1.Numpy简介

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

NumPy 的前身 Numeric 最早是由 Jim Hugunin 与其它协作者共同开发，2005 年，Travis Oliphant 在 Numeric 中结合了另一个同性质的程序库 Numarray 的特色，并加入了其它扩展而开发了 NumPy。NumPy 为开放源代码并且由许多协作者共同维护开发。

NumPy 是一个运行速度非常快的数学库，主要用于数组计算，包含：

- 一个强大的N维数组对象 ndarray
- 广播功能函数
- 整合 C/C++/Fortran 代码的工具
- 线性代数、傅里叶变换、随机数生成等功能

> NumPy 通常与 SciPy（Scientific Python）和 Matplotlib（绘图库）一起使用， 这种组合广泛用于替代 MatLab，是一个强大的科学计算环境，有助于我们通过 Python 学习数据科学或者机器学习。
>
> SciPy 是一个开源的 Python 算法库和数学工具包。
>
> SciPy 包含的模块有最优化、线性代数、积分、插值、特殊函数、快速傅里叶变换、信号处理和图像处理、常微分方程求解和其他科学与工程中常用的计算。
>
> Matplotlib 是 Python 编程语言及其数值数学扩展包 NumPy 的可视化操作界面。它为利用通用的图形用户界面工具包，如 Tkinter，wxPython，Qt 或 GTK+ 向应用程序嵌入式绘图提供了应用程序接口（API）。

2.Numpy安装

1. windows

```
pip3 install numpy scipy matplotlib -i https://pypi.tuna.tsinghua.edu.cn/simple
```

2. linux

```
sudo apt-get install python3-numpy python3-scipy python3-matplotlib ipython ipython-notebook python-pandas python-sympy python-nose
```

3. mac os

```
pip3 install numpy scipy matplotlib -i https://pypi.tuna.tsinghua.edu.cn/simple
```

3.Numpy-Ndarray

NumPy 最重要的一个特点是其 N 维数组对象 ndarray(N-dimensional array)，它是一系列同类型数据的集合，以0下标为开始进行集合中元素的索引。

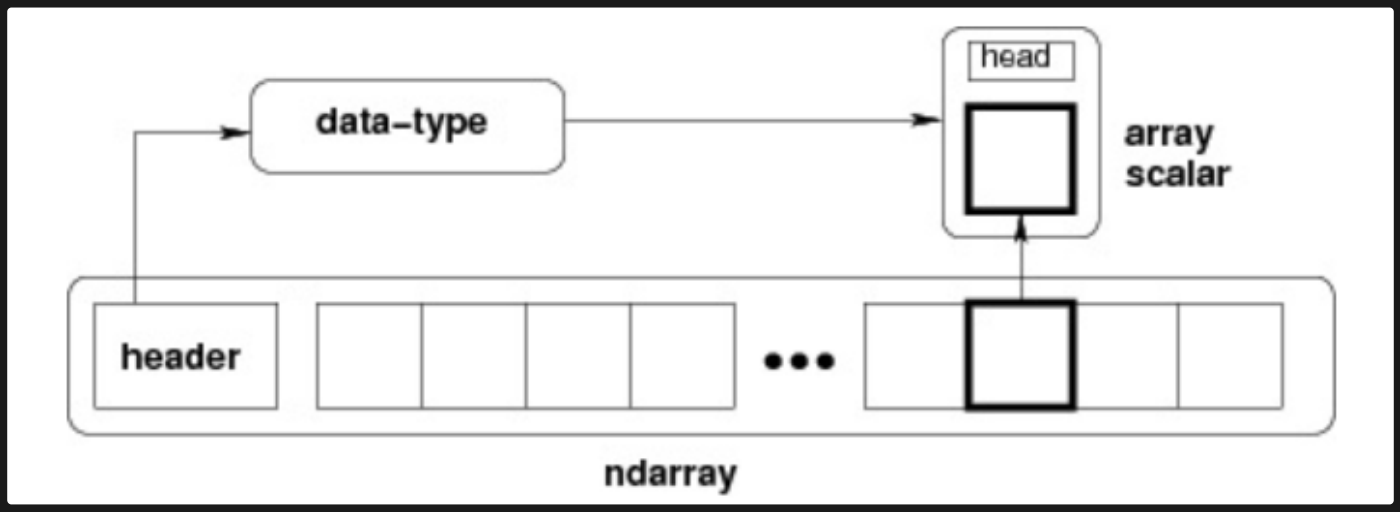
ndarray 对象是用于存放同类型元素的多维数组。

ndarray 中的每个元素在内存中都有相同存储大小的区域。

ndarray 内部由以下内容组成：

- 一个指向数据（内存或内存映射文件中的一块数据）的指针。
- 数据类型或 dtype，描述在数组中的固定大小值的格子。
- 一个表示数组形状（shape）的元组，表示各维度大小的元组。
- 一个跨度元组（stride），其中的整数指的是为了前进到当前维度下一个元素需要"跨过"的字节数。

ndarray 的内部结构：



跨度可以是负数，这样会使数组在内存中后向移动，切片中 **obj[::-1]** 或 **obj[:,::-1]** 就是如此。

创建一个 ndarray 只需调用 NumPy 的 array 函数即可：

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

名称	描述
object	数组或嵌套的数列
dtype	数组元素的数据类型，可选
copy	对象是否需要复制，可选
order	创建数组的样式，C为行方向，F为列方向，A为任意方向（默认）
subok	默认返回一个与基类类型一致的数组
ndmin	指定生成数组的最小维度

3.1numpy创建一维数组

```
# 导入numpy库，命名为np
import numpy as np

# 使用array函数插件数组内部存储列表
lst = np.array([1, 2, 3])
print(lst) # 将列表中的成员打印到终端上
```

3.2numpy创建二维数组

```
import numpy as np

# 将两个列表作为参数创建二维数组
arr = np.array([[1,2,3,4],[5,6,7,8]])
print(arr)
```

3.3numpy数组成员的访问

可以通过索引来访问数组中的元素。索引从 0 开始，可以使用负数索引从数组末尾开始访问元素。例如：

```
import numpy as np

arr = np.array([1,2,3,4,5])
print(arr)
print(arr[0])
print(arr[-1])
```

3.4numpy指定最小维度

```
import numpy as np

arr = np.array([1,2,3,4,5].ndmin=2)
print(arr) #把数组构造成二维的格式
```

3.5numpy数组属性

NumPy 数组对象具有多个属性，包括形状（shape）、维度（ndim）、元素类型（dtype）等。可以通过访问这些属性来获取数组的详细信息

```
import numpy as np

arr = np.array([[1,2],[3,4]],dtype=complex)
print(arr.shape)  # 输出数组的形状
print(arr.ndim)   # 输出数组的维度
print(arr.dtype)  # 输出数组的元素类型
```

3.6numpy数组运算

NumPy 数组支持各种数学运算和函数操作，如加法、乘法、平方根、指数等。这些操作可以逐元素地应用于数组。

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

print(arr1 + arr2)  # [5 7 9]

print(arr1 - arr2)  # [-3 -3 -3]

print(arr1 * arr2)  # [ 4 10 18]

print(arr1 / arr2)  # [0.25 0.4  0.5 ]
```

```
print(np.sqrt(arr1))  # [1.         1.41421356 1.73205081]

print(np.exp(arr1))  # [ 2.71828183  7.3890561  20.08553692]
```

4.Numpy数据类型

numpy 支持的数据类型比 Python 内置的类型要多很多，基本上可以和 C 语言的数据类型对应上，其中部分类型对应为 Python 内置的类型。下表列举了常用 NumPy 基本类型。

名称	描述
bool_	布尔型数据类型（True 或者 False）
int_	默认的整数类型（类似于 C 语言中的 long，int32 或 int64）
intc	与 C 的 int 类型一样，一般是 int32 或 int 64
intp	用于索引的整数类型（类似于 C 的 ssize_t，一般情况下仍然是 int32 或 int64）
int8	字节（-128 to 127）
int16	整数（-32768 to 32767）
int32	整数（-2147483648 to 2147483647）
int64	整数（-9223372036854775808 to 9223372036854775807）
uint8	无符号整数（0 to 255）
uint16	无符号整数（0 to 65535）
uint32	无符号整数（0 to 4294967295）
uint64	无符号整数（0 to 18446744073709551615）
float_	float64 类型的简写
float16	半精度浮点数，包括：1 个符号位，5 个指数位，10 个尾数位
float32	单精度浮点数，包括：1 个符号位，8 个指数位，23 个尾数位
float64	双精度浮点数，包括：1 个符号位，11 个指数位，52 个尾数位
complex_	complex128 类型的简写，即 128 位复数
complex64	复数，表示双 32 位浮点数（实数部分和虚数部分）
complex128	复数，表示双 64 位浮点数（实数部分和虚数部分）

numpy 的数值类型实际上是 dtype 对象的实例，并对应唯一的字符，包括 np.bool_，np.int32，np.float32，等等。

4.1数据类型对象（dtype）

数据类型对象（numpy.dtype 类的实例）用来描述与数组对应的内存区域是如何使用，它描述了数据的以下几个方面：：

- 数据的类型（整数，浮点数或者 Python 对象）
- 数据的大小（例如， 整数使用多少个字节存储）
- 数据的字节顺序（小端法或大端法）
- 在结构化类型的情况下，字段的名称、每个字段的数据类型和每个字段所取的内存块的部分
- 如果数据类型是子数组，那么它的形状和数据类型是什么。

字节顺序是通过对数据类型预先设定 < 或 > 来决定的。 < 意味着小端法(最小值存储在最小的地址，即低位组放在最前面)。> 意味着大端法(最重要的字节存储在最小的地址，即高位组放在最前面)。

dtype 对象是使用以下语法构造的：

```
numpy.dtype(object, align, copy)
```

- object - 要转换为的数据类型对象
- align - 如果为 true，填充字段使其类似 C 的结构体。
- copy - 复制 dtype 对象 ，如果为 false，则是对内置数据类型对象的引用

4.2数据类型实例

4.2.1标量类型

```
import numpy as np
# 使用标量类型
dt = np.dtype(np.int32)
print(dt) # int32
```

4.2.2四种常用的类型

```
import numpy as np
# i1    i2    i4    i8   f4      f8
# int8 int16 int32 int64 float32 float64
dt = np.dtype('i8')
print(dt) #int64
```

4.2.3字节序的实例

```
import numpy as np
# 小端字节序
dt = np.dtype('<i4')
print(dt)
```

4.2.4结构化类型

```
import numpy as np
#声明的结构化类型是name可以有20个字符，age是一个int32类型的整数
a = np.array([("zhangsan", 20), ("lisi", 21), ("wangwu", 18)], np.dtype([('name',
'S20'), ('age', 'i4')]))
print(a)
```

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
b	布尔型
i	(有符号) 整型
u	无符号整型 integer
f	浮点型
c	复数浮点型
m	timedelta（时间间隔）
M	datetime（日期时间）
O	(Python) 对象
S, a	(byte-)字符串
U	Unicode
V	原始数据 (void)

5.数组的属性

NumPy 数组的维数称为秩（rank），秩就是轴的数量，即数组的维度，一维数组的秩为 1，二维数组的秩为 2，以此类推。

在 NumPy中，每一个线性的数组称为是一个轴（axis），也就是维度（dimensions）。比如说，二维数组相当于是两个一维数组，其中第一个一维数组中每个元素又是一个一维数组。所以一维数组就是 NumPy 中的轴（axis），第一个轴相当于是底层数组，第二个轴是底层数组里的数组。而轴的数量—秩，就是数组的维数。

很多时候可以声明 axis。axis=0，表示沿着第 0 轴进行操作，即对每一列进行操作；axis=1，表示沿着第1轴进行操作，即对每一行进行操作。

NumPy 的数组中比较重要 ndarray 对象属性有：

属性	说明
ndarray.ndim	秩，即轴的数量或维度的数量
ndarray.shape	数组的维度，对于矩阵，n 行 m 列
ndarray.size	数组元素的总个数，相当于 .shape 中 n*m 的值
ndarray.dtype	ndarray 对象的元素类型
ndarray.itemsize	ndarray 对象中每个元素的大小，以字节为单位
ndarray.flags	ndarray 对象的内存信息
ndarray.real	ndarray元素的实部
ndarray.imag	ndarray 元素的虚部
ndarray.data	包含实际数组元素的缓冲区，由于一般通过数组的索引获取元素，所以通常不需要使用这个属性。

5.1数组属性实例

```
import numpy as np
# 代码格式化：command+option+l
arr = np.array([[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12]])
print(arr)

print(arr.ndim)  # 打印数组的维度 2
arr = arr.reshape(2, 3, 2)
print(arr)
print(arr.ndim)
print(arr.shape)
print(arr.size)

print(arr.flags)
```

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

属性	描述
C_CONTIGUOUS (C)	数据是在一个单一的C风格的连续段中
F_CONTIGUOUS (F)	数据是在一个单一的Fortran风格的连续段中
OWNDATA (O)	数组拥有它所使用的内存或从另一个对象中借用它
WRITEABLE (W)	数据区域可以被写入，将该值设置为 False，则数据为只读
ALIGNED (A)	数据和所有元素都适当地对齐到硬件上
UPDATEIFCOPY (U)	这个数组是其它数组的一个副本，当这个数组被释放时，原数组的内容将被更新

6.Numpy创建数组

ndarray 数组除了可以使用底层 ndarray 构造器来创建外，也可以通过以下几种方式来创建。

6.1numpy.empty

numpy.empty 方法用来创建一个指定形状（shape）、数据类型（dtype）且未初始化的数组：

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

参数说明：

参数	描述
shape	数组形状
dtype	数据类型，可选
order	有"C"和"F"两个选项,分别代表，行优先和列优先，在计算机内存中的存储元素的顺序。

下面是一个创建空数组的实例：

实例

```
import numpy as np
x = np.empty([3,2], dtype = int)
print (x)
```

输出结果为：

```
[[ 6917529027641081856  5764616291768666155]
 [ 6917529027641081859 -5764598754299804209]
 [          4497473538      844429428932120]]
```

注意 - 数组元素为随机值，因为它们未初始化。

6.2numpy.zeros

创建指定大小的数组，数组元素以 0 来填充：

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

参数说明：

参数	描述
shape	数组形状
dtype	数据类型，可选
order	'C' 用于 C 的行数组，或者 'F' 用于 FORTRAN 的列数组

实例

```
import numpy as np
# 默认为浮点数
x = np.zeros(5)
print(x)
# 设置类型为整数
y = np.zeros((5,), dtype = int)
print(y)
# 自定义类型
z = np.zeros((2,2), dtype = [('x', 'i4'), ('y', 'i4')])
print(z)
```

输出结果为：

```
[0. 0. 0. 0. 0.]
[0 0 0 0 0]
[[(0, 0) (0, 0)]
 [(0, 0) (0, 0)]]
```

6.3numpy.ones

创建指定形状的数组，数组元素以 1 来填充：

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

参数说明：

参数	描述
shape	数组形状
dtype	数据类型，可选
order	'C' 用于 C 的行数组，或者 'F' 用于 FORTRAN 的列数组

实例

```
import numpy as np
# 默认为浮点数
x = np.ones(5)
print(x)
# 自定义类型
x = np.ones([2,2], dtype = int)
print(x)
```

输出结果为：

```
[1. 1. 1. 1. 1.]
[[1 1]
 [1 1]]
```

6.4numpy.zeros_like

numpy.zeros_like 用于创建一个与给定数组具有相同形状的数组，数组元素以 0 来填充。

numpy.zeros 和 numpy.zeros_like 都是用于创建一个指定形状的数组，其中所有元素都是 0。

它们之间的区别在于：numpy.zeros 可以直接指定要创建的数组的形状，而 numpy.zeros_like 则是创建一个与给定数组具有相同形状的数组。

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

参数说明：

参数	描述
a	给定要创建相同形状的数组
dtype	创建的数组的数据类型
order	数组在内存中的存储顺序，可选值为 'C'（按行优先）或 'F'（按列优先），默认为 'K'（保留输入数组的存储顺序）
subok	是否允许返回子类，如果为 True，则返回一个子类对象，否则返回一个与 a 数组具有相同数据类型和存储顺序的数组
shape	创建的数组的形状，如果不指定，则默认为 a 数组的形状。

创建一个与 arr 形状相同的，所有元素都为 0 的数组：

实例

```
import numpy as np
# 创建一个 3x3 的二维数组
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# 创建一个与 arr 形状相同的，所有元素都为 0 的数组
zeros_arr = np.zeros_like(arr)
print(zeros_arr)
```

输出结果为：

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

6.5numpy.ones_like

numpy.ones_like 用于创建一个与给定数组具有相同形状的数组，数组元素以 1 来填充。

numpy.ones 和 numpy.ones_like 都是用于创建一个指定形状的数组，其中所有元素都是 1。

它们之间的区别在于：numpy.ones 可以直接指定要创建的数组的形状，而 numpy.ones_like 则是创建一个与给定数组具有相同形状的数组。

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

参数说明：

参数	描述
a	给定要创建相同形状的数组
dtype	创建的数组的数据类型
order	数组在内存中的存储顺序，可选值为 'C'（按行优先）或 'F'（按列优先），默认为 'K'（保留输入数组的存储顺序）
subok	是否允许返回子类，如果为 True，则返回一个子类对象，否则返回一个与 a 数组具有相同数据类型和存储顺序的数组
shape	创建的数组的形状，如果不指定，则默认为 a 数组的形状。

创建一个与 arr 形状相同的，所有元素都为 1 的数组：

实例

```
import numpy as np
# 创建一个 3x3 的二维数组
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# 创建一个与 arr 形状相同的，所有元素都为 1 的数组
ones_arr = np.ones_like(arr)
print(ones_arr)
```

输出结果为：

```
[[1 1 1]
 [1 1 1]
 [1 1 1]]
```

6.5综合实例

```
import numpy as np

# empty
arr1 = np.empty([2, 3], dtype='i4')
arr1[0] = 1
arr1[1] = 2
print(arr1)

# zeros
```

```
arr2 = np.zeros([2, 3], dtype='i8')
print(arr2)
print(arr2.size)

# ones
arr3 = np.ones([2, 3], dtype='f4')
print(arr3)

# zeros_like
arr4 = np.zeros_like(arr3)
print(arr4)

# ones_likes
arr5 = np.ones_like(arr3)
print(arr5)
```

7.从已有数组中创建数组

7.1numpy.asarray

numpy.asarray 类似 numpy.array，但 numpy.asarray 参数只有三个，比 numpy.array 少两个。

```
numpy.asarray(a, dtype = None, order = None)
```

参数说明：

参数	描述
a	任意形式的输入参数，可以是，列表，列表的元组，元组，元组的元组，元组的列表，多维数组
dtype	数据类型，可选
order	可选，有"C"和"F"两个选项,分别代表，行优先和列优先，在计算机内存中的存储元素的顺序。

7.2numpy.frombuffer

numpy.frombuffer 用于实现动态数组。

numpy.frombuffer 接受 buffer 输入参数，以流的形式读入转化成 ndarray 对象。

```
numpy.frombuffer(buffer, dtype = float, count = -1, offset = 0)
```

> **注意**：buffer 是字符串的时候，Python3 默认 str 是 Unicode 类型，所以要转成 bytestring 在原 str 前加上 b。

参数说明：

参数	描述
buffer	可以是任意对象，会以流的形式读入。
dtype	返回数组的数据类型，可选
count	读取的数据数量，默认为-1，读取所有数据。
offset	读取的起始位置，默认为0。

7.3numpy.fromiter

numpy.fromiter 方法从可迭代对象中建立 ndarray 对象，返回一维数组。

```
numpy.fromiter(iterable, dtype, count=-1)
```

参数	描述
iterable	可迭代对象
dtype	返回数组的数据类型
count	读取的数据数量，默认为-1，读取所有数据

7.4实例如下

```
import numpy as np

x = [1, 2, 3, 4, 5, 6]
print(x)

arr = np.asarray(x, "i4")
arr = arr.reshape(2, 3)
print(arr)

s = b'hello world'  # 默认是Unicode变量，转为bytestring
arr1 = np.frombuffer(s, "S1")
print(arr1)

import numpy as np
```

```
# 使用 range 函数创建列表对象
lst = range(5)
it = iter(lst)
# 使用迭代器创建 ndarray
x = np.fromiter(it, dtype=float)
print(x)
```

8.NumPy从数值范围创建数组

8.1创建范围数组numpy.arange

numpy 包中的使用 arange 函数创建数值范围并返回 ndarray 对象，函数格式如下：

```
numpy.arange(start, stop, step, dtype)
```

根据 start 与 stop 指定的范围以及 step 设定的步长，生成一个 ndarray。

参数说明：

参数	描述
`start`	起始值，默认为 `0`
`stop`	终止值（不包含）
`step`	步长，默认为 `1`
`dtype`	返回 `ndarray` 的数据类型，如果没有提供，则会使用输入数据的类型。

```
import numpy as np

arr = np.arange(start=20, stop=50, step=5, dtype="f4")
print(arr)
```

8.2等差数列numpy.linspace

numpy.linspace 函数用于创建一个一维数组，数组是一个等差数列构成的，格式如下：

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

参数说明：

参数	描述
start	序列的起始值
stop	序列的终止值，如果 endpoint 为 true ，该值包含于数列中
num	要生成的等步长的样本数量，默认为 50
endpoint	该值为 true 时，数列中包含 stop 值，反之不包含，默认是True。
retstep	如果为 True 时，生成的数组中会显示间距，反之不显示。
dtype	ndarray 的数据类型

```
import numpy as np
# start:起始位置，end:结束位置  num从start到end平均分的份数
a = np.linspace(1, 10, 2)
print(a)
```

8.3等比数列numpy.logspace

numpy.logspace 函数用于创建一个于等比数列。格式如下：

```
np.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)
```

base 参数意思是取对数的时候 log 的下标。

参数	描述
start	序列的起始值为：base ** start （这里的**是幂运算）
stop	序列的终止值为：base ** stop。如果 endpoint 为 true ，该值包含于数列中
num	要生成的等步长的样本数量，默认为 50
endpoint	该值为 true 时，数列中中包含 stop 值，反之不包含，默认是True。
base	对数 log 的底数。
dtype	ndarray 的数据类型

```
import numpy as np

a1 = np.logspace(start=1, stop=2, num=5, base=10)
print(a1)
#结果如下:
#  10^1         10^1.25      10^1.5       10^1.75      10^2
#[ 10.          17.7827941   31.6227766   56.23413252 100.        ]
```

9.NumPy数组的切片和索引

9.1slice切片用法

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

ndarray 数组可以基于 0 - n 的下标进行索引，切片对象可以通过内置的 slice 函数，并设置 start, stop 及 step 参数进行，从原数组中切割出一个新数组。

```
import numpy as np
a = np.arange(10)
s = slice(2,7,2)   # 从索引 2 开始到索引 7 停止，间隔为2 print (a[s])
```

输出结果为:

```
[2  4  6]
```

9.2[start:stop:step]切片的用法

以上实例中，我们首先通过 arange() 函数创建 ndarray 对象。 然后，分别设置起始，终止和步长的参数为 2，7 和 2。我们也可以通过冒号分隔切片参数 **start:stop:step** 来进行切片操作:

```
import numpy as np
a = np.arange(10)
b = a[2:7:2]   # 从索引 2 开始到索引 7 停止，间隔为 2 print(b)
```

输出结果为:

```
[2  4  6]
```

冒号 : 的解释：如果只放置一个参数，如 **[2]**，将返回与该索引相对应的单个元素。如果为 **[2:]**，表示从该索引开始以后的所有项都将被提取。如果使用了两个参数，如 **[2:7]**，那么则提取两个索引(不包括停止索引)之间的项。

```
import numpy as np
a = np.arange(10)  # [0 1 2 3 4 5 6 7 8 9]
b = a[5]
print(b)
```

输出结果为:

```
5
```

```
import numpy as np
a = np.arange(10)
print(a[2:])
```

输出结果为：

```
[2  3  4  5  6  7  8  9]
```

```
import numpy as np
a = np.arange(10)  # [0 1 2 3 4 5 6 7 8 9]
print(a[2:5])
```

输出结果为：

```
[2  3  4]
```

9.3多维数组切片

多维数组同样适用上述索引提取方法：

```
import numpy as np
a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print(a) # 从某个索引处开始切割 print('从数组索引 a[1:] 处开始切割')
print(a[1:])
```

输出结果为：

```
[[1 2 3]
 [3 4 5]
 [4 5 6]]
从数组索引 a[1:] 处开始切割
[[3 4 5]
 [4 5 6]]
```

9.4[...,...]省略行或者列的用法

切片还可以包括省略号 …，来使选择元组的长度与数组的维度相同。 如果在行位置使用省略号，它将返回包含行中元素的 ndarray。

```
import numpy as np
a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print (a[...,1])   # 第2列元素
print (a[1,...])   # 第2行元素
print (a[...,1:])  # 第2列及剩下的所有元素
```

输出结果为:

```
[2 4 5]
[3 4 5]
[[2 3]
 [4 5]
 [5 6]]
```

10.NumPy 广播(Broadcast)

广播(Broadcast)是 numpy 对不同形状(shape)的数组进行数值计算的方式， 对数组的算术运算通常在相应的元素上进行。

如果两个数组 a 和 b 形状相同，即满足 **a.shape == b.shape**，那么 a*b 的结果就是 a 与 b 数组对应位相乘。这要求维数相同，且各维度的长度相同。

```
import numpy as np
a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
print (c)
```

输出结果为:

```
[ 10  40  90 160]
```

当运算中的 2 个数组的形状不同时，numpy 将自动触发广播机制。如:

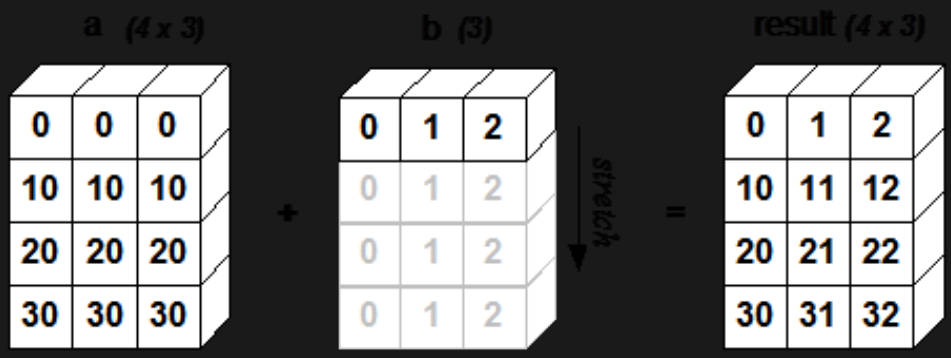
10.1广播运算方式

```
import numpy as np
a = np.array([[ 0, 0, 0],[10,10,10],[20,20,20],[30,30,30]])
b = np.array([0,1,2])
print(a + b)
```

输出结果为:

```
[[ 0  1  2]
 [10 11 12]
 [20 21 22]
 [30 31 32]]
```

下面的图片展示了数组 b 如何通过广播来与数组 a 兼容。



4x3 的二维数组与长为 3 的一维数组相加，等效于把数组 b 在二维上重复 4 次再运算：

```
import numpy as np
a = np.array([[ 0, 0, 0],[10,10,10],[20,20,20],[30,30,30]])
b = np.array([1,2,3])
bb = np.tile(b, (4, 1))  # 重复 b 的各个维度 print(a + bb)
```

输出结果为：

```
[[ 1  2  3]
 [11 12 13]
 [21 22 23]
 [31 32 33]]
```

10.2广播的规则

广播的规则：

- 让所有输入数组都向其中形状最长的数组看齐，形状中不足的部分都通过在前面加 1 补齐。
- 输出数组的形状是输入数组形状的各个维度上的最大值。
- 如果输入数组的某个维度和输出数组的对应维度的长度相同或者其长度为 1 时，这个数组能够用来计算，否则出错。
- 当输入数组的某个维度的长度为 1 时，沿着此维度运算时都用此维度上的第一组值。

简单理解：对两个数组，分别比较他们的每一个维度（若其中一个数组没有当前维度则忽略），满足：

- 数组拥有相同形状。
- 当前维度的值相等。
- 当前维度的值有一个是 1。

若条件不满足，抛出 **"ValueError: frames are not aligned"** 异常。

11.Numpy 数组操作

Numpy 中包含了一些函数用于处理数组，大概可分为以下几类：

- 修改数组形状
- 翻转数组

11.1修改数组形状

函数	描述
`reshape`	不改变数据的条件下修改形状
`flat`	数组元素迭代器
`flatten`	返回一份数组拷贝，对拷贝所做的修改不会影响原始数组
`ravel`	返回展开数组

11.1.1numpy.reshape

numpy.reshape 函数可以在不改变数据的条件下修改形状，格式如下：

```
numpy.reshape(arr, newshape, order='C')
```

- `arr`：要修改形状的数组
- `newshape`：整数或者整数数组，新的形状应当兼容原有形状
- order: 'C' -- 按行，'F' -- 按列，'A' -- 原顺序，'k' -- 元素在内存中的出现顺序。

```
import numpy as np
a = np.arange(8)
print ('原始数组: ')
print (a)
print ('\n')
b = a.reshape(4,2)
print ('修改后的数组: ')
print (b)
```

输出结果如下：

```
原始数组:
[0 1 2 3 4 5 6 7]

修改后的数组:
[[0 1]
 [2 3]
 [4 5]
 [6 7]]
```

11.2翻转数组

函数	描述
`transpose`	对换数组的维度
`ndarray.T`	和 `self.transpose()` 相同
`rollaxis`	向后滚动指定的轴
`swapaxes`	对换数组的两个轴

11.2.1对换数组维度numpy.transpose

numpy.transpose 函数用于对换数组的维度，格式如下：

```
numpy.transpose(arr, axes)
```

参数说明：

- `arr`：要操作的数组
- `axes`：整数列表，对应维度，通常所有维度都会对换。

```
import numpy as np
a = np.arange(12).reshape(3,4)
print ('原数组: ')
print (a )
print ('\n')
print ('对换数组: ')
print (np.transpose(a))
```

输出结果如下：

```
原数组:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]


对换数组:
[[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
```

11.2.2对换数组维度numpy.ndarray.T

numpy.ndarray.T 类似 numpy.transpose:

```
import numpy as np
a = np.arange(12).reshape(3,4)
print ('原数组: ')
print (a)
print ('\n')
print ('转置数组: ')
print (a.T)
```

输出结果如下:

```
原数组:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]


转置数组:
[[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
```

11.2.3滚动特定的轴numpy.rollaxis

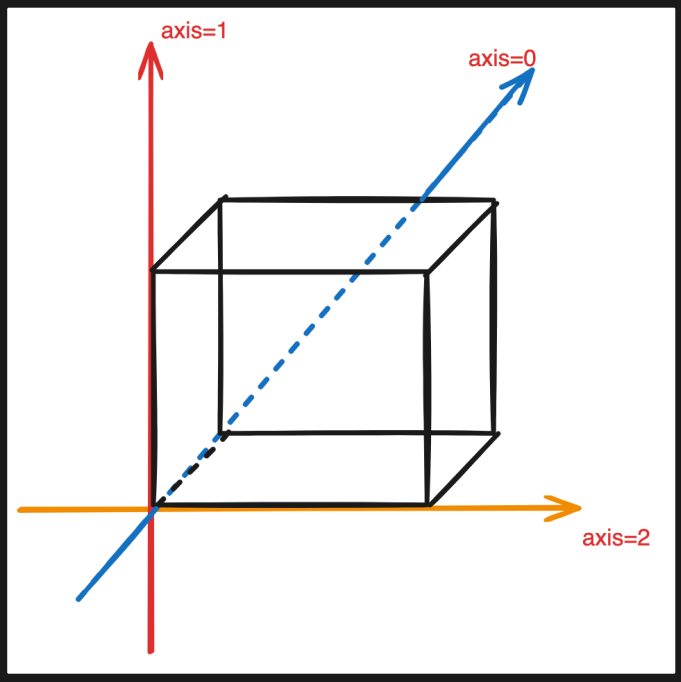
numpy.rollaxis 函数向后滚动特定的轴到一个特定位置，格式如下:

```
numpy.rollaxis(arr, axis, start)
```

参数说明:

- `arr` : 数组
- `axis` : 要向后滚动的轴，其它轴的相对位置不会改变
- `start` : 默认为零，表示完整的滚动。会滚动到特定位置。

先来探讨下三维数组的各个轴向：




```python
import numpy as np

# 创建一个三维数组
array_3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print("原始三维数组:")
print(array_3d)


# 解析0轴（沿着第一个维度的方向）
axis_0_data = array_3d[0]
print("\n0轴的数据:")
print(axis_0_data)

# 解析1轴（沿着第二个维度的方向）
axis_1_data = array_3d[:, 0, :]
print("\n1轴的数据:")
print(axis_1_data)

# 解析2轴（沿着第三个维度的方向）
axis_2_data = array_3d[:, :, 0]
print("\n2轴的数据:")
print(axis_2_data)
"""
原始三维数组:
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

```
0轴的数据:
[[1 2 3]
 [4 5 6]]

1轴的数据:
[[1 2 3]
 [7 8 9]]

2轴的数据:
[[ 1  4]
 [ 7 10]]
"""
```

```
import numpy as np

# 创建了三维的 ndarray
a = np.arange(8).reshape(2,2,2)

print ('原数组: ')
print (a)
print ('获取数组中一个值: ')
print(np.where(a==6))
print(a[1,1,0])  # 为 6
print ('\n')

 # 将轴 2 滚动到轴 0（宽度到深度）

print ('调用 rollaxis 函数: ')
b = np.rollaxis(a,2,0)
print (b)
# 查看元素 a[1,1,0]，即 6 的坐标，变成 [0, 1, 1]
# 最后一个 0 移动到最前面
print(np.where(b==6))
print ('\n')

# 将轴 2 滚动到轴 1:（宽度到高度）

print ('调用 rollaxis 函数: ')
c = np.rollaxis(a,2,1)
print (c)
# 查看元素 a[1,1,0]，即 6 的坐标，变成 [1, 0, 1]
# 最后的 0 和 它前面的 1 对换位置
print(np.where(c==6))
print ('\n')
```

输出结果如下:

```
原数组:
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
获取数组中一个值:
(array([1]), array([1]), array([0]))
6


调用 rollaxis 函数:
[[[0 2]
  [4 6]]

 [[1 3]
  [5 7]]]
(array([0]), array([1]), array([1]))


调用 rollaxis 函数:
[[[0 2]
  [1 3]]

 [[4 6]
  [5 7]]]
(array([1]), array([0]), array([1]))
```

11.2.4通过轴线交换数组numpy.swapaxes

numpy.swapaxes 函数用于交换数组的两个轴，格式如下：

```
numpy.swapaxes(arr, axis1, axis2)
```

- `arr`：输入的数组
- `axis1`：对应第一个轴的整数
- `axis2`：对应第二个轴的整数

```
import numpy as np
# 创建了三维的 ndarray
a = np.arange(8).reshape(2,2,2)
print ('原数组: ')
print (a)
print ('\n')
# 现在交换轴 0（深度方向）到轴 2（宽度方向）
print ('调用 swapaxes 函数后的数组: ')
print (np.swapaxes(a, 2, 0))
```

输出结果如下：

```
原数组:
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]


调用 swapaxes 函数后的数组:
[[[0 4]
  [2 6]]

 [[1 5]
  [3 7]]]
```

12.NumPy 矩阵库(Matrix)

NumPy 中包含了一个矩阵库 numpy.matlib，该模块中的函数返回的是一个矩阵，而不是 ndarray 对象。

一个 $m \times n$ 的矩阵是一个由 m 行（row） n 列（column）元素排列成的矩形阵列。

矩阵里的元素可以是数字、符号或数学式。以下是一个由 6 个数字元素构成的 2 行 3 列的矩阵：

$$\begin{bmatrix} 1 & 9 & -13 \\ 20 & 5 & -6 \end{bmatrix}$$

12.1转置矩阵

NumPy 中除了可以使用 numpy.transpose 函数来对换数组的维度，还可以使用 T 属性。。

例如有个 m 行 n 列的矩阵，使用 t() 函数就能转换为 n 行 m 列的矩阵。

A

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

```python
import numpy as np

a = np.arange(12).reshape(3,4)

print ('原数组: ')
print (a)
print ('\n')

print ('转置数组: ')
print (a.T)
```

输出结果如下:

```
原数组:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]


转置数组:
[[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
```

12.2空随机矩阵matlib.empty()

matlib.empty() 函数返回一个新的矩阵，语法格式为：

```
numpy.matlib.empty(shape, dtype, order)
```

参数说明：

- **shape**: 定义新矩阵形状的整数或整数元组
- **Dtype**: 可选，数据类型
- **order**: C（行序优先） 或者 F（列序优先）

```
import numpy.matlib
import numpy as np

print (np.matlib.empty((2,2)))
# 填充为随机数据
```

输出结果为：

```
[[-1.49166815e-154 -1.49166815e-154]
 [ 2.17371491e-313  2.52720790e-212]]
```

12.3值为0的矩阵numpy.matlib.zeros()

numpy.matlib.zeros() 函数创建一个以 0 填充的矩阵。

```
import numpy.matlib
import numpy as np
print (np.matlib.zeros((2,2)))
```

输出结果为：

```
[[0. 0.]
 [0. 0.]]
```

12.4值为1的矩阵numpy.matlib.ones()

numpy.matlib.ones()函数创建一个以 1 填充的矩阵。

```
import numpy.matlib
import numpy as np
print (np.matlib.ones((2,2)))
```

输出结果为:

```
[[1. 1.]
 [1. 1.]]
```

12.5对角线1的矩阵numpy.matlib.eye()

numpy.matlib.eye() 函数返回一个矩阵，对角线元素为 1，其他位置为零。

```
numpy.matlib.eye(n, M,k, dtype)
```

参数说明:

- **n**: 返回矩阵的行数
- **M**: 返回矩阵的列数，默认为 n
- **k**: 对角线的索引
- **dtype**: 数据类型

```
import numpy.matlib
import numpy as np
print (np.matlib.eye(n =  3, M =  4, k =  0, dtype =  float))
```

输出结果为:

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
```

12.6方单位矩阵numpy.matlib.identity()

numpy.matlib.identity() 函数返回给定大小的单位矩阵。

单位矩阵是个方阵，从左上角到右下角的对角线（称为主对角线）上的元素均为 1，除此以外全都为 0。

$$I_1 = [1],\ I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ \cdots,\ I_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

```
import numpy.matlib
import numpy as np
# 大小为 5, 类型位浮点型
print (np.matlib.identity(5, dtype =  float))
```

输出结果为：

```
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]]
```

12.7随机数矩阵numpy.matlib.rand()

numpy.matlib.rand() 函数创建一个给定大小的矩阵，数据是随机填充的。

```
import numpy.matlib
import numpy as np
print (np.matlib.rand(3,3))
```

输出结果为：

```
[[0.23966718 0.16147628 0.14162   ]
 [0.28379085 0.59934741 0.62985825]
 [0.99527238 0.11137883 0.41105367]]
```

矩阵总是二维的，而 ndarray 是一个 n 维数组。 两个对象都是可互换的。

```
import numpy.matlib
import numpy as np
i = np.matrix('1,2;3,4')
print (i)
```

输出结果为：

```
[[1  2]
 [3  4]]
```

```
import numpy.matlib
import numpy as np
j = np.asarray(i)
print (j)
```

输出结果为：

```
[[1  2]
 [3  4]]
```

```
import numpy.matlib
import numpy as np
k = np.asmatrix (j)
print (k)
```

输出结果为：

```
[[1  2]
 [3  4]]
```

13.NumPy线性代数

NumPy 提供了线性代数函数库 **linalg**，该库包含了线性代数所需的所有功能，可以看看下面的说明：

函数	描述
`dot`	两个数组的点积，即元素对应相乘。
`vdot`	两个向量的点积
`inner`	两个数组的内积
`matmul`	两个数组的矩阵积
`determinant`	数组的行列式
`solve`	求解线性矩阵方程
`inv`	计算矩阵的乘法逆矩阵

13.1点积|乘积|乘积和numpy.dot()

numpy.dot()

对于两个一维的数组，计算的是这两个数组对应下标元素的乘积和(数学上称之为 `向量点积`)；

对于二维数组，计算的是两个数组的矩阵乘积；

对于多维数组，它的通用计算公式如下，即结果数组中的每个元素都是：数组a的最后一维上的所有元素与数组b的倒数第二位上的所有元素的乘积和： **dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])**。

```
numpy.dot(a, b, out=None)
```

参数说明：

- **a** : ndarray 数组
- **b** : ndarray 数组
- **out** : ndarray, 可选，用来保存dot()的计算结果

```
import numpy.matlib
import numpy as np
a = np.array([[1,2],[3,4]])
b = np.array([[11,12],[13,14]])
print(np.dot(a,b))
```

输出结果为：

```
[[37  40]
 [85  92]]
```

计算式为：

```
[[1*11+2*13, 1*12+2*14],[3*11+4*13, 3*12+4*14]]
```

13.2点积numpy.vdot()

numpy.vdot() 函数是两个向量的点积。 如果第一个参数是复数，那么它的共轭复数会用于计算。 如果参数是多维数组，它会被展开。

```
import numpy as np
a = np.array([[1,2],[3,4]])
b = np.array([[11,12],[13,14]])
# vdot 将数组展开计算内积
print (np.vdot(a,b))
```

输出结果为：

```
130
```

计算式为：

```
1*11 + 2*12 + 3*13 + 4*14 = 130
```

13.3n向量内积umpy.inner()

numpy.inner() 函数返回一维数组的向量内积。对于更高的维度，它返回最后一个轴上的和的乘积。

```
import numpy as np
print (np.inner(np.array([1,2,3]),np.array([0,1,0])))
# 等价于 1*0+2*1+3*0
```

输出结果为：

```
2
```

多维数组实例

```
import numpy as np
a = np.array([[1,2], [3,4]])
print ('数组 a: ')
print (a)
b = np.array([[11, 12], [13, 14]])
print ('数组 b: ')
print (b)
print ('内积: ')
print (np.inner(a,b))
```

输出结果为：

```
数组 a:
[[1 2]
 [3 4]]
数组 b:
[[11 12]
 [13 14]]
内积:
[[35 41]
 [81 95]]
```

内积计算式为：

```
1*11+2*12, 1*13+2*14
3*11+4*12, 3*13+4*14
```

13.4矩阵乘积numpy.matmul

numpy.matmul 函数返回两个数组的矩阵乘积。 虽然它返回二维数组的正常乘积，但如果任一参数的维数大于2，则将其视为存在于最后两个索引的矩阵的栈，并进行相应广播。另一方面，如果任一参数是一维数组，则通过在其维度上附加 1 来将其提升为矩阵，并在乘法之后被去除。

对于二维数组，它就是矩阵乘法：

实例

```
import numpy.matlib
import numpy as np
a = [[1,0],[0,1]]
b = [[4,1],[2,2]]
print (np.matmul(a,b))
```

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

我们可以计算它们的乘积 AB：

$$C = AB$$

其中，

$$C_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j}$$

因此，

$$C_{11} = 1 \times 5 + 2 \times 7 = 19$$

$$C_{12} = 1 \times 6 + 2 \times 8 = 22$$

$$C_{21} = 3 \times 5 + 4 \times 7 = 43$$

$$C_{22} = 3 \times 6 + 4 \times 8 = 50$$

输出结果为：

```
[[4  1]
 [2  2]]
```

二维和一维运算：

实例：

```
import numpy.matlib
import numpy as np
a = [[1,0],[0,1]]
# 因为数组维度不同，会自动触发广播，也就意味着b数组广播后是[[1,2],[1,2]]
b = [1,2]
print (np.matmul(a,b))
print (np.matmul(b,a))
```

输出结果为：

```
[1  2]
[1  2]
```

维度大于二的数组 ：

实例：

```
import numpy.matlib
import numpy as np
a = np.arange(8).reshape(2,2,2)
b = np.arange(4).reshape(2,2)
print (np.matmul(a,b))
```

输出结果为：

$$a = \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$$

现在，我们将这两个矩阵相乘，按照矩阵相乘的规则，结果矩阵的每个元素可以通过对应位置元素的乘积之和得到。

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 0*0+1*2 & 0*1+1*3 \\ 2*0+3*2 & 2*1+3*3 \\ 4*0+5*2 & 4*1+5*3 \\ 6*0+7*2 & 6*1+7*3 \end{bmatrix}$$

```
[[[ 2  3]
  [ 6 11]]

 [[10 19]
  [14 27]]]
```

13.5矩阵行列式numpy.linalg.det()

numpy.linalg.det() 函数计算输入矩阵的行列式。

行列式在线性代数中是非常有用的值。 它从方阵的对角元素计算。 对于 2×2 矩阵，它是左上和右下元素的乘积与其他两个的乘积的差。

换句话说，对于矩阵[[a，b]，[c，d]]，行列式计算为 ad-bc。 较大的方阵被认为是 2×2 矩阵的组合。

对于一个 2×2 的矩阵：

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

它的行列式计算公式为：

$$\det(A) = ad - bc$$

对于一个 3×3 的矩阵：

$$B = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

它的行列式计算公式为：

$$\det(B) = a(ei - fh) - b(di - fg) + c(dh - eg)$$

实例：

```
import numpy as np
a = np.array([[1,2], [3,4]])
print (np.linalg.det(a))
```

输出结果为：

```
-2.0
```

实例

```
import numpy as np
b = np.array([[6,1,1], [4, -2, 5], [2,8,7]])
print (b)
print (np.linalg.det(b))
print (6*(-2*7 - 5*8) - 1*(4*7 - 5*2) + 1*(4*8 - -2*2))
```

输出结果为：

```
[[ 6  1  1]
 [ 4 -2  5]
 [ 2  8  7]]
-306.0
-306
```

13.6线性方程的解numpy.linalg.solve()

numpy.linalg.solve() 函数给出了矩阵形式的**线性方程的解**。

考虑以下线性方程：

```
x + y + z = 6

2y + 5z = -4

2x + 5y - z = 27
```

可以使用矩阵表示为：

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 5 \\ 2 & 5 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 6 \\ -4 \\ 27 \end{bmatrix}$$

如果矩阵成为A、X和B，方程变为：

```
AX = B

或

X = A^(-1)B
```

```
import numpy as np

a = np.array([[1,1,1],[0,2,5],[2,5,-1]])
b = np.array([6,-4,27])

x = np.linalg.solve(a,b)
print(x)
```

结果如下：

```
[ 5.  3. -2.]
```

13.7逆矩阵numpy.linalg.inv()

numpy.linalg.inv() 函数计算矩阵的乘法逆矩阵。

逆矩阵（inverse matrix）：设A是数域上的一个n阶矩阵，若在相同数域上存在另一个n阶矩阵B，使得： AB=BA=E ，则我们称B是A的逆矩阵，而A则被称为可逆矩阵。注：E为单位矩阵。

实例

```
import numpy as np
x = np.array([[1,2],[3,4]])
y = np.linalg.inv(x)
print (x)
print (y)
print (np.dot(x,y))
```

输出结果为：

```
[[1 2]
 [3 4]]
[[-2.   1. ]
 [ 1.5 -0.5]]
[[1.0000000e+00 0.0000000e+00]
 [8.8817842e-16 1.0000000e+00]]
```

现在创建一个矩阵A的逆矩阵：

实例

```
import numpy as np
a = np.array([[1,1,1],[0,2,5],[2,5,-1]])
print ('数组 a: ')
print (a)
ainv = np.linalg.inv(a)
print ('a 的逆: ')
print (ainv)
print ('矩阵 b: ')
b = np.array([[6],[-4],[27]])
print (b)
print ('计算: A^(-1)B: ')
x = np.linalg.solve(a,b)
print (x) # 这就是线性方向 x = 5, y = 3, z = -2 的解
```

输出结果为：

```
数组 a:
[[ 1  1  1]
 [ 0  2  5]
 [ 2  5 -1]]
a 的逆:
[[ 1.28571429 -0.28571429 -0.14285714]
 [-0.47619048  0.14285714  0.23809524]
 [ 0.19047619  0.14285714 -0.0952381 ]]
矩阵 b:
[[ 6]
 [-4]
 [27]]
计算: A^(-1)B:
[[ 5.]
```

```
 [ 3.]
 [-2.]]
```

结果也可以使用以下函数获取：

```
x = np.dot(ainv,b)
```