False False False False]
#  [False False False False False]
#  [False False False  True False]
#  [False False False False False]
#  [False False False False False]]

z = x != y
print(z)
print(np.not_equal(x, y))
# [[ True  True  True  True  True]
#  [ True  True  True  True  True]
#  [ True  True  True False  True]
#  [ True  True  True  True  True]
#  [ True  True  True  True  True]]

z = x < y
print(z)
print(np.less(x, y))
# [[ True  True  True  True False]
#  [ True  True  True  True False]
#  [ True  True  True False False]
#  [ True  True  True False False]
#  [ True  True False False False]]

z = x <= y
print(z)
print(np.less_equal(x, y))
# [[ True  True  True  True False]
#  [ True  True  True  True False]
#  [ True  True  True  True False]
#  [ True  True  True False False]
#  [ 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：

```
np.absolute(a - b) <= (atol + rtol * absolute(b))

- atol：float，绝对公差。
- 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

x = np.isclose([1e10, 1e-7], [1.00001e10, 1e-8])
print(x)  # [ True False]

x = np.allclose([1e10, 1e-7], [1.00001e10, 1e-8])
print(x)  # False

x = np.isclose([1e10, 1e-8], [1.00001e10, 1e-9])
print(x)  # [ True  True]

x = np.allclose([1e10, 1e-8], [1.00001e10, 1e-9])
print(x)  # True

x = np.isclose([1e10, 1e-8], [1.0001e10, 1e-9])
print(x)  # [False  True]

x = np.allclose([1e10, 1e-8], [1.0001e10, 1e-9])
print(x)  # False

x = np.isclose([1.0, np.nan], [1.0, np.nan])
print(x)  # [ True False]

x = np.allclose([1.0, np.nan], [1.0, np.nan])
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。

【例】

```
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]
#  [0.01 4.23 0.19 1.73 9.27]
#  [7.99 4.97 0.88 7.32 4.29]
#  [9.05 0.07 8.95 7.9  6.99]]

y = np.sort(x)
print(y)
# [[1.73 2.32 6.22 7.54 9.78]
#  [5.17 6.93 8.25 9.28 9.76]
#  [0.01 0.19 1.73 4.23 9.27]
#  [0.88 4.29 4.97 7.32 7.99]
#  [0.07 6.99 7.9  8.95 9.05]]

y = np.sort(x, axis=0)
print(y)
# [[0.01 0.07 0.19 1.73 4.29]
#  [2.32 4.23 0.88 1.73 6.22]
#  [6.93 4.97 8.95 7.32 6.99]
#  [7.99 5.17 9.28 7.9  8.25]
#  [9.05 7.54 9.78 9.76 9.27]]

y = np.sort(x, axis=1)
print(y)
# [[1.73 2.32 6.22 7.54 9.78]
#  [5.17 6.93 8.25 9.28 9.76]
#  [0.01 0.19 1.73 4.23 9.27]
#  [0.88 4.29 4.97 7.32 7.99]
```

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

【例】

```
import numpy as np

dt = np.dtype([('name', 'S10'), ('age', np.int)])
a = np.array([("Mike", 21), ("Nancy", 25), ("Bob", 17), ("Jane", 27)], dtype=dt)
b = np.sort(a, order='name')
print(b)
# [(b'Bob', 17) (b'Jane', 27) (b'Mike', 21) (b'Nancy', 25)]

b = np.sort(a, order='age')
print(b)
# [(b'Bob', 17) (b'Mike', 21) (b'Nancy', 25) (b'Jane', 27)]
```

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。

```
import numpy as np

np.random.seed(20200612)
x = np.random.randint(0, 10, 10)
print(x)
# [6 1 8 5 5 4 1 2 9 1]

y = np.argsort(x)
print(y)
# [1 6 9 7 5 3 4 0 2 8]

print(x[y])
# [1 1 1 2 4 5 5 6 8 9]

y = np.argsort(-x)
print(y)
# [8 2 0 3 4 5 7 1 6 9]

print(x[y])
# [9 8 6 5 5 4 2 1 1 1]
```

【例】

```
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]
#  [0.01 4.23 0.19 1.73 9.27]
#  [7.99 4.97 0.88 7.32 4.29]
```

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

y = np.argsort(x)
print(y)
# [[3 0 4 1 2]
#  [1 0 4 2 3]
#  [0 2 3 1 4]
#  [2 4 1 3 0]
#  [1 4 3 2 0]]

y = np.argsort(x, axis=0)
print(y)
# [[2 4 2 0 3]
#  [0 2 3 2 0]
#  [1 3 4 3 4]
#  [3 1 1 4 1]
#  [4 0 0 1 2]]

y = np.argsort(x, axis=1)
print(y)
# [[3 0 4 1 2]
#  [1 0 4 2 3]
#  [0 2 3 1 4]
#  [2 4 1 3 0]
#  [1 4 3 2 0]]

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。
print(y)
# [[1.73 2.32 6.22 7.54 9.78]
#  [5.17 6.93 8.25 9.28 9.76]
#  [0.01 0.19 1.73 4.23 9.27]
#  [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]
```

```
#  [0.01 4.23 0.19 1.73 9.27]
#  [7.99 4.97 0.88 7.32 4.29]
#  [9.05 0.07 8.95 7.9  6.99]]

index = np.lexsort([x[:, 0]])
print(index)
# [2 0 1 3 4]

y = x[index]
print(y)
# [[0.01 4.23 0.19 1.73 9.27]
#  [2.32 7.54 9.78 1.73 6.22]
#  [6.93 5.17 9.28 9.76 8.25]
#  [7.99 4.97 0.88 7.32 4.29]
#  [9.05 0.07 8.95 7.9  6.99]]

index = np.lexsort([-1 * x[:, 0]])
print(index)
# [4 3 1 0 2]

y = x[index]
print(y)
# [[9.05 0.07 8.95 7.9  6.99]
#  [7.99 4.97 0.88 7.32 4.29]
#  [6.93 5.17 9.28 9.76 8.25]
#  [2.32 7.54 9.78 1.73 6.22]
#  [0.01 4.23 0.19 1.73 9.27]]
```

【例】

```
import numpy as np

x = np.array([1, 5, 1, 4, 3, 4, 4])
y = np.array([9, 4, 0, 4, 0, 2, 1])
a = np.lexsort([x])
b = np.lexsort([y])
print(a)
# [0 2 4 3 5 6 1]
print(x[a])
# [1 1 3 4 4 4 5]

print(b)
# [2 4 6 5 1 3 0]
print(y[b])
# [0 0 1 2 4 4 9]

z = np.lexsort([y, x])
print(z)
# [2 0 4 6 5 3 1]
print(x[z])
# [1 1 3 4 4 4 5]

z = np.lexsort([x, y])
```

```
print(z)
# [2 4 6 5 3 1 0]
print(y[z])
# [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。

```
import numpy as np

np.random.seed(100)
x = np.random.randint(1, 30, [8, 3])
print(x)
# [[ 9 25  4]
#  [ 8 24 16]
#  [17 11 21]
#  [ 3 22  3]
#  [ 3 15  3]
#  [18 17 25]
#  [16  5 12]
#  [29 27 17]]

y = np.sort(x, axis=0)
print(y)
# [[ 3  5  3]
#  [ 3 11  3]
#  [ 8 15  4]
#  [ 9 17 12]
#  [16 22 16]
#  [17 24 17]
#  [18 25 21]
#  [29 27 25]]

z = np.partition(x, kth=2, axis=0)
print(z)
# [[ 3  5  3]
#  [ 3 11  3]
#  [ 8 15  4]
#  [ 9 22 21]
#  [17 24 16]
#  [18 17 25]
#  [16 25 12]
#  [29 27 17]]
```

【例】选取每一列第三小的数

```
import numpy as np

np.random.seed(100)
```

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

【例】选取每一列第三大的数据

```
import numpy as np

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

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.

【例】

```
import numpy as np

np.random.seed(100)
x = np.random.randint(1, 30, [8, 3])
print(x)
# [[ 9 25  4]
#  [ 8 24 16]
#  [17 11 21]
#  [ 3 22  3]
#  [ 3 15  3]
#  [18 17 25]
#  [16  5 12]
#  [29 27 17]]

```

```
y = np.argsort(x, axis=0)
print(y)
# [[3 6 3]
#  [4 2 4]
#  [1 4 0]
#  [0 5 6]
#  [6 3 1]
#  [2 1 7]
#  [5 0 2]
#  [7 7 5]]

z = np.argpartition(x, kth=2, axis=0)
print(z)
# [[3 6 3]
#  [4 2 4]
#  [1 4 0]
#  [0 3 2]
#  [2 1 1]
#  [5 5 5]
#  [6 0 6]
#  [7 7 7]]
```

【例】选取每一列第三小的数的索引

```
import numpy as np

np.random.seed(100)
x = np.random.randint(1, 30, [8, 3])
print(x)
# [[ 9 25  4]
#  [ 8 24 16]
#  [17 11 21]
#  [ 3 22  3]
#  [ 3 15  3]
#  [18 17 25]
#  [16  5 12]
#  [29 27 17]]

z = np.argpartition(x, kth=2, axis=0)
print(z[2])
# [1 4 0]
```

【例】选取每一列第三大的数的索引

```
import numpy as np

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

```
#  [18 17 25]
#  [16  5 12]
#  [29 27 17]]

z = np.argpartition(x, kth=-3, axis=0)
print(z[-3])
# [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

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]
#  [0.01 4.23 0.19 1.73 9.27]
#  [7.99 4.97 0.88 7.32 4.29]
#  [9.05 0.07 8.95 7.9  6.99]]

y = np.argmax(x)
print(y)  # 2

y = np.argmax(x, axis=0)
print(y)
# [4 0 0 1 2]

y = np.argmax(x, axis=1)
print(y)
# [2 3 4 0 0]
```

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

【例】

```
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]
```

```
#  [0.01 4.23 0.19 1.73 9.27]
#  [7.99 4.97 0.88 7.32 4.29]
#  [9.05 0.07 8.95 7.9  6.99]]

y = np.argmin(x)
print(y)  # 10

y = np.argmin(x, axis=0)
print(y)
# [2 4 2 0 3]

y = np.argmin(x, axis=1)
print(y)
# [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

x = np.array([0, 2, 3])
print(x)  # [0 2 3]
print(x.shape)  # (3,)
print(x.ndim)  # 1

y = np.nonzero(x)
print(y)  # (array([1, 2], dtype=int64),)
print(np.array(y))  # [[1 2]]
print(np.array(y).shape)  # (1, 2)
print(np.array(y).ndim)  # 2
print(np.transpose(y))
# [[1]
#  [2]]
print(x[np.nonzero(x)])
#[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]]
print(x.shape)  # (3, 3)
print(x.ndim)  # 2

y = np.nonzero(x)
print(y)
# (array([0, 1, 2, 2], dtype=int64), array([0, 1, 0, 1], dtype=int64))
print(np.array(y))
# [[0 1 2 2]
#  [0 1 0 1]]
print(np.array(y).shape)  # (2, 4)
print(np.array(y).ndim)  # 2

y = x[np.nonzero(x)]
print(y)  # [3 4 5 6]

y = np.transpose(np.nonzero(x))
print(y)
# [[0 0]
#  [1 1]
#  [2 0]
#  [2 1]]
```

【例】三维数组

```
import numpy as np

x = np.array([[[0, 1], [1, 0]], [[0, 1], [1, 0]], [[0, 0], [1, 0]]])
print(x)
# [[[0 1]
#   [1 0]]
#
#  [[0 1]
#   [1 0]]
#
#  [[0 0]
#   [1 0]]]
print(np.shape(x))  # (3, 2, 2)
print(x.ndim)  # 3

y = np.nonzero(x)
print(np.array(y))
# [[0 0 1 1 2]
#  [0 1 0 1 1]
#  [1 0 1 0 0]]
print(np.array(y).shape)  # (3, 5)
print(np.array(y).ndim)  # 2
print(y)
# (array([0, 0, 1, 1, 2], dtype=int64), array([0, 1, 0, 1, 1], dtype=int64), array([1, 0, 1, 0, 0], dtype=int64))
print(x[np.nonzero(x)])
#[1 1 1 1 1]
```

【例】`nonzero()`将布尔数组转换成整数数组进行操作。

```

```

```python
import numpy as np

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

y = x > 3
print(y)
# [[False False False]
#  [ True  True  True]
#  [ True  True  True]]

y = np.nonzero(x > 3)
print(y)
# (array([1, 1, 1, 2, 2, 2], dtype=int64), array([0, 1, 2, 0, 1, 2], dtype=int64))

y = x[np.nonzero(x > 3)]
print(y)
# [4 5 6 7 8 9]

y = x[x > 3]
print(y)
# [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`。

```python
import numpy as np

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

y = np.where(x < 5, x, 10 * x)
print(y)
# [ 0  1  2  3  4 50 60 70 80 90]

x = np.array([[0, 1, 2],
              [0, 2, 4],
              [0, 3, 6]])
y = np.where(x < 4, x, -1)
print(y)
# [[ 0  1  2]
#  [ 0  2 -1]
#  [ 0  3 -1]]
```

【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。

```python
import numpy as np

```

```
x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = np.where(x > 5)
print(y)
# (array([5, 6, 7], dtype=int64),)
print(x[y])
# [6 7 8]

y = np.nonzero(x > 5)
print(y)
# (array([5, 6, 7], dtype=int64),)
print(x[y])
# [6 7 8]

x = np.array([[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]])
y = np.where(x > 25)
print(y)
# (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))

print(x[y])
# [26 27 28 29 30 31 32 33 34 35]

y = np.nonzero(x > 25)
print(y)
# (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))
print(x[y])
# [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。

【例】

```
import numpy as np

x = np.array([0, 1, 5, 9, 11, 18, 26, 33])
y = np.searchsorted(x, 15)
print(y)  # 5

y = np.searchsorted(x, 15, side='right')
print(y)  # 5

y = np.searchsorted(x, -1)
```

```
print(y)  # 0

y = np.searchsorted(x, -1, side='right')
print(y)  # 0

y = np.searchsorted(x, 35)
print(y)  # 8

y = np.searchsorted(x, 35, side='right')
print(y)  # 8

y = np.searchsorted(x, 11)
print(y)  # 4

y = np.searchsorted(x, 11, side='right')
print(y)  # 5

y = np.searchsorted(x, 0)
print(y)  # 0

y = np.searchsorted(x, 0, side='right')
print(y)  # 1

y = np.searchsorted(x, 33)
print(y)  # 7

y = np.searchsorted(x, 33, side='right')
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]
```

【例】

```
import numpy as np

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

x_sort = np.argsort(x)
print(x_sort)  # [6 1 7 2 4 3 5 0]

y = np.searchsorted(x, [-1, 0, 11, 15, 33, 35], sorter=x_sort)
print(y)  # [0 0 4 5 7 8]

```

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

```
import numpy as np

x = np.count_nonzero(np.eye(4))
print(x)  # 4

x = np.count_nonzero([[0, 1, 7, 0, 0], [3, 0, 0, 2, 19]])
print(x)  # 5

x = np.count_nonzero([[0, 1, 7, 0, 0], [3, 0, 0, 2, 19]], axis=0)
print(x)  # [1 1 1 1 1]

x = np.count_nonzero([[0, 1, 7, 0, 0], [3, 0, 0, 2, 19]], axis=1)
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

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=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。

```python
import numpy as np

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

x = sorted(set([1, 1, 3, 2, 3, 3]))
print(x)  # [1, 2, 3]

x = np.array([[1, 1], [2, 3]])
u = np.unique(x)
print(u)  # [1 2 3]

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

x = np.array(['a', 'b', 'b', 'c', 'a'])
u, index = np.unique(x, return_index=True)
print(u)  # ['a' 'b' 'c']
print(index)  # [0 1 3]
print(x[index])  # ['a' 'b' 'c']

x = np.array([1, 2, 6, 4, 2, 3, 2])
u, index = np.unique(x, return_inverse=True)
print(u)  # [1 2 3 4 6]
print(index)  # [0 1 4 3 1 2 1]
print(u[index])  # [1 2 6 4 2 3 2]

u, count = np.unique(x, return_counts=True)
print(u)  # [1 2 3 4 6]
```

```
print(count)  # [1 3 1 1 1]
```

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 array.

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

test = np.array([0, 1, 2, 5, 0])
states = [0, 2]
mask = np.in1d(test, states)
print(mask)  # [ True False  True False  True]
print(test[mask])  # [0 2 0]

mask = np.in1d(test, states, invert=True)
print(mask)  # [False  True False  True False]
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
from functools import reduce

x = np.intersect1d([1, 3, 4, 3], [3, 1, 2, 1])
print(x)  # [1 3]

x = np.array([1, 1, 2, 3, 4])
y = np.array([2, 1, 4, 6])
xy, x_ind, y_ind = np.intersect1d(x, y, return_indices=True)
print(x_ind)  # [0 2 4]
print(y_ind)  # [1 0 2]
print(xy)  # [1 2 4]
print(x[x_ind])  # [1 2 4]
print(y[y_ind])  # [1 2 4]

```

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

x = np.union1d([-1, 0, 1], [-2, 0, 2])
print(x)  # [-2 -1  0  1  2]
x = reduce(np.union1d, ([1, 3, 4, 3], [3, 1, 2, 1], [6, 3, 4, 2]))
print(x)  # [1 2 3 4 6]
'''
functools.reduce(function, iterable[, initializer])
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: 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。

大致相当于：
def reduce(function, iterable, initializer=None):
    it = iter(iterable)
    if initializer is None:
        value = next(it)
    else:
        value = initializer
    for element in it:
        value = function(value, element)
    return value
'''
```

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。

```python
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。

```python
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