Table of Contents

- 1 常用运算操作
  - 1.1 基本运算符
  - 1.2 基本运算函数
  - 1.3 复合赋值运算符
  - 1.4 矩阵运算
  - 1.5 判断符的妙用
  - 1.6 练习
- 2 聚合函数
  - 2.1 常用聚合函数
  - 2.2 Numpy聚合函数使用场景
- 3 Numpy的快速排序
  - 3.1 np.sort()
  - 3.2 np.argsort()
- 4 唯一化和集合逻辑
- 5 练习

In [1]:
```
#全部行都能输出
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import numpy as np
```

Numpy官网：http://www.numpy.org/ (http://www.numpy.org/)

Numpy 中数组上的算术运算符使用元素级别。最后的结果使用新的一个数组来返回。

1 常用运算操作

1.1 基本运算符

In [2]:
```
a = np.array([20,30,40,50])
b = np.arange(4)
a
b
```

Out[2]: array([20, 30, 40, 50])

Out[2]: array([0, 1, 2, 3])

In [3]:
```
c = a-b
c
```

Out[3]: array([20, 29, 38, 47])

In [4]:
```
b
b*2
b**2
```

Out[4]: array([0, 1, 2, 3])

Out[4]: array([0, 2, 4, 6])

Out[4]: array([0, 1, 4, 9], dtype=int32)

In [5]:
```
a<35
```

Out[5]: array([True, True, False, False])

1.2 基本运算函数

数组的运算符及其对应的 ufunc 函数

运 算 符	对应的 ufunc 函数
y = x1 + x2	add(x1, x2 [, y])
y = x1 - x2	subtract(x1, x2 [, y])
y = x1 * x2	multiply (x1, x2 [, y])
y = x1 / x2	divide (x1, x2 [, y]), 如果两个数组的元素为整数，那么用整数除法
y = x1 / x2	true_divide (x1, x2 [, y]), 总是返回精确的商
y = x1 // x2	floor_divide (x1, x2 [, y]), 总是对返回值取整
y = -x	negative(x [,y])
y = x1**x2	power(x1, x2 [, y])
y = x1 % x2	remainder(x1, x2 [, y]),或 mod(x1, x2, [, y])

In [3]:
```
a = np.array([20,30,40,50])
b = list(np.arange(1,5))
a
b
```

Out[3]: array([20, 30, 40, 50])

Out[3]: [1, 2, 3, 4]

In [4]:
```
np.add(a,b)
a+b
```

Out[4]: array([21, 32, 43, 54])

Out[4]: array([21, 32, 43, 54])

In [25]:
```
np.subtract(a,b)
a-b
```

Out[25]: array([19, 28, 37, 46])

Out[25]: array([19, 28, 37, 46])

In [26]:
```
np.multiply(a,b)
a*b
```

Out[26]:
```
array([ 20,  60, 120, 200])
```

Out[26]:
```
array([ 20,  60, 120, 200])
```

In [27]:
```
np.divide(a,b)
a/b
```

Out[27]:
```
array([20.        , 15.        , 13.33333333, 12.5       ])
```

Out[27]:
```
array([20.        , 15.        , 13.33333333, 12.5       ])
```

In [29]:
```
a = np.array([2,3,4,5])
b = np.full(4,2)
a
b

np.power(a,b)
a**b
```

Out[29]:
```
array([2, 3, 4, 5])
```

Out[29]:
```
array([2, 2, 2, 2])
```

Out[29]:
```
array([ 4,  9, 16, 25], dtype=int32)
```

Out[29]:
```
array([ 4,  9, 16, 25], dtype=int32)
```

1.3 复合赋值运算符

某些操作（如+=和*=）可以修改现有数组，而不是创建新数组。

In [30]:
```
a=np.array([[1, 1],
            [0, 1]])

b=np.array([[2, 0],
            [3, 4]])

b+=a
b
```

Out[30]:
```
array([[3, 1],
       [3, 5]])
```

In [31]:
```
a = np.ones((2,3), dtype=np.int32)
a

a *= 3
a
```

Out[31]:
```
array([[1, 1, 1],
       [1, 1, 1]])
```

Out[31]:
```
array([[3, 3, 3],
       [3, 3, 3]])
```

1.4 矩阵运算

- 需要注意的是，乘法运算符*的运算在NumPy数组中也是元素级别的。
- 如果想要执行矩阵乘积，可以使用dot函数：
- dot(a, b, out=None)
  - 如果'a'和'b'都是1-D数组，它就是向量的内积。
  - 如果'a'和'b'都是二维数组，那就是矩阵乘法。
  - 如果'a'或'b'是0-D（标量），它相当于'numpy.multiply（a，b）'或'a * b'是首选。
  - 如果'a'是N-D数组而'b'是1-D数组，则它是和的乘积'a'和'b'的最后一个轴。

如果'a'和'b'都是1-D数组，它就是向量的内积。

In [5]:
```
a = np.array([1,1])
b = np.array([2,0])
a
b
```

Out[5]:
```
array([1, 1])
```

Out[5]:
```
array([2, 0])
```

In [8]:
```
b.dot(b)
```

Out[8]:
```
4
```

如果'a'和'b'都是二维数组，那就是矩阵乘法。

In [37]:
```
c = np.array( [[1,1], [0,1]] )
d = np.array( [[2,0], [3,4]] )
c
d
```

Out[37]:
```
array([[1, 1],
       [0, 1]])
```

Out[37]:
```
array([[2, 0],
       [3, 4]])
```

In [39]:
```
c.dot(d)                    # 矩阵相乘（matrix product）
# np.dot(c, d)              # 矩阵相乘的另一种方式（another matrix product）
```

Out[39]:
```
array([[5, 4],
       [3, 4]])
```

Out[39]:
```
array([[5, 4],
       [3, 4]])
```

1.5 判断符的妙用

比较运算符及其对应的 ufunc 函数

比较运算符	ufunc 函数
y = x1 == x2	equal(x1, x2 [, y])
y = x1 != x2	not_equal(x1, x2 [, y])
y = x1 < x2	less(x1, x2, [, y])
y = x1 <= x2	less_equal(x1, x2, [, y])
y = x1 > x2	greater(x1, x2, [, y])
y = x1 >= x2	greater_equal(x1, x2, [, y])

In [5]:
```
a=np.array([1,2,3])
b=np.array([True,False,True])

a[b]
```

Out[5]:
```
array([1, 3])
```

In [40]:
```
np.random.seed(0)

a=np.random.randint(10,size=(3,3))
a
```

Out[40]:
```
array([[5, 0, 3],
       [3, 7, 9],
       [3, 5, 2]])
```

In [41]:
```
a>5
a[a>5]   #提取a数组中大于5的数字
```

Out[41]:
```
array([[False, False, False],
       [False,  True,  True],
       [False, False, False]])
```

Out[41]:
```
array([7, 9])
```

如果想要将数组a中非偶数的元素删除：

In [42]:
```
[a%2==0]
a[a%2==0]
```

Out[42]:
```
[array([[False,  True, False],
        [False, False, False],
        [False, False,  True]])]
```

Out[42]:
```
array([0, 2])
```

1.6 练习

In [32]:
```
np.random.seed(0)

a=np.random.randint(1,20,(5,5))
a
```

Out[32]:
```
array([[13, 16,  1,  4,  4],
       [ 8, 10, 19,  5,  7],
       [13,  2,  7,  8, 15],
       [18,  6, 14,  9, 10],
       [17,  6, 16, 16,  1]])
```

1、如何查找出能被3整除的数?
2、如何查找出32和30的公约数?

In [ ]:
```

```

2 聚合函数

下面所有的函数都支持axis来指定不同的轴，用法都是类似的。

2.1 常用聚合函数

函数	说明
sum	对数组中全部或某轴向的元素。零长度的数组sum为0
mean	算数平均数。零长度的数组mean为nan
std、var	标准差、方差
min、max	最大值、最小值
argmin、argmax	最大和最小元素的索引
cumsum、cumprod	累计和、累计积

In [15]:
```
np.random.seed(0)
x = np.random.randint(10, size=(2,3))
x
```

Out[15]:
```
array([[5, 0, 3],
       [3, 7, 9]])
```

In [60]:
```
np.sum(x)           #矩阵所有元素都求和一遍
np.sum(x,axis=0)    #axis=0，从单独一列看，所有行的值求和
np.sum(x,axis=1)     #axis=1，从单独一行看，所有列的值求和
```

Out[60]: 27

Out[60]: array([8, 7, 12])

Out[60]: array([8, 19])

上面的求和方式可以通过数组对象来调用.sum()函数：

In [65]:
```
x

x.sum()           #矩阵所有元素都求和一遍
x.sum(axis=0)    #axis=0，从单独一列看，所有行的值求和
x.sum(axis=1)     #axis=1，从单独一行看，所有列的值求和
```

Out[65]:
```
array([[5, 0, 3],
       [3, 7, 9]])
```

Out[65]: 27

Out[65]: array([8, 7, 12])

Out[65]: array([8, 19])

In [66]:
```
x

np.mean(x)
np.mean(x, axis=0)
np.mean(x, axis=1)
```

Out[66]:
```
array([[5, 0, 3],
       [3, 7, 9]])
```

Out[66]: 4.5

Out[66]: array([4. , 3.5, 6.])

Out[66]: array([2.66666667, 6.33333333])

In [67]:
```
x

np.std(x)
np.std(x, axis=0)
np.std(x, axis=1)
```

Out[67]:
```
array([[5, 0, 3],
       [3, 7, 9]])
```

Out[67]: 2.9297326385411577

Out[67]: array([1. , 3.5, 3.])

Out[67]: array([2.05480467, 2.49443826])

In [68]:
```
x

np.var(x)
np.var(x, axis=0)
np.var(x, axis=1)
```

Out[68]:
```
array([[5, 0, 3],
       [3, 7, 9]])
```

Out[68]:
```
8.583333333333334
```

Out[68]:
```
array([ 1.  , 12.25,  9.  ])
```

Out[68]:
```
array([4.22222222, 6.22222222])
```

In [69]:
```
x

np.max(x)
np.max(x, axis=0)
np.max(x, axis=1)
```

Out[69]:
```
array([[5, 0, 3],
       [3, 7, 9]])
```

Out[69]:
```
9
```

Out[69]:
```
array([5, 7, 9])
```

Out[69]:
```
array([5, 9])
```

In [70]:
```
x

np.cumsum(x)
np.cumsum(x, axis=0)
np.cumsum(x, axis=1)  #axis=0，从单独一列看，所有行的值逐次累加
```

Out[70]:
```
array([[5, 0, 3],
       [3, 7, 9]])
```

Out[70]:
```
array([ 5,  5,  8, 11, 18, 27], dtype=int32)
```

Out[70]:
```
array([[ 5,  0,  3],
       [ 8,  7, 12]], dtype=int32)
```

Out[70]:
```
array([[ 5,  5,  8],
       [ 3, 10, 19]], dtype=int32)
```

In [71]:
```
x=np.array([[5, 0, 3],
            [3, 7, 9]])

x
np.percentile(x,75,axis=1)
#第一行是0、3、5，数和数之间一共有两个间隔，所以每个四分位间为2/4=0.5个数；
    #第一个数为0，75%百分位在1+0.5*3=2.5，就是在第二个数和第三个数之间，也就是3和5之间 ===> 3+0.5

#第二行是3、7、9,数和数之间一共有两个间隔，所以每个四分位间为2/4=0.5个数；
    #第一个数为3，75%百分位在1+0.5*3=2.5，就是在第二个数和第三个数之间，也就是7和9之间 ===> 7+0.5
```

Out[71]:
```
array([[5, 0, 3],
       [3, 7, 9]])
```

Out[71]:
```
array([4., 8.])
```

In [78]:
```
x=np.array([[5, 0, 3],
            [3, 7, 9],
            [6, 8, 6],
            [9, 1, 2]])

np.argmax(x)
np.argmax(x,axis=0)  #同一列中的值，最大值的行索引
np.argmax(x,axis=1)  #同一行中的值，最大值的列索引
```

Out[78]: 5

Out[78]: array([3, 2, 1], dtype=int64)

Out[78]: array([0, 2, 1, 0], dtype=int64)

2.2 Numpy聚合函数使用场景

In [79]:
```
import pandas as pd
grade=pd.read_csv(r"student_grade.txt",sep='\t')
grade.head(5)
```

Out[79]:

	姓名	语文	数学	英语	总分	班名次
0	杨璐	131	143	144	418	1
1	王雪	131	135	144	410	2
2	韩林霖	127	139	142	408	3
3	沙龙逸	123	148	136	407	4
4	李鉴学	126	135	140	401	5

In [81]:
```
np.mean(grade['语文'])
np.mean(grade['数学'])
np.mean(grade['英语'])
```

Out[81]: 108.0

Out[81]: 102.82352941176471

Out[81]: 109.0

3 Numpy的快速排序

3.1 np.sort()

- 语法：a.sort()
- 作用：就地对数组进行排序

In [19]:
```
np.random.seed(0)
a = np.random.randint(7,size=(1,10))
a
```

Out[19]:
```
array([[4, 5, 0, 3, 3, 3, 1, 3, 5, 2]])
```

In [84]:
```
a.sort()
a
```

Out[84]:
```
array([[0, 1, 2, 3, 3, 3, 3, 4, 5, 5]])
```

In [85]:
```
#如果是降序排列
a=-np.sort(-a)
a
```

Out[85]:
```
array([[5, 5, 4, 3, 3, 3, 3, 2, 1, 0]])
```

如果是多维数组排序

In [88]:
```
np.random.seed(0)
arr = np.random.randn(3, 3)
arr

arr.sort(axis=0) #如果arr.sort()内参数都不写的话，默认最后一个轴排（列轴）
arr
```

Out[88]:
```
array([[ 1.76405235,  0.40015721,  0.97873798],
       [ 2.2408932 ,  1.86755799, -0.97727788],
       [ 0.95008842, -0.15135721, -0.10321885]])
```

Out[88]:
```
array([[ 0.95008842, -0.15135721, -0.97727788],
       [ 1.76405235,  0.40015721, -0.10321885],
       [ 2.2408932 ,  1.86755799,  0.97873798]])
```

3.2 np.argsort()

返回将对此数组进行排序的索引。

In [97]:
```
np.random.seed(0)
a = np.random.randn(3, 3)
a

a.sort(axis=0)
a
```

Out[97]:
```
array([[ 1.76405235,  0.40015721,  0.97873798],
       [ 2.2408932 ,  1.86755799, -0.97727788],
       [ 0.95008842, -0.15135721, -0.10321885]])
```

Out[97]:
```
array([[ 0.95008842, -0.15135721, -0.97727788],
       [ 1.76405235,  0.40015721, -0.10321885],
       [ 2.2408932 ,  1.86755799,  0.97873798]])
```

```
array([[0 1.76405235,  0.40015721,  0.97873798],
       [1 2.2408932 ,  1.86755799, -0.97727788],
       [2 0.95008842, -0.15135721, -0.10321885]])

array([[2 0.95008842, -0.15135721, -0.97727788],
       [0 1.76405235,  0.40015721, -0.10321885],
       [1 2.2408932 ,  1.86755799,  0.97873798]])
```

In [99]:
```
np.random.seed(0)
a = np.random.randn(3, 3)

a.argsort(axis=0)  #按照第一个轴进行排列，但是显示的是其行索引
```

Out[99]:
```
array([[2, 2, 1],
       [0, 0, 2],
       [1, 1, 0]], dtype=int64)
```

同样地，如果要降序：

In [46]:
```
np.random.seed(0)
arr = np.random.randn(3, 3)
arr=np.argsort(-arr,0)

arr
```

Out[46]:
```
array([[1, 1, 0],
       [0, 0, 2],
       [2, 2, 1]], dtype=int64)
```

4 唯一化和集合逻辑

方法	说明
unique(x)	计算x中的唯一元素，并返回有序结果。
intersect1d(x,y)	计算x和y的公共元素，并返回有序结果。
union1d（x,y)	计算x和y的并集，并返回有序结果。
in1d(x,y)	得到一个表示“x的元素是否包含于y”的布尔型数组。
setdiff1d(x,y)	集合的差，即元素在x中且不在y中。
setxor1d(x,y)	集合的对称性，即存在于一个数组中但不同时存在于两个数组中的元素。

In [107]:
```
ints = np.array([1,2,3,4,2,4,3,5])
ints

np.unique(ints)
```

Out[107]:
```
array([1, 2, 3, 4, 2, 4, 3, 5])
```

Out[107]:
```
array([1, 2, 3, 4, 5])
```

In [108]:
```
x=np.array([3,4,5,6,7,8])
y=np.array([6,4,2,6,2,1])

np.intersect1d(x,y)       #求交集
```

Out[108]:
```
array([4, 6])
```

In [109]:
```
x=np.array([[3,4,5,6,7,8],[3,4,5,6,7,8],[11,13,31,14,51,51]])
y=np.array([[6,4,2,6,2,1],[3,4,5,6,7,8],[11,13,31,14,15,15]])

np.intersect1d(x,y)
```

Out[109]:
```
array([ 3,  4,  5,  6,  7,  8, 11, 13, 14, 31])
```

In [110]:
```
x=np.array([3,4,5,6,7,8])
y=np.array([6,4,2,6,2,1])

np.union1d(x,y)       #求并集（自带去重功能）
```

Out[110]:
```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

In [111]:
```
x=np.array([3,4,5,6,7,8])
y=np.array([6,4,2,6,2,1])

np.in1d(x,y)       #得到一个表示“x的元素是否包含于y”的布尔型数组
```

Out[111]:
```
array([False,  True, False,  True, False, False])
```

In [112]:
```
x=np.array([3,4,5,6,7,8])
y=np.array([6,4,2,6,2,1])

np.setdiff1d(x,y)       #差集，即元素在x中且不在y中
```

Out[112]:
```
array([3, 5, 7, 8])
```

In [113]:
```
x=np.array([3,4,5,6,7,8])
y=np.array([6,4,2,6,2,1])
np.setxor1d(x,y)       #存在于一个数组中但不同时存在于两个数组中的元素，并集减交集
```

Out[113]:
```
array([1, 2, 3, 5, 7, 8])
```

5 练习

如何提取数组a中大于2，小于7的数据?

In [40]:
```
np.random.seed(0)
a=np.random.randint(10,size=(3,3))
a
```

Out[40]:
```
array([[5, 0, 3],
       [3, 7, 9],
       [3, 5, 2]])
```

In [ ]:
```

```