# 100 Numpy Exercises

小小白

2019 年 9 月 11 日

首先，这是numpy-100中文版及部分题目的注解。然后是一些阅读与练习建议。64 题之前的题目都是 1 星和 2 星的，可以仔细阅读下。然后三星的题目就有些比较偏，可以尝试练习下用 *np.lookfor* 来锻炼“根据需求找函数”的技巧（文中有示例）。最后，翻译的太差了，能看还是去看原版吧 555

### 1. 导入 Numpy

```
In [1]: import numpy as np
```

### 2. 打印 Numpy 版本号及其配置

```
In [2]: np.__version__
```

```
Out[2]: '1.16.2'
```

```
In [3]: np.show_config()
```

```
mkl_info:
    libraries = ['mkl_rt', 'pthread']
    library_dirs = ['/home/shensir/anaconda3/lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['/home/shensir/anaconda3/include']
blas_mkl_info:
    libraries = ['mkl_rt', 'pthread']
    library_dirs = ['/home/shensir/anaconda3/lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['/home/shensir/anaconda3/include']
blas_opt_info:
    libraries = ['mkl_rt', 'pthread']
    library_dirs = ['/home/shensir/anaconda3/lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
```

```
    include_dirs = ['/home/shensir/anaconda3/include']
lapack_mkl_info:
    libraries = ['mkl_rt', 'pthread']
    library_dirs = ['/home/shensir/anaconda3/lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['/home/shensir/anaconda3/include']
lapack_opt_info:
    libraries = ['mkl_rt', 'pthread']
    library_dirs = ['/home/shensir/anaconda3/lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['/home/shensir/anaconda3/include']
```

**3. 创建一个长度 (size) 为 10 的向量 (vector)** 注意这里就是创建一个 `np.ndarray`

```
In [4]: z = np.zeros(10)
        z

Out[4]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

**4. 计算数组的内存大小**

```
In [5]: z = np.zeros((10, 10))
        print("%d bytes" % (z.size * z.itemsize))

800 bytes
```

注意这里就是简单的把总元素个数 `z.size`（100），乘上单个元素所占的内存 `z.itemsize`(8 bytes).

**5. 在命令行打印出 Numpy 中 add 函数的帮助文档信息** 这里主要是有关命令行调用 Python 的问题，我们可以在命令行从 `python --help` 开始，找到 `python -c` 符合我们的要求，所以这里只需要在命令行执行 `python -c "import numpy; numpy.info(numpy.add)"` 即可。就等同于在 Python 解释器中执行如下程序：

```
import numpy
numpy.info(numpy.add)
```

此外，针对 Numpy, 我们可以有很多种方式查看文档: 上面的 `np.info(np.add)`, 以及利用 help 的 `help(np.add)`, 和比较少用的 doc 方法的调用 `print(np.add.__doc__)`（输出和 `np.info` 一致）, 一般来说用 `np.info` 就可以了，也比较方便。如果使用 `IPython` 的话，可以直接 `np.add?` 回车

**6. 创建一个长度为 10 的向量，其第五个值为 1，其他为 0**

```
In [6]: z = np.zeros(10)
        z[4] = 1
        z

Out[6]: array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])
```

**7. 创建一个包含从 10 到 49 所有整数的向量**

```
In [7]: z = np.arange(10, 50)
        z

Out[7]: array([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, 35, 36, 37, 38, 39, 40, 41, 42, 43,
               44, 45, 46, 47, 48, 49])
```

注意 Python 也有内建函数 `range` 具有相似的功能，相对而言，Numpy 的 `arange` 由于使用了内存优化技术，其效率要高很多。我们可以做个简单的水平对比。例子来自 *Scipy Lecture Notes*

```
In [8]: %timeit [i**2 for i in range(1000)]

317 ţs ś 7.1 ţs per loop (mean ś std. dev. of 7 runs, 1000 loops each)


In [9]: %timeit a = np.arange(1000) ** 2

5.13 ţs ś 842 ns per loop (mean ś std. dev. of 7 runs, 100000 loops each)
```

**8. 反转一个向量（逆序）**

```
In [10]: z = np.arange(10)
         z

Out[10]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [11]: z = z[::-1]
         z

Out[11]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

**9. 创建一个 3x3 的矩阵，包含数字 0 到 8**

```
In [12]: z = np.arange(0, 9).reshape(3, 3)
         z

Out[12]: array([[0, 1, 2],
                [3, 4, 5],
                [6, 7, 8]])
```

这里可以在原题目的基础上进行拓展，上面的实现中 0 到 8 是可以看作以行为顺序，如我我们希望 0 到 8 以列为顺序排列呢?

```
In [13]: # 列为顺序
         z = np.arange(0, 9).reshape(3, 3).T
         z

Out[13]: array([[0, 3, 6],
                [1, 4, 7],
                [2, 5, 8]])
```

其实只需要把原来的矩阵转置就可以了:-)

**10. 找出 [1,2,0,0,4,0] 中非 0 数字的位置**

```
In [14]: z = np.array([1,2,0,0,4,0])
         z.nonzero()

Out[14]: (array([0, 1, 4]),)
```

**11. 创建 3x3 的单位矩阵**

```
In [15]: z = np.eye(3)
         z

Out[15]: array([[1., 0., 0.],
                [0., 1., 0.],
                [0., 0., 1.]])
```

**12. 创建 3x3x3 数组，以随机数字填充**

```
In [16]: z = np.random.random((3, 3, 3))
         z

Out[16]: array([[[0.22223755, 0.02993117, 0.4890893 ],
                 [0.0820244 , 0.67123686, 0.30827137],
                 [0.7508472 , 0.81738707, 0.84870083]],

                [[0.65084705, 0.40438861, 0.04736435],
                 [0.09258906, 0.50895712, 0.34483339],
                 [0.91449234, 0.16814904, 0.86294636]],

                [[0.56517313, 0.13952306, 0.76674818],
                 [0.04259055, 0.28120883, 0.13086924],
                 [0.11580096, 0.91740276, 0.24734521]]])
```

**13. 创建 10x10 数组，以随机数字填充，并找出其中的最大值和最小值**

```
In [17]: z = np.random.random((10, 10))
         z

Out[17]: array([[0.12752282, 0.79054209, 0.50129149, 0.58857169, 0.35450135,
                 0.39841021, 0.17807883, 0.88929081, 0.97780616, 0.773384  ],
                [0.90780362, 0.64876217, 0.0585608 , 0.26701752, 0.76980913,
                 0.28396342, 0.86735689, 0.0125925 , 0.40396858, 0.14115772],
                [0.8098699 , 0.61503945, 0.30412717, 0.09408111, 0.45769837,
                 0.47180697, 0.92817308, 0.68631218, 0.17004323, 0.48279816],
                [0.58715196, 0.2383897 , 0.28221602, 0.08891016, 0.16125275,
                 0.00269889, 0.20931989, 0.18024097, 0.11391687, 0.73651609],
                [0.25771839, 0.85886099, 0.13783206, 0.92036633, 0.47577829,
                 0.9711062 , 0.32103083, 0.70490514, 0.52713121, 0.19038581],
                [0.30706035, 0.67517319, 0.32025366, 0.08029852, 0.72571681,
                 0.07372837, 0.86243916, 0.25484642, 0.70585143, 0.08274462],
                [0.74795535, 0.8067741 , 0.89113355, 0.47931097, 0.10301097,
                 0.4847746 , 0.92899191, 0.12575855, 0.88174842, 0.47706755],
                [0.92802187, 0.66522839, 0.2899387 , 0.46113872, 0.28817592,
                 0.20630832, 0.39909759, 0.23957712, 0.98647748, 0.14215   ],
                [0.05693348, 0.35706197, 0.0287801 , 0.2528074 , 0.13481535,
```

```
       0.2227437 , 0.68933693, 0.19408625, 0.20216115, 0.99834648],
      [0.0363542 , 0.26831658, 0.50569523, 0.19128829, 0.53549785,
       0.49938433, 0.9457238 , 0.31700083, 0.19944951, 0.03825154]])
```

```
In [18]: z.max(), z.min()
```

```
Out[18]: (0.9983464750382925, 0.0026988893597477137)
```

**14. 创建长度为 10 的随机向量，并计算其均值**

```
In [19]: z = np.random.random((10))
         z
```

```
Out[19]: array([0.32286031, 0.59936123, 0.6659822 , 0.00112788, 0.71261922,
                0.07623688, 0.23427983, 0.70193797, 0.31707463, 0.6425674 ])
```

```
In [20]: z.mean()
```

```
Out[20]: 0.4274047541501854
```

**15. 创建一个二维数组，其边界值为 1，内部值为 0**

```
In [21]: z = np.ones((5, 5))
         z[1:-1, 1:-1] = 0
         z
```

```
Out[21]: array([[1., 1., 1., 1., 1.],
                [1., 0., 0., 0., 1.],
                [1., 0., 0., 0., 1.],
                [1., 0., 0., 0., 1.],
                [1., 1., 1., 1., 1.]])
```

**16. 将现有的数组（nxn）用 0 组成的边界包裹**

```
In [22]: z = np.ones((5, 5))
         z
```

```
Out[22]: array([[1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1.]])
```

```
In [23]: m = np.pad(z, (1, 1), mode='constant', constant_values=0)
         m

Out[23]: array([[0., 0., 0., 0., 0., 0., 0.],
                [0., 1., 1., 1., 1., 1., 0.],
                [0., 1., 1., 1., 1., 1., 0.],
                [0., 1., 1., 1., 1., 1., 0.],
                [0., 1., 1., 1., 1., 1., 0.],
                [0., 1., 1., 1., 1., 1., 0.],
                [0., 0., 0., 0., 0., 0., 0.]])
```

**17. 下列表达式的结果是什么**

```
0 * np.nan
np.nan == np.nan
np.inf > np.nan
np.nan - np.nan
np.nan in set([np.nan])
0.3 == 3 * 0.1
```

```
In [24]: 0 * np.nan

Out[24]: nan
```

有 np.nan 参与的算术操作返回均为 np.nan

```
In [25]: np.nan == np.nan

Out[25]: False
```

这里是合理的，比如我们从数据集读出两列数据全部是 np.nan，如果上面的表达式设计为返回 True，那么我们在完全不知道两列数据的情况下就判定二者是相等的，这显然是不合理的。所以这里返回的是 False.

```
In [26]: np.inf > np.nan

Out[26]: False
```

同样地，我们不能比较无穷大与缺失值的大小

```
In [27]: np.nan in set([np.nan])
```

```
Out[27]: True

In [28]: 0.3 == 3 * 0.1

Out[28]: False
```

由于浮点数（float）运算存在误差，我们不能直接比较其大小。Numpy 为我们提供了 np.allclose 函数来比较浮点数之间的近似相等。此外，此函数还支持 np.ndarray 的比较。

```
In [29]: np.allclose(0.3, 3 * 0.1)

Out[29]: True
```

**18. 创建一个 5x5 的矩阵，其中 1，2，3，4 正好在矩阵对角线元素下方**

```
In [30]: z = np.diag(np.arange(1, 5), k=-1)
         z

Out[30]: array([[0, 0, 0, 0, 0],
                [1, 0, 0, 0, 0],
                [0, 2, 0, 0, 0],
                [0, 0, 3, 0, 0],
                [0, 0, 0, 4, 0]])
```

**19. 创建一个 8x8 的矩阵，并用 0，1 标记为国际象棋棋盘的形式** 如下所示, 黑色部分标记为 1.

```
In [31]: z = np.zeros((8, 8))
         z[1::2, ::2] = 1  # 第 2, 4, 6, 8 行填充
         z[::2, 1::2] = 1  # 第 1,  3,  5,  7 行填充
         z

Out[31]: array([[0., 1., 0., 1., 0., 1., 0., 1.],
                [1., 0., 1., 0., 1., 0., 1., 0.],
                [0., 1., 0., 1., 0., 1., 0., 1.],
                [1., 0., 1., 0., 1., 0., 1., 0.],
                [0., 1., 0., 1., 0., 1., 0., 1.],
                [1., 0., 1., 0., 1., 0., 1., 0.],
                [0., 1., 0., 1., 0., 1., 0., 1.],
                [1., 0., 1., 0., 1., 0., 1., 0.]])
```

**20. 现有维度为 6x7x8 的数组，找出其中第 100 个元素的索引 (x, y, z)**

```
In [32]: print(np.unravel_index(99, (6, 7, 8)))
```

```
(1, 5, 3)
```

上面的是给出的答案，一开始我并不知道这个函数，采用了下面的方法，可以作为参考。

```
In [33]: z = np.arange(6*7*8).reshape(6, 7, 8)
         np.where(z == 99)
```

```
Out[33]: (array([1]), array([5]), array([3]))
```

这是通过 Numpy 找出来具体位置，但是具体计算的方法并未给出，这里简单解释下。

首先，我们可以形象第考虑“数组的维度越往后，对应数据的颗粒度越小”，也就是说，在上面的例子中，我们可以认为 6x7x8 的立方体是通过如下的方法来构建的：先将所有的一列值 (6 * 7 * 8) 排成一行，之后每 8 个组成一个“长条”，这样就有 6*7 个长条；之后将每 7 个长条，上下拼接，铺成一个平面；这样我们就有 6 个平面，将这 6 个平面堆起来，就得到了我们最终的“立方体”。

那么第 100 个元素又在哪里呢？为方便起见，我们从“颗粒度”大的开始，依次定位其位置。首先，可以知道每一层含有 7*8=56 个元素，所以由 `100 // 56 = 1` 得其位于第二层，对应到该维度得到索引就是 1，即返回的 `array[1]`。之后在第二层中继续定位，去除第一层的 56 个元素，这里还剩下 44 个。又由于平面为 7x8 的，所以由 `44 // 8 = 5` 得其位于第 6 行，对应该维度的索引是 5，即返回的 `array[5]`，最后剩下 4 个元素在新的一行，对应维度的索引为 3，即返回的 `array[3]`. 由此得到最终的索引为（1，5，3）

**21. 用 `tile` 函数创建一个 8x8 的棋盘**

```
In [34]: unit = np.array([[0, 1], [1, 0]])
         z = np.tile(unit, (4, 4))
         z
```

```
Out[34]: array([[0, 1, 0, 1, 0, 1, 0, 1],
                [1, 0, 1, 0, 1, 0, 1, 0],
                [0, 1, 0, 1, 0, 1, 0, 1],
                [1, 0, 1, 0, 1, 0, 1, 0],
                [0, 1, 0, 1, 0, 1, 0, 1],
                [1, 0, 1, 0, 1, 0, 1, 0],
                [0, 1, 0, 1, 0, 1, 0, 1],
                [1, 0, 1, 0, 1, 0, 1, 0]])
```

`tile` 的原意就是铺瓷砖，是其作用的一个形象的比喻，这里我们把 8x8 的棋盘划分为 4x4=16 块“瓷砖”（这里的 `unit`），之后将其平铺在一起即可。

**22. 标准化一个 5x5 的随机矩阵**

```
In [35]: z = np.random.random((5, 5))
         z = (z - z.mean()) / z.std()
         z

Out[35]: array([[-1.12827257, -0.50820625, -0.0323797 , -0.49744146,  1.47655254],
                [-1.86064942,  1.47067894, -1.07098905, -0.06903328,  1.39029289],
                [-0.25113084, -0.93021344,  1.81900584, -0.85037806, -0.51959092],
                [-0.83086235,  0.55036267,  1.1242602 , -0.09267453,  0.43763645],
                [-0.57602888,  1.27352951,  0.11596341, -1.3557155 ,  0.91528381]])
```

**23. 创建一个自定义的包含四个无符号字节（RGBA）的 dtype 来描述颜色**

```
In [36]: color = np.dtype([("r", np.ubyte, 1),
                           ("g", np.ubyte, 1),
                           ("b", np.ubyte, 1),
                           ("a", np.ubyte, 1)])
         color

Out[36]: dtype([('r', 'u1'), ('g', 'u1'), ('b', 'u1'), ('a', 'u1')])
```

**24. 5x3 的矩阵与 3x2 的矩阵相乘**

```
In [37]: z = np.dot(np.ones((5, 3)), np.ones((3, 2)))
         z

Out[37]: array([[3., 3.],
                [3., 3.],
                [3., 3.],
                [3., 3.],
                [3., 3.]])

In [38]: # 也可以使用操作符 @
         z = np.ones((5, 3)) @ np.ones((3, 2))
         z

Out[38]: array([[3., 3.],
                [3., 3.],
                [3., 3.],
                [3., 3.],
                [3., 3.]])
```

**25. 给定一个一维数组，将值在 3 和 8 之间的数字变为其负数**

```
In [39]: z = np.arange(10)
         z[(z > 3) & (z < 8)] *= -1
         z

Out[39]: array([ 0,  1,  2,  3, -4, -5, -6, -7,  8,  9])
```

**26. 下面脚本的输出是什么**

```
# Author: Jake VanderPlas

print(sum(range(5),-1))
from numpy import *
print(sum(range(5),-1))
```

```
In [40]: sum(range(5), -1)

Out[40]: 9
```

这里是使用 Python 内置的 `sum` 函数，它把所有的参数都当作求和的一部分相加，这里就是简单地将所有的数字相加，10 - 1 = 9

```
In [41]: np.sum(range(5), -1)

Out[41]: 10
```

这里使用的是 Numpy 中的 `np.sum`，这里的`-1` 并非待加的数字，而是另外一个参数的值，代表多维数组在求和时各个轴求和的顺序。具体可以 `help(np.sum)`

**27. z 是整数组成的向量，判断下列表达式是否正确**

```
In [42]: z = np.arange(10)
         z

Out[42]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [43]: # 1
         z ** z

Out[43]: array([        1,         1,         4,        27,       256,      3125,
                   46656,    823543,  16777216, 387420489])
```

前面提到过，对于数组之间的 ** 等算术运算，是元素一一对应进行运算的 (element-wise), 如这里 387420489，就等于 9**9

```
In [44]: # 2
         2 << z >> 2
```

```
Out[44]: array([  0,   1,   2,   4,   8,  16,  32,  64, 128, 256])
```

本质上进行两次移位运算，也就是等于 (2 << z) >> 2. 下面将其拆开来看。

```
In [45]: 2 << z
```

```
Out[45]: array([   2,    4,    8,   16,   32,   64,  128,  256,  512, 1024])
```

就是将 2 分别左移 0，1，2，...，9 位，得到的就是 2 << 0，2<<1，...，2<<9, 如下所示：

```
In [46]: part1 = [2 << i for i in range(10)]
         part1
```

```
Out[46]: [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

可以看到与 2 << z 的输出是一致的。

之后就是进行右移位操作，不同之处在于，这里是对于数组 2 << z 中的每个元素进行右移位，分别右移 2 个位置。每个数值右移 2 代表对每个值 x，取 x // 4, 对 part1 继续处理

```
In [47]: [i // 4 for i in part1]
```

```
Out[47]: [0, 1, 2, 4, 8, 16, 32, 64, 128, 256]
```

可以看到其与 2 << z >> 2 的输出是一致的。

```
In [48]: # 3
         z <- z
```

```
Out[48]: array([False, False, False, False, False, False, False, False, False,
                False])
```

这里涉及的主要是优先级的问题，随便找个操作符，如 <, 通过 help("<") 即可查看所有操作符的优先级，默认是从低优先级到高优先级。可以看到，-1 相比 < 具有更高的优先级，所以这里就等同于 z < (-z)，测试如下

```
In [49]: z < (-z)
```

```
Out[49]: array([False, False, False, False, False, False, False, False, False,
                False])

In [50]: # 4
         1j * z

Out[50]: array([0.+0.j, 0.+1.j, 0.+2.j, 0.+3.j, 0.+4.j, 0.+5.j, 0.+6.j, 0.+7.j,
                0.+8.j, 0.+9.j])
```

对复数的运算的支持

```
In [51]: # 5
         z/1/1

Out[51]: array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

也就是 (z/1)/1

```
In [181]: # 6
          z < z > z


          ---------------------------------------------------------------------------

          ValueError                                Traceback (most recent call last)

          <ipython-input-181-856b68e674c4> in <module>
            1 # 6
      ----> 2 z < z > z


          ValueError: The truth value of an array with more than one element is ambiguous. Use a
```

这里参考了下 Python 表达式的文档, 和stackoverflow 找到 >Formally, if a, b, c, ..., y, z are expressions and op1, op2, ..., opN are comparison operators, then a op1 b op2 c ... y opN z is equivalent to a op1 b and b op2 c and ... y opN z, except that each expression is evaluated at most once.

就是说，在一个表达式里面进行连续比较的时候，如 x < y <= z, 首先是符合语法的，其等同于 x < y adn y <=z, 只不过对于重复的元素（这里的 y）只估计一次。所以我们在 Python 原生的 list 中进行上述 z 的运算是可以正常返回的，代码如下。

```
In [53]: l1 = [1, 2]
         l1 < l1 > l1

Out[53]: False

In [54]: (l1 < l1) and (l1 > l1)

Out[54]: False
```

因为 l1 < l1 和 l1 > l1 全部是 False, 所以其 and 也是 False. 但是对于我们的 z，也就是 np.ndarray 类型的数据，情况就有所不同。

这里 z < z > z 依旧是估计为 z < z and z > z, 其中 z < z 与 z > z 都是可以正常返回的，且结果都是一个长度为 z.size() 的 array，元素全部是布尔值。

```
In [55]: z < z

Out[55]: array([False, False, False, False, False, False, False, False, False,
                False])

In [56]: z > z

Out[56]: array([False, False, False, False, False, False, False, False, False,
                False])
```

不可行的是两者之间的 and。因为在进行 and 操作时，Numpy 无法确切地知道形如 array([False, False, ...]) 的数组到底是估计为 False，还是 True，因为这里有两种方法来定义一个数组的布尔值：其一是 a l l，即所有的元素全是 T r u e 才判定为 T r u e，否则为 F a l s e；另外一种方法是 a n y，即只要数组中有一个 T r u e，我们就判定其为 T r u e，否则判定为 F a l s e. 正是这种不确定性使得 Numpy 报错，并建议使用 any 或者 all.

**28. 下列表达式的结果是什么**

```
np.array(0) / np.array(0)
np.array(0) // np.array(0)
np.array([np.nan]).astype(int).astype(float)

In [57]: np.array(0) / np.array(0)

/home/shensir/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: RuntimeWarning: in
  """Entry point for launching an IPython kernel.
```

```
Out[57]: nan
```

返回 nan 并带有警告说在进行真除 (true_divide) 的时候出现问题，即 0 做分母。

```
In [58]: np.array(0) // np.array(0)
```

```
/home/shensir/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: RuntimeWarning: di
  """Entry point for launching an IPython kernel.


Out[58]: 0
```

返回 0, 并带有警告说在进行地板除 (floor_divide) 的时候出现问题，即 0 做分母。

```
In [59]: np.array([np.nan]).astype(int).astype(float)

Out[59]: array([-9.22337204e+18])
```

**29. 舍入浮点数数组，使其尽可能远离 0** 即-0.3，-0.5，-0.6 等近似为-1，而非 0；0.3，0.5，0.6 等近似为 1，而非 0.

```
In [60]: z = np.random.uniform(-10, 10, 10)
         z

Out[60]: array([-7.76242024, -5.15644074, -5.19760008,  4.24343511,  3.25711345,
                 4.83702175,  4.92154317, -9.77526978,  0.45551806, -9.95335581])

In [61]: np.copysign(np.ceil(np.abs(z)), z)

Out[61]: array([ -8.,  -6.,  -6.,   5.,   4.,   5.,   5., -10.,   1., -10.])
```

**30. 找到两个数组中相同的元素** 我们首先考虑内置的函数，但是我们不知道是否有类似的函数，所以我们可以灵活使用 np.lookfor 来找出我们要的函数。

```
In [63]: np.lookfor("common values")

Search results for 'common values'
----------------------------------
numpy.ma.intersect1d
    Returns the unique elements common to both arrays.
numpy.gcd
    Returns the greatest common divisor of ``|x1|`` and ``|x2|``
```

```
numpy.lcm
    Returns the lowest common multiple of ``|x1|`` and ``|x2|``
numpy.intersect1d
    Find the intersection of two arrays.
numpy.ma.common_fill_value
    Return the common filling value of two masked arrays, if any.
numpy.add
    Add arguments element-wise.
numpy.less
    Return the truth value of (x1 < x2) element-wise.
numpy.sign
    Returns an element-wise indication of the sign of a number.
numpy.sqrt
    Return the non-negative square-root of an array, element-wise.
numpy.choose
    Construct an array from an index array and a set of arrays to choose from.
numpy.einsum
    einsum(subscripts, *operands, out=None, dtype=None, order='K',
numpy.nditer
    Efficient multi-dimensional iterator object to iterate over arrays.
numpy.greater
    Return the truth value of (x1 > x2) element-wise.
numpy.nonzero
    Return the indices of the elements that are non-zero.
numpy.less_equal
    Return the truth value of (x1 =< x2) element-wise.
numpy.bitwise_not
    Compute bit-wise inversion, or bit-wise NOT, element-wise.
numpy.histogram2d
    Compute the bi-dimensional histogram of two data samples.
numpy.ma.add
    Add arguments element-wise.
numpy.ma.cov
    Estimate the covariance matrix.
numpy.ma.less
    Return the truth value of (x1 < x2) element-wise.
numpy.ma.sqrt
```

```
    Return the non-negative square-root of an array, element-wise.
numpy.greater_equal
    Return the truth value of (x1 >= x2) element-wise.
numpy.linalg.qr
    Compute the qr factorization of a matrix.
numpy.ma.greater
    Return the truth value of (x1 > x2) element-wise.
numpy.ma.nonzero
    nonzero(self)
numpy.ma.less_equal
    Return the truth value of (x1 =< x2) element-wise.
numpy.histogram_bin_edges
    Function to calculate only the edges of the bins used by the `histogram` function.
numpy.ma.greater_equal
    Return the truth value of (x1 >= x2) element-wise.
```

我们发现返回得到第一个函数 `np.intersect1d` 就是我们要找的，进一步查看其用法。

```
In [64]: np.info(np.intersect1d)

 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.

Parameters
----------
ar1, ar2 : array_like
    Input arrays. Will be flattened if not already 1D.
assume_unique : bool
    If True, the input arrays are both assumed to be unique, which
    can speed up the calculation.  Default is False.
return_indices : bool
    If True, the indices which correspond to the intersection of the two
    arrays are returned. The first instance of a value is used if there are
    multiple. Default is False.
```

```
    .. versionadded:: 1.15.0

Returns
-------
intersect1d : ndarray
    Sorted 1D array of common and unique elements.
comm1 : ndarray
    The indices of the first occurrences of the common values in `ar1`.
    Only provided if `return_indices` is True.
comm2 : ndarray
    The indices of the first occurrences of the common values in `ar2`.
    Only provided if `return_indices` is True.


See Also
--------
numpy.lib.arraysetops : Module with a number of other functions for
                        performing set operations on arrays.

Examples
--------
>>> np.intersect1d([1, 3, 4, 3], [3, 1, 2, 1])
array([1, 3])

To intersect more than two arrays, use functools.reduce:

>>> from functools import reduce
>>> reduce(np.intersect1d, ([1, 3, 4, 3], [3, 1, 2, 1], [6, 3, 4, 2]))
array([3])

To return the indices of the values common to the input arrays
along with the intersected values:
>>> 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)
>>> x_ind, y_ind
(array([0, 2, 4]), array([1, 0, 2]))
```

```
>>> xy, x[x_ind], y[y_ind]
(array([1, 2, 4]), array([1, 2, 4]), array([1, 2, 4]))
```

根据文档就可以直接使用了。

```
In [65]: z1 = np.arange(-5, 5)
         z2 = np.arange(10)
         np.intersect1d(z1, z2)

Out[65]: array([0, 1, 2, 3, 4])
```

**31. 如何忽视所有 Numpy 的警告（不推荐）**

```
In [66]: # 自杀模式启动:-)
         defaults = np.seterr(all="ignore")
         Z = np.ones(1) / 0
         # 恢复理智
         _ = np.seterr(**defaults)

In [67]: # 也可以定义错误处理的细节
         with np.errstate(divide='warn'):
             Z = np.ones(1) / 0

/home/shensir/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3: RuntimeWarning: di
  This is separate from the ipykernel package so we can avoid doing imports until
```

**32. 下面的表达式会返回 True 吗**

```
np.sqrt(-1) == np.emath.sqrt(-1)

In [68]: np.sqrt(-1) == np.emath.sqrt(-1)

Out[68]: False

In [69]: np.sqrt(-1), np.emath.sqrt(-1)

Out[69]: (nan, 1j)
```

**33. 如何获取昨天，今天，明天的日期**

```
In [70]: yesterday = np.datetime64('today', 'D') - np.timedelta64(1, 'D')
         today     = np.datetime64('today', 'D')
         tomorrow  = np.datetime64('today', 'D') + np.timedelta64(1, 'D')

In [71]: yesterday, today, tomorrow

Out[71]: (numpy.datetime64('2019-09-10'),
          numpy.datetime64('2019-09-11'),
          numpy.datetime64('2019-09-12'))
```

**34. 如何获取 2016 年 7 月全部 31 天的日期**

```
In [72]: z = np.arange('2016-07', '2016-08', dtype='datetime64[D]')
         z

Out[72]: array(['2016-07-01', '2016-07-02', '2016-07-03', '2016-07-04',
                '2016-07-05', '2016-07-06', '2016-07-07', '2016-07-08',
                '2016-07-09', '2016-07-10', '2016-07-11', '2016-07-12',
                '2016-07-13', '2016-07-14', '2016-07-15', '2016-07-16',
                '2016-07-17', '2016-07-18', '2016-07-19', '2016-07-20',
                '2016-07-21', '2016-07-22', '2016-07-23', '2016-07-24',
                '2016-07-25', '2016-07-26', '2016-07-27', '2016-07-28',
                '2016-07-29', '2016-07-30', '2016-07-31'], dtype='datetime64[D]')
```

**35. 如何以替换的方式（in place）计算 $((A+B)*(-A/2))$（不通过复制）**

```
In [73]: A = np.ones(3)*1
         B = np.ones(3)*2
         C = np.ones(3)*3
         np.add(A,B,out=B)
         np.divide(A,2,out=A)
         np.negative(A,out=A)
         np.multiply(A,B,out=A)

Out[73]: array([-1.5, -1.5, -1.5])
```

**36. 用 5 种方法提取随机数组中的整数部分**

```
In [74]: z = np.random.uniform(0, 10, 10)
         z

Out[74]: array([5.07555068, 0.21193302, 4.33521758, 4.46313056, 2.38819989,
                8.30245732, 7.44764177, 5.86479001, 4.92867853, 4.8735588 ])

In [75]: # 1
         z - z % 1

Out[75]: array([5., 0., 4., 4., 2., 8., 7., 5., 4., 4.])

In [76]: # 2
         np.floor(z)

Out[76]: array([5., 0., 4., 4., 2., 8., 7., 5., 4., 4.])

In [77]: # 3
         np.ceil(z) - 1

Out[77]: array([5., 0., 4., 4., 2., 8., 7., 5., 4., 4.])

In [78]: # 4
         z.astype(int)

Out[78]: array([5, 0, 4, 4, 2, 8, 7, 5, 4, 4])

In [79]: # 5
         np.trunc(z)

Out[79]: array([5., 0., 4., 4., 2., 8., 7., 5., 4., 4.])
```

**37. 创建一个 5x5 的矩阵，每行均为 0 到 4**

```
In [80]: # 1, 答案的方法
         z = np.zeros((5, 5))
         z += np.arange(5)
         z

Out[80]: array([[0., 1., 2., 3., 4.],
                [0., 1., 2., 3., 4.],
                [0., 1., 2., 3., 4.],
                [0., 1., 2., 3., 4.],
                [0., 1., 2., 3., 4.]])
```

```
In [81]: # 2, 利用 tile
         z = np.tile(np.arange(5), (5, 1))
         z

Out[81]: array([[0, 1, 2, 3, 4],
                [0, 1, 2, 3, 4],
                [0, 1, 2, 3, 4],
                [0, 1, 2, 3, 4],
                [0, 1, 2, 3, 4]])
```

**38. 现有一个可以生成 10 个整数的生成器函数，利用其建立一个数组**

```
In [82]: # 1, 答案的方法
         def gen():
             for i in range(10):
                 yield i

         z = np.fromiter(gen(), dtype=float, count=-1)
         z

Out[82]: array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])

In [83]: # 2, 列表解析
         z = np.array([i for i in gen()])
         z

Out[83]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

**39. 创建一个长度为 10，范围从 0 到 1 的向量（不包括 0，1）**

```
In [84]: z = np.linspace(0, 1, 11, endpoint=False)[1:]
         z

Out[84]: array([0.09090909, 0.18181818, 0.27272727, 0.36363636, 0.45454545,
                0.54545455, 0.63636364, 0.72727273, 0.81818182, 0.90909091])
```

**40. 创建一个长度为 10 的随机数组并排序**

```
In [85]: z = np.random.random(10)
         z
```

```
Out[85]: array([0.2667407 , 0.6050111 , 0.75354372, 0.27058423, 0.52230328,
                0.09832853, 0.71363667, 0.88404059, 0.56705442, 0.99448158])

In [86]: z.sort()
         z

Out[86]: array([0.09832853, 0.2667407 , 0.27058423, 0.52230328, 0.56705442,
                0.6050111 , 0.71363667, 0.75354372, 0.88404059, 0.99448158])
```

**41. 对于长度较小的数组，如何更高效地求和（相对 np.sum）**

```
In [87]: z = np.arange(10)
         np.add.reduce(z)

Out[87]: 45

In [88]: %timeit np.add.reduce(z)

1.52 µs ś 26.7 ns per loop (mean ś std. dev. of 7 runs, 1000000 loops each)


In [89]: %timeit np.sum(z)

3.94 µs ś 299 ns per loop (mean ś std. dev. of 7 runs, 100000 loops each)
```

可以看到 np.add.reduce 此时差不多快上一倍

**42. 检查两个数组 A，B 是否相等**

```
In [90]: A = np.random.randint(0,2,5)
         B = np.random.randint(0,2,5)

In [91]: # 1, 已知 A, B 的 shape 相等
         # 存在容错，适用于浮点数的比较
         np.allclose(A, B)

Out[91]: False

In [92]: # 2. 同时检查 shape 与数值
         # 要求数值完全相等
         np.array_equal(A, B)

Out[92]: False
```

**43. 限制数组为不可变数组（read only）**

```
In [93]: z = np.zeros(10)
         z.flags

Out[93]:   C_CONTIGUOUS : True
           F_CONTIGUOUS : True
           OWNDATA : True
           WRITEABLE : True
           ALIGNED : True
           WRITEBACKIFCOPY : False
           UPDATEIFCOPY : False

In [94]: z.flags.writeable = False
         z.flags

Out[94]:   C_CONTIGUOUS : True
           F_CONTIGUOUS : True
           OWNDATA : True
           WRITEABLE : False
           ALIGNED : True
           WRITEBACKIFCOPY : False
           UPDATEIFCOPY : False

In [95]: z[0] = 1

         ---------------------------------------------------------------------------

         ValueError                                Traceback (most recent call last)

         <ipython-input-95-ba34f733b6cb> in <module>
    ----> 1 z[0] = 1


         ValueError: assignment destination is read-only
```

**44. 给定 10x2 矩阵代表平面座标系中座标，将其转化为极座标系座标**

```
In [96]: z = np.random.random((10, 2))
         x, y = z[:, 0], z[:, 1]
         r = np.sqrt(x**2 + y**2)
         theta = np.arctan2(y, x)
         r, theta
```

```
Out[96]: (array([1.0191402 , 0.3789241 , 1.01760735, 0.28642266, 1.09696314,
                 0.35963712, 0.86915363, 0.91189768, 1.18794229, 0.97413596]),
          array([1.08779371, 0.93177059, 0.79171987, 1.50554416, 1.01316742,
                 0.34043887, 0.93478081, 1.23714664, 0.70362634, 1.29930594]))
```

**45. 创建一个长度为 10 的随机向量，并将其中最大的数改为 0**

```
In [97]: z = np.random.random(10)
         z
```

```
Out[97]: array([0.93864246, 0.74504455, 0.91073504, 0.23722471, 0.49496735,
                0.80987834, 0.95456578, 0.63748325, 0.91084975, 0.69213675])
```

```
In [98]: z[z.argmax()] = 0
         z
```

```
Out[98]: array([0.93864246, 0.74504455, 0.91073504, 0.23722471, 0.49496735,
                0.80987834, 0.        , 0.63748325, 0.91084975, 0.69213675])
```

**46. 创建一个结构化的数组，其元素为 x 轴，y 轴的座标，并覆盖 [0,1]x[0, 1]**

```
In [99]: z = np.zeros((5, 5), [('x', float), ('y', float)])
         z = np.meshgrid(np.linspace(0, 1, 5),
                         np.linspace(0, 1, 5))
         z
```

```
Out[99]: [array([[0.  , 0.25, 0.5 , 0.75, 1.  ],
                 [0.  , 0.25, 0.5 , 0.75, 1.  ],
                 [0.  , 0.25, 0.5 , 0.75, 1.  ],
                 [0.  , 0.25, 0.5 , 0.75, 1.  ],
                 [0.  , 0.25, 0.5 , 0.75, 1.  ]]),
          array([[0.  , 0.  , 0.  , 0.  , 0.  ],
                 [0.25, 0.25, 0.25, 0.25, 0.25],
                 [0.5 , 0.5 , 0.5 , 0.5 , 0.5 ],
                 [0.75, 0.75, 0.75, 0.75, 0.75],
                 [1.  , 1.  , 1.  , 1.  , 1.  ]])]
```

**47. 给定两个数组 X，Y，计算其柯西矩阵 C(Cauchy Matrix) 并求其行列式**

$$C_{ij} = \frac{1}{x_i - y_j}$$

```
In [100]: x = np.arange(8)
          y = x + 0.5
          C = np.subtract.outer(x, y)
          C
```

```
Out[100]: array([[-0.5, -1.5, -2.5, -3.5, -4.5, -5.5, -6.5, -7.5],
                 [ 0.5, -0.5, -1.5, -2.5, -3.5, -4.5, -5.5, -6.5],
                 [ 1.5,  0.5, -0.5, -1.5, -2.5, -3.5, -4.5, -5.5],
                 [ 2.5,  1.5,  0.5, -0.5, -1.5, -2.5, -3.5, -4.5],
                 [ 3.5,  2.5,  1.5,  0.5, -0.5, -1.5, -2.5, -3.5],
                 [ 4.5,  3.5,  2.5,  1.5,  0.5, -0.5, -1.5, -2.5],
                 [ 5.5,  4.5,  3.5,  2.5,  1.5,  0.5, -0.5, -1.5],
                 [ 6.5,  5.5,  4.5,  3.5,  2.5,  1.5,  0.5, -0.5]])
```

```
In [101]: np.linalg.det(C)
```

```
Out[101]: 1.8457040860738383e-92
```

其实这里 `np.subtract.outer` 就等于进行了 broadcast, 我们也可以像下面这样写。

```
In [102]: C_test = x.reshape(8, 1)- y.reshape(1, 8)
          C_test
```

```
Out[102]: array([[-0.5, -1.5, -2.5, -3.5, -4.5, -5.5, -6.5, -7.5],
                 [ 0.5, -0.5, -1.5, -2.5, -3.5, -4.5, -5.5, -6.5],
                 [ 1.5,  0.5, -0.5, -1.5, -2.5, -3.5, -4.5, -5.5],
                 [ 2.5,  1.5,  0.5, -0.5, -1.5, -2.5, -3.5, -4.5],
                 [ 3.5,  2.5,  1.5,  0.5, -0.5, -1.5, -2.5, -3.5],
                 [ 4.5,  3.5,  2.5,  1.5,  0.5, -0.5, -1.5, -2.5],
                 [ 5.5,  4.5,  3.5,  2.5,  1.5,  0.5, -0.5, -1.5],
                 [ 6.5,  5.5,  4.5,  3.5,  2.5,  1.5,  0.5, -0.5]])
```

```
In [103]: np.linalg.det(C_test)
```

```
Out[103]: 1.8457040860738383e-92
```

```
In [104]: # 测试两种方法返回的 C 是否相同
          np.array_equal(C, C_test)
```

```
Out[104]: True
```

### 48. 打印 Numpy 所有标量类型 (scalar type) 可表示的最值

```
In [105]: for dtype in [np.int8, np.int16, np.int32, np.int64]:
              info = np.iinfo(dtype)
              print(f"{dtype}: min={info.min}, max={info.max}")
          for dtype in [np.float16, np.float32, np.float64, np.float128]:
              info = np.finfo(dtype)
              print(f"{dtype}: min={info.min}, max={info.max}")
```

```
<class 'numpy.int8'>: min=-128, max=127
<class 'numpy.int16'>: min=-32768, max=32767
<class 'numpy.int32'>: min=-2147483648, max=2147483647
<class 'numpy.int64'>: min=-9223372036854775808, max=9223372036854775807
<class 'numpy.float16'>: min=-65504.0, max=65504.0
<class 'numpy.float32'>: min=-3.4028234663852886e+38, max=3.4028234663852886e+38
<class 'numpy.float64'>: min=-1.7976931348623157e+308, max=1.7976931348623157e+308
<class 'numpy.float128'>: min=-inf, max=inf
```

### 49. 打印数组所有元素（不省略）

```
In [106]: with np.printoptions(threshold=np.inf):
              z = np.ones((10, 10))
              print(z)
```

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

### 50. 给定一个数，在数组中找出距离其最近的数

```
In [107]: # 给定的数组
          z = np.random.uniform(0, 1, 10)
          z

Out[107]: array([0.04294299, 0.8335869 , 0.36994852, 0.936557  , 0.48305288,
                 0.12533161, 0.96445418, 0.01702583, 0.67657077, 0.14043997])

In [108]: # 给定的数
          x = 0.5
          # 定位距离最近数的位置
          index = np.abs(z - x).argmin()
          # 找到该数字
          z[index]

Out[108]: 0.48305287517652307
```

**51. 创建一个结构化的数组，其元素为一个座标 (x,y) 和一个颜色参数 (r,g,b)** 和 46 题类似。> 另外，这可以是一个像素点的表示方式

```
In [109]: z = np.zeros(10, [('position', [('x', float, 1),
                                          ('y', float, 1)]),
                            ('color', [('r', float, 1),
                                       ('g', float, 1),
                                       ('b', float, 1)])
                            ]
                    )
          z

Out[109]: array([((0., 0.), (0., 0., 0.)), ((0., 0.), (0., 0., 0.)),
                 ((0., 0.), (0., 0., 0.)), ((0., 0.), (0., 0., 0.)),
                 ((0., 0.), (0., 0., 0.)), ((0., 0.), (0., 0., 0.)),
                 ((0., 0.), (0., 0., 0.)), ((0., 0.), (0., 0., 0.)),
                 ((0., 0.), (0., 0., 0.)), ((0., 0.), (0., 0., 0.))],
                dtype=[('position', [('x', '<f8'), ('y', '<f8')]), ('color', [('r', '<f8'), ('g
```

**52. 考虑一个形状为（10，2）的随机向量，若其代表二维平面中的点，求各点之间的距离**

```
In [110]: z = np.random.random((10, 2))
          x, y = np.atleast_2d(z[:, 0], z[:, 1])
          d = np.sqrt((x - x.T)**2 + (y - y.T)**2)
          d
```

```
Out[110]: array([[0.        , 0.82757935, 0.64681519, 0.43368238, 0.85165766,
                  0.57613768, 0.48566732, 0.44501945, 0.90701745, 0.55962651],
                 [0.82757935, 0.        , 0.3477681 , 0.96918723, 0.67596378,
                  0.71128017, 0.37897508, 0.43258791, 0.50371629, 0.26882844],
                 [0.64681519, 0.3477681 , 0.        , 0.65720606, 0.37061334,
                  0.36563979, 0.18181895, 0.43879256, 0.29206274, 0.27712044],
                 [0.43368238, 0.96918723, 0.65720606, 0.        , 0.64632996,
                  0.35301865, 0.59527913, 0.73887741, 0.80043261, 0.74933684],
                 [0.85165766, 0.67596378, 0.37061334, 0.64632996, 0.        ,
                  0.30214225, 0.51433777, 0.78889208, 0.22248961, 0.6454205 ],
                 [0.57613768, 0.71128017, 0.36563979, 0.35301865, 0.30214225,
                  0.        , 0.39640111, 0.64228203, 0.44903039, 0.56316235],
                 [0.48566732, 0.37897508, 0.18181895, 0.59527913, 0.51433777,
                  0.39640111, 0.        , 0.274627  , 0.47255738, 0.1680208 ],
                 [0.44501945, 0.43258791, 0.43879256, 0.73887741, 0.78889208,
                  0.64228203, 0.274627  , 0.        , 0.72766324, 0.18906137],
                 [0.90701745, 0.50371629, 0.29206274, 0.80043261, 0.22248961,
                  0.44903039, 0.47255738, 0.72766324, 0.        , 0.55335753],
                 [0.55962651, 0.26882844, 0.27712044, 0.74933684, 0.6454205 ,
                  0.56316235, 0.1680208 , 0.18906137, 0.55335753, 0.        ]])
```

这里使用 `np.atleast_2d` 使得我们得到的 `x`，`y` 直接就是 2 维的数组，方便了我们后面直接使用 `broadcasting`. 我们也可以采用下面的方法代替这行，但是不够简洁：

```
x = z[:, 0].reshape(10, 1)
y = z[:, 1].reshape(1, 10)
```

此外我们也可以使用 `scipy` 内置的函数，其效率要高一些。

```
In [111]: import scipy
          import scipy.spatial

          d = scipy.spatial.distance.cdist(z, z)
          d
```

```
Out[111]: array([[0.        , 0.82757935, 0.64681519, 0.43368238, 0.85165766,
                  0.57613768, 0.48566732, 0.44501945, 0.90701745, 0.55962651],
                 [0.82757935, 0.        , 0.3477681 , 0.96918723, 0.67596378,
                  0.71128017, 0.37897508, 0.43258791, 0.50371629, 0.26882844],
```

```
       [0.64681519, 0.3477681 , 0.        , 0.65720606, 0.37061334,
        0.36563979, 0.18181895, 0.43879256, 0.29206274, 0.27712044],
       [0.43368238, 0.96918723, 0.65720606, 0.        , 0.64632996,
        0.35301865, 0.59527913, 0.73887741, 0.80043261, 0.74933684],
       [0.85165766, 0.67596378, 0.37061334, 0.64632996, 0.        ,
        0.30214225, 0.51433777, 0.78889208, 0.22248961, 0.6454205 ],
       [0.57613768, 0.71128017, 0.36563979, 0.35301865, 0.30214225,
        0.        , 0.39640111, 0.64228203, 0.44903039, 0.56316235],
       [0.48566732, 0.37897508, 0.18181895, 0.59527913, 0.51433777,
        0.39640111, 0.        , 0.274627  , 0.47255738, 0.1680208 ],
       [0.44501945, 0.43258791, 0.43879256, 0.73887741, 0.78889208,
        0.64228203, 0.274627  , 0.        , 0.72766324, 0.18906137],
       [0.90701745, 0.50371629, 0.29206274, 0.80043261, 0.22248961,
        0.44903039, 0.47255738, 0.72766324, 0.        , 0.55335753],
       [0.55962651, 0.26882844, 0.27712044, 0.74933684, 0.6454205 ,
        0.56316235, 0.1680208 , 0.18906137, 0.55335753, 0.        ]])
```

**53. 将一个 32 位的浮点数数组，（不使用额外内存）转化为 32 为的整数数组**

```
In [112]: z = np.zeros(10, dtype=np.float32)
          z

Out[112]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)

In [113]: z = z.astype(np.int32, copy=False)
          z

Out[113]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

**54. 如何读取下面的文件**

```
1, 2, 3, 4, 5
6,  ,  , 7, 8
 ,  , 9,10,11
```

```
In [114]: from io import StringIO
          # “假”的文件
          s = StringIO("""1, 2, 3, 4, 5\n
                          6,  ,  , 7, 8\n
                           ,  , 9,10,11\n""")
```

```
z = np.genfromtxt(s, delimiter=",", missing_values=' ')
z
```

```
Out[114]: array([[ 1.,  2.,  3.,  4.,  5.],
                 [ 6., nan, nan,  7.,  8.],
                 [nan, nan,  9., 10., 11.]])
```

**55. Python 内置有 enumerate,Numpy 中与之对应的是？**

```
In [115]: Z = np.arange(9).reshape(3,3)
          for index, value in np.ndenumerate(Z):
              print(index, value)
```

```
(0, 0) 0
(0, 1) 1
(0, 2) 2
(1, 0) 3
(1, 1) 4
(1, 2) 5
(2, 0) 6
(2, 1) 7
(2, 2) 8
```

```
In [116]: for index in np.ndindex(Z.shape):
              print(index, Z[index])
```

```
(0, 0) 0
(0, 1) 1
(0, 2) 2
(1, 0) 3
(1, 1) 4
(1, 2) 5
(2, 0) 6
(2, 1) 7
(2, 2) 8
```

**56. 生成二维高斯分布**

```
In [117]: X, Y = np.meshgrid(np.linspace(-1,1,10), np.linspace(-1,1,10))
          D = np.sqrt(X*X+Y*Y)
          sigma, mu = 1.0, 0.0
          G = np.exp(-( (D-mu)**2 / ( 2.0 * sigma**2 ) ) )
          G

Out[117]: array([[0.36787944, 0.44822088, 0.51979489, 0.57375342, 0.60279818,
                  0.60279818, 0.57375342, 0.51979489, 0.44822088, 0.36787944],
                 [0.44822088, 0.54610814, 0.63331324, 0.69905581, 0.73444367,
                  0.73444367, 0.69905581, 0.63331324, 0.54610814, 0.44822088],
                 [0.51979489, 0.63331324, 0.73444367, 0.81068432, 0.85172308,
                  0.85172308, 0.81068432, 0.73444367, 0.63331324, 0.51979489],
                 [0.57375342, 0.69905581, 0.81068432, 0.89483932, 0.9401382 ,
                  0.9401382 , 0.89483932, 0.81068432, 0.69905581, 0.57375342],
                 [0.60279818, 0.73444367, 0.85172308, 0.9401382 , 0.98773022,
                  0.98773022, 0.9401382 , 0.85172308, 0.73444367, 0.60279818],
                 [0.60279818, 0.73444367, 0.85172308, 0.9401382 , 0.98773022,
                  0.98773022, 0.9401382 , 0.85172308, 0.73444367, 0.60279818],
                 [0.57375342, 0.69905581, 0.81068432, 0.89483932, 0.9401382 ,
                  0.9401382 , 0.89483932, 0.81068432, 0.69905581, 0.57375342],
                 [0.51979489, 0.63331324, 0.73444367, 0.81068432, 0.85172308,
                  0.85172308, 0.81068432, 0.73444367, 0.63331324, 0.51979489],
                 [0.44822088, 0.54610814, 0.63331324, 0.69905581, 0.73444367,
                  0.73444367, 0.69905581, 0.63331324, 0.54610814, 0.44822088],
                 [0.36787944, 0.44822088, 0.51979489, 0.57375342, 0.60279818,
                  0.60279818, 0.57375342, 0.51979489, 0.44822088, 0.36787944]])
```

**57. 随机地在二维数组中放置 p 个元素**

```
In [118]: n = 5
          p = 3
          z = np.zeros((n, n))
          np.put(z, np.random.choice(range(n*n), p), 1)
          z

Out[118]: array([[0., 0., 0., 0., 0.],
                 [1., 0., 1., 0., 1.],
```

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

**58. 矩阵每行进行中心化（减去均值）**

```
In [119]: # 1, 答案的方法
          z = np.arange(10).reshape(2, 5)
          z

Out[119]: array([[0, 1, 2, 3, 4],
                 [5, 6, 7, 8, 9]])

In [120]: z_new = z - z.mean(axis=1, keepdims=True)
          z_new

Out[120]: array([[-2., -1.,  0.,  1.,  2.],
                 [-2., -1.,  0.,  1.,  2.]])
```

注意这里，设置 `keepdims` 可以方便进行 `broadcasting`, 免去手动 `reshape` 的流程。

我们也可以考虑对矩阵每一行应用一个中心化的函数来完成任务。

```
In [121]: # 2, apply 方法
          def centered(xs):
              return xs - xs.mean()

          z_new = np.apply_along_axis(centered, 1, z)
          z_new

Out[121]: array([[-2., -1.,  0.,  1.,  2.],
                 [-2., -1.,  0.,  1.,  2.]])
```

**59. 根据某列数据来排列数组**

```
In [122]: z = np.random.randint(0, 10, (3, 3))
          z

Out[122]: array([[9, 0, 7],
                 [8, 4, 6],
                 [5, 7, 5]])
```

```
In [123]: # 根据第二列顺序排列
          z[z[:, 1].argsort(), ]

Out[123]: array([[9, 0, 7],
                 [8, 4, 6],
                 [5, 7, 5]])
```

**60. 判断二维数组是否含有空列 (全为 0)**

```
In [124]: z = np.random.randint(0, 3, (3, 10))
          z

Out[124]: array([[1, 2, 2, 1, 2, 0, 0, 1, 2, 2],
                 [2, 0, 2, 2, 2, 1, 2, 0, 0, 0],
                 [0, 1, 0, 1, 1, 2, 1, 0, 0, 2]])

In [125]: print((~z.any(axis=0)).any())

False
```

一旦有空列的时候，z.any(axis=0) 返回 False，即 ~z.any(axis=0) 返回 True，之后再应用 any, 则比返回 True。反之，若无任何空列，~z.any(axis=0) 全部返回 False，应用 any，依旧返回 False。

**61. 给定一个数，在数组中找出距离其最近的数** 与 50 题重复.

**62. 考虑两个数组，形状分别是 (3，1)，(1，3)，如何使用迭代器将其相加？**

```
In [126]: A = np.arange(3).reshape(3,1)
          B = np.arange(3).reshape(1,3)
          it = np.nditer([A,B,None])
          for x,y,z in it: z[...] = x + y
          print(it.operands[2])

[[0 1 2]
 [1 2 3]
 [2 3 4]]
```

**63. 创建一个带有名称属性的数组类**

```
In [127]: class NamedArray(np.ndarray):
              def __new__(cls, array, name="no name"):
                  obj = np.asarray(array).view(cls)
                  obj.name = name
                  return obj
              def __array_finalize__(self, obj):
                  if obj is None: return
                  self.info = getattr(obj, 'name', "no name")

          Z = NamedArray(np.arange(10), "range_10")
          print (Z.name)
```

```
range_10
```

**64. 给定一个数值向量，和一个索引向量，根据后者的索引，在前者对应位置加 1（注意重复索引）**

```
In [128]: # Author: Brett Olsen
          Z = np.ones(10)
          I = np.random.randint(0,len(Z),20)

          Z_new = Z + np.bincount(I, minlength=len(Z))
          Z_new
```

```
Out[128]: array([2., 2., 6., 5., 2., 3., 3., 3., 3., 1.])
```

```
In [129]: # Another solution
          # Author: Bartosz Telenczuk
          np.add.at(Z, I, 1)
          Z
```

```
Out[129]: array([2., 2., 6., 5., 2., 3., 3., 3., 3., 1.])
```

**65. 根据索引列表 I，对数值列表 X 进行累加，得到 F**

```
In [130]: X = [1,2,3,4,5,6]
          I = [1,3,9,3,4,1]
          F = np.bincount(I,X)
          print(F)
```

```
[0. 7. 0. 6. 5. 0. 0. 0. 0. 3.]
```

这里 bincount 的用法有点绕... 让我们举个例子先:-)

我们把 I 中出现的数字比作个人的银行账户编号，可以看到这里最大的编号为 9，所以我们暂时可以只考虑编号 0~9 的账户情况，这 10 个账户，正好对应最后得到 F 的 10 个位置。进一步地，I 与 X 结合，可以看作这些银行账户交易的流水，其中 I 为账户编号，X 为对应的金额。比如，因为 I[0]=1，我们知道是账户 1 发生交易，对应的 X[0]=1，所以账户 1 的金额要加 1；此外账户 1 还发生一次交易（I[5]=1），对应的金额 X[5] = 6，所以这段时间账户 1 总的金额就是 6 + 1 = 7，所以得到 F[1] = 7.

简言之，我们的任务就是根据 X 和 I 组成的交易流水，来计算各个账户总的金额。

**66. 给定一张照片（w, h, 3），计算其中不同颜色的个数**

```
In [131]: # Author: Nadav Horesh

          w,h = 16, 16
          I = np.random.randint(0,2,(h,w,3)).astype(np.ubyte)

In [132]: # 注意我们这里必须先乘 256*256，否则会爆栈
          F = I[...,0]*(256*256) + I[...,1]*256 +I[...,2]
          n = len(np.unique(F))
          print(n)
```

```
8
```

**67. 考虑一个四维数组，计算后两个轴上的元素和**

```
In [133]: A = np.random.randint(0,10,(3,4,3,4))
          A.sum(axis=(-2,-1))

Out[133]: array([[52, 44, 44, 68],
                 [61, 60, 44, 53],
                 [42, 66, 72, 57]])
```

**68. 给定向量 D，根据索引数组 S 得到子集，计算子集上的均值**

```
In [134]: D = np.random.uniform(0,1,100)
          S = np.random.randint(0,10,100)
```

```
D_sums = np.bincount(S, weights=D)
D_counts = np.bincount(S)
D_means = D_sums / D_counts
print(D_means)
```

```
[0.58150261 0.55134568 0.57372635 0.46632861 0.51824552 0.5358013
 0.42991891 0.46169025 0.39369914 0.44701464]
```

结合 65 题给出的例子，在那里是根据流水计算各个账户总的金额，这里是计算各个账户每次交易的平均金额，也就是该账户总的金额除以其交易的次数。

**69. 获取矩阵点乘 (dot product) 结果的对角线元素**

```
In [135]: # Author: Mathieu Blondel

          A = np.random.uniform(0,1,(5,5))
          B = np.random.uniform(0,1,(5,5))

          # 慢的版本
          np.diag(np.dot(A, B))

          # 快的版本
          np.sum(A * B.T, axis=1)

          # 更快的版本
          np.einsum("ij,ji->i", A, B)

Out[135]: array([1.27203858, 1.95734402, 1.1140674 , 1.32168438, 0.85313171])
```

**70. 如何在数组 [1, 2, 3, 4, 5] 每两个值的中间添加三个 0**

```
In [136]: z = np.array([1, 2, 3, 4, 5])
          nz = 3  # 0 的个数
          v = np.zeros(len(z) + nz*(len(z)-1))
          v[::nz+1] = z
          v

Out[136]: array([1., 0., 0., 0., 2., 0., 0., 0., 3., 0., 0., 0., 4., 0., 0., 0., 5.])
```

**71. 维度分别为（5，5，3），（5，5）的两个数组相乘**

```
In [137]: A = np.ones((5,5,3))
          B = 2*np.ones((5,5))
          print(A * B[:,:,None])
```

```
[[[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]

 [[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]

 [[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]

 [[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]

 [[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]]
```

**72. 交换数组的两行**

```
In [138]: A = np.arange(25).reshape((5, 5))
          A

Out[138]: array([[ 0,  1,  2,  3,  4],
                 [ 5,  6,  7,  8,  9],
                 [10, 11, 12, 13, 14],
                 [15, 16, 17, 18, 19],
                 [20, 21, 22, 23, 24]])

In [139]: # 交换第一、二两行
          A[[0, 1]] = A[[1, 0]]
          A

Out[139]: array([[ 5,  6,  7,  8,  9],
                 [ 0,  1,  2,  3,  4],
                 [10, 11, 12, 13, 14],
                 [15, 16, 17, 18, 19],
                 [20, 21, 22, 23, 24]])
```

**73. 给定 10 个三元组描述 10 个三角形，找出所有边的集合**

```
In [140]: faces = np.random.randint(0,100,(10,3))
          F = np.roll(faces.repeat(2,axis=1),-1,axis=1)
          F = F.reshape(len(F)*3,2)
          F = np.sort(F,axis=1)
          G = F.view( dtype=[('p0',F.dtype),('p1',F.dtype)] )
          G = np.unique(G)
          print(G)

[( 6, 31) ( 6, 96) ( 7, 30) ( 7, 82) (11, 18) (11, 48) (15, 28) (18, 48)
 (24, 29) (24, 65) (25, 51) (25, 84) (28, 28) (29, 32) (29, 65) (30, 82)
 (31, 96) (32, 65) (43, 77) (43, 88) (46, 54) (46, 71) (51, 84) (54, 71)
 (68, 83) (68, 89) (77, 88) (83, 89)]
```

感觉这里的方法比较巧妙，可以将每一步拆解来理解怎么将每条边抽取出来。之后比较细节的地方就是对描述 “边” 的二元组排序 (如果不排序，后面比较的时候就会出现 (a, b) 与（b, a）不是同一条边的错误判断)，之后通过 `view` 转化类型，方便比较（使用 `np.unique`）

**74. 给定 A，我们有 C = np.bincount(A), 那么，给定 C，如何找到对应的 A？**

```
In [141]: C = np.bincount([1,1,2,3,4,4,6])
          A = np.repeat(np.arange(len(C)), C)
          print(A)
```

```
[1 1 2 3 4 4 6]
```

**75. 用滑动窗口计算平均值**

```
In [142]: def moving_average(a, n=3) :
              ret = np.cumsum(a, dtype=float)
              ret[n:] = ret[n:] - ret[:-n]
              return ret[n - 1:] / n
          Z = np.arange(20)
          print(moving_average(Z, n=3))
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18.]
```

**76. 给定一个一维数组，组建一个二维数组，使得第一行为 Z[0], Z[1], Z[2], 第二行为 Z[1], Z[2], Z[3], 依此类推，最后一行为 Z[-3], Z[-2], Z[-1]**

```
In [143]: from numpy.lib import stride_tricks

          def rolling(a, window):
              shape = (a.size - window + 1, window)
              strides = (a.itemsize, a.itemsize)
              return stride_tricks.as_strided(a, shape=shape, strides=strides)
          Z = rolling(np.arange(10), 3)
          print(Z)
```

```
[[0 1 2]
 [1 2 3]
 [2 3 4]
 [3 4 5]
 [4 5 6]
 [5 6 7]
 [6 7 8]
```

```
 [7 8 9]]
```

**77. 如何原地对布尔值取反，如何原地改变数字的正负**

```
In [144]: Z = np.random.randint(0,2,100)
          np.logical_not(Z, out=Z)

          Z = np.random.uniform(-1.0,1.0,100)
          np.negative(Z, out=Z)
```

```
Out[144]: array([-0.73959528,  0.74138049, -0.94811162, -0.50843124, -0.83335491,
                  0.92459432,  0.42830231, -0.64581576,  0.45053274,  0.8965521 ,
                  0.70411344,  0.96701466, -0.56483325,  0.30175679, -0.49935569,
                  0.89009322,  0.84360355, -0.57868105,  0.25912389,  0.77807512,
                  0.18956795, -0.30259756,  0.12742359,  0.5950497 ,  0.11222084,
                 -0.48429619,  0.80796982, -0.02344253, -0.68723289, -0.9870873 ,
                 -0.26151235,  0.42905334,  0.74795648,  0.97328859,  0.32368876,
                  0.03135965, -0.09238209,  0.92792125,  0.8775511 ,  0.95367849,
                 -0.01477888,  0.46681795,  0.10938847, -0.94721852, -0.59555263,
                  0.43174897, -0.11157278,  0.60482401, -0.58120172,  0.82476071,
                 -0.67585093,  0.51747489,  0.13147537,  0.14427685,  0.79576564,
                 -0.81762747,  0.70197738, -0.81513605,  0.97909284,  0.65293123,
                 -0.67827731,  0.37949073, -0.69797135, -0.8226879 , -0.79980223,
                 -0.73445726, -0.67552438,  0.49271431, -0.68601626, -0.23525618,
                 -0.96835393, -0.22108029, -0.30339673, -0.30893032, -0.10465462,
                  0.6285398 , -0.96195274,  0.80812673, -0.49652576, -0.04923682,
                  0.41848055,  0.54361849, -0.00747832, -0.86391432, -0.14825956,
                 -0.97948923,  0.14045395,  0.1672885 , -0.25575461, -0.81473666,
                  0.86115452, -0.10080629,  0.69078833, -0.4371216 ,  0.7291862 ,
                  0.99090707, -0.07503235,  0.9668144 , -0.07102907,  0.90861586])
```

**78. 计算点 p 到各个直线 i 的距离, 其中直线由 (P0[i]，P1[i]) 表示，P0，P1 为一系列对应的点**

```
In [145]: def distance(P0, P1, p):
              T = P1 - P0
              L = (T**2).sum(axis=1)
              U = -((P0[:,0]-p[...,0])*T[:,0] + (P0[:,1]-p[...,1])*T[:,1]) / L
              U = U.reshape(len(U),1)
```

```
    D = P0 + U*T - p
    return np.sqrt((D**2).sum(axis=1))

P0 = np.random.uniform(-10,10,(10,2))
P1 = np.random.uniform(-10,10,(10,2))
p  = np.random.uniform(-10,10,( 1,2))
print(distance(P0, P1, p))
```

```
[ 1.01154163  5.12606935 13.94958511 22.07140895  0.94980679 11.10441586
 10.13624982 15.90978826 12.76884591  7.35750952]
```

**79. 接上题，如何计算 P0 中各点 P0[j] 到各直线（P0[i], P1[i]）的距离**

```
In [146]: # based on distance function from previous question
          P0 = np.random.uniform(-10, 10, (10,2))
          P1 = np.random.uniform(-10,10,(10,2))
          p = np.random.uniform(-10, 10, (10,2))
          print(np.array([distance(P0,P1,p_i) for p_i in p]))
```

```
[[ 3.5734595   0.66180336  7.3187876   4.99413157  0.96165362  1.50776605
  15.24621075  5.09372975  5.79926051  2.37223415]
 [ 9.65216196  3.90422416  0.02454316  5.90288419  6.93066482  5.33370707
   4.85444051  7.35964798  7.41460408  6.10650993]
 [ 3.0170691   9.17194136  4.03658393  0.93905036 11.63440516  9.62787647
   2.93837077  6.05884677 13.55270024  0.66358603]
 [ 6.64712401  9.12738601  5.142417    5.57297746 12.16615989 10.53392643
   0.31688742 10.17275773 12.38143328  2.29469197]
 [ 1.29889259  9.25138092  3.58160414  1.21483096 11.44657219  9.26597142
   4.10370268  4.18161798 14.15205666  2.07913113]
 [ 0.9143175   0.64478816  6.63477487  7.52739507  1.96629743  0.70986043
  15.51887126  6.57975813  7.63913062  0.12958737]
 [14.36169595  8.51086264 13.08230922  3.56073338  5.90285668  7.66843812
  17.43569283  2.12028403  3.62911512 13.19184398]
 [14.54713139  1.95977899  0.74333133 10.87761572  5.58552605  4.39352621
   3.75662835 10.64759725  4.38286275 10.59080599]
 [ 8.93133026  5.51552187  1.63519872  6.06603912  8.57874172  6.99229651
   3.30246759  8.46418042  8.88224559  5.09797049]
```

```
[10.31880276  1.36013315  5.98582952  3.20046907  1.2761327   0.53036216
 11.17109363  1.81264107  3.14840606  8.03104193]]
```

**80. 给定任意一个数组，编写一个函数，接受数组和一个元素为参数，返回以元素为中心的子集（必要的时候可以进行填充）**

In [147]:
```
# Author: Nicolas Rougier

Z = np.random.randint(0,10,(10,10))
shape = (5,5)
fill  = 0
position = (1,1)

R = np.ones(shape, dtype=Z.dtype)*fill
P  = np.array(list(position)).astype(int)
Rs = np.array(list(R.shape)).astype(int)
Zs = np.array(list(Z.shape)).astype(int)

R_start = np.zeros((len(shape),)).astype(int)
R_stop  = np.array(list(shape)).astype(int)
Z_start = (P-Rs//2)
Z_stop  = (P+Rs//2)+Rs%2

R_start = (R_start - np.minimum(Z_start,0)).tolist()
Z_start = (np.maximum(Z_start,0)).tolist()
R_stop = np.maximum(R_start, (R_stop - np.maximum(Z_stop-Zs,0))).tolist()
Z_stop = (np.minimum(Z_stop,Zs)).tolist()

r = [slice(start,stop) for start,stop in zip(R_start,R_stop)]
z = [slice(start,stop) for start,stop in zip(Z_start,Z_stop)]
R[r] = Z[z]
print(Z)
print(R)
```

```
[[7 9 1 5 1 6 6 3 7 6]
 [9 5 1 4 5 8 5 4 4 7]
 [3 8 4 7 5 1 5 6 0 9]
```

```
 [2 9 9 5 4 1 1 4 7 8]
 [4 2 8 8 1 4 0 4 5 3]
 [2 9 5 0 4 6 9 6 9 0]
 [2 0 0 3 1 1 4 5 9 2]
 [1 4 4 7 2 7 1 5 6 3]
 [3 3 4 6 6 1 9 8 4 1]
 [7 9 2 1 8 9 7 7 4 5]]
[[0 0 0 0 0]
 [0 7 9 1 5]
 [0 9 5 1 4]
 [0 3 8 4 7]
 [0 2 9 9 5]]
```

```
/home/shensir/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:27: FutureWarning: Us
```

PS: 感觉难度为三星的题目（64 题及之后）很多出的不太好... 考察的是更加灵活地运用 Numpy，逻辑是没问题，但是缺乏具体的示例，没有依托实际的问题，就显得比较空洞 Orz

**81. 有数组 Z = [1,2,3,4,5,6,7,8,9,10,11,12,13,14]，如何生成 = [[1,2,3,4], [2,3,4,5], [3,4,5,6], ..., [11,12,13,14]] ?**

```
In [148]: # 1, 答案的方法
          Z = np.arange(1,15,dtype=np.uint32)
          R = stride_tricks.as_strided(Z,(11,4),(4,4))
          print(R)
```

```
[[ 1  2  3  4]
 [ 2  3  4  5]
 [ 3  4  5  6]
 [ 4  5  6  7]
 [ 5  6  7  8]
 [ 6  7  8  9]
 [ 7  8  9 10]
 [ 8  9 10 11]
 [ 9 10 11 12]
 [10 11 12 13]
```

```
 [11 12 13 14]]
```

```
In [149]: # 2,  76 题的一个特殊形式
          z = np.arange(1, 15)
          rolling(z, 4)
```

```
Out[149]: array([[ 1,  2,  3,  4],
                 [ 2,  3,  4,  5],
                 [ 3,  4,  5,  6],
                 [ 4,  5,  6,  7],
                 [ 5,  6,  7,  8],
                 [ 6,  7,  8,  9],
                 [ 7,  8,  9, 10],
                 [ 8,  9, 10, 11],
                 [ 9, 10, 11, 12],
                 [10, 11, 12, 13],
                 [11, 12, 13, 14]])
```

**82. 计算矩阵的秩**

```
In [150]: # 1, 答案的方法
          Z = np.random.uniform(0,1,(10,10))
          U, S, V = np.linalg.svd(Z) # Singular Value Decomposition
          rank = np.sum(S > 1e-10)
          print(rank)
```

```
10
```

```
In [151]: # 2, 调用 API
          np.linalg.matrix_rank(Z)
```

```
Out[151]: 10
```

**83. 找出数组中的众数**

```
In [152]: Z = np.random.randint(0,10,50)
          Z
```

```
Out[152]: array([1, 8, 0, 9, 3, 8, 0, 5, 5, 3, 8, 4, 2, 1, 6, 1, 7, 5, 2, 3, 3, 7,
                 0, 6, 0, 4, 2, 0, 9, 0, 1, 7, 5, 7, 9, 3, 7, 5, 0, 2, 9, 1, 7, 5,
                 0, 9, 2, 2, 2, 0])

In [153]: print(np.bincount(Z).argmax())

0
```

如果不限制在 Numpy 之内，我们可以直接调用 Scipy 提供的 API

```
In [154]: from scipy import stats
          stats.mode(Z)

Out[154]: ModeResult(mode=array([0]), count=array([9]))
```

**84. 从 10x10 的矩阵中抽取出所有的 3x3 矩阵**

```
In [155]: Z = np.random.randint(0,5,(10,10))
          n = 3
          i = 1 + (Z.shape[0]-3)
          j = 1 + (Z.shape[1]-3)
          C = stride_tricks.as_strided(Z, shape=(i, j, n, n), strides=Z.strides + Z.strides)
          print(C)

[[[[1 0 3]
   [4 4 1]
   [4 1 4]]

  [[0 3 1]
   [4 1 1]
   [1 4 0]]

  [[3 1 3]
   [1 1 2]
   [4 0 4]]

  [[1 3 3]
   [1 2 2]
   [0 4 2]]
```

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

 [[3 2 0]
  [2 1 1]
  [2 1 0]]

 [[2 0 1]
  [1 1 1]
  [1 0 1]]

 [[0 1 2]
  [1 1 1]
  [0 1 4]]]


[[[4 4 1]
  [4 1 4]
  [1 1 1]]

 [[4 1 1]
  [1 4 0]
  [1 1 1]]

 [[1 1 2]
  [4 0 4]
  [1 1 1]]

 [[1 2 2]
  [0 4 2]
  [1 1 4]]

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

```
 [[2 1 1]
  [2 1 0]
  [4 4 0]]

 [[1 1 1]
  [1 0 1]
  [4 0 1]]

 [[1 1 1]
  [0 1 4]
  [0 1 4]]]


[[[4 1 4]
  [1 1 1]
  [4 4 3]]

 [[1 4 0]
  [1 1 1]
  [4 3 3]]

 [[4 0 4]
  [1 1 1]
  [3 3 1]]

 [[0 4 2]
  [1 1 4]
  [3 1 4]]

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

 [[2 1 0]
  [4 4 0]
  [4 4 0]]
```

```
 [[1 0 1]
  [4 0 1]
  [4 0 1]]

 [[0 1 4]
  [0 1 4]
  [0 1 0]]]


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

 [[1 1 1]
  [4 3 3]
  [3 3 1]]

 [[1 1 1]
  [3 3 1]
  [3 1 0]]

 [[1 1 4]
  [3 1 4]
  [1 0 0]]

 [[1 4 4]
  [1 4 4]
  [0 0 2]]

 [[4 4 0]
  [4 4 0]
  [0 2 3]]

 [[4 0 1]
  [4 0 1]
  [2 3 0]]
```

```
 [[0 1 4]
  [0 1 0]
  [3 0 3]]]


[[[4 4 3]
  [2 3 3]
  [4 0 0]]

 [[4 3 3]
  [3 3 1]
  [0 0 2]]

 [[3 3 1]
  [3 1 0]
  [0 2 1]]

 [[3 1 4]
  [1 0 0]
  [2 1 4]]

 [[1 4 4]
  [0 0 2]
  [1 4 1]]

 [[4 4 0]
  [0 2 3]
  [4 1 4]]

 [[4 0 1]
  [2 3 0]
  [1 4 1]]

 [[0 1 0]
  [3 0 3]
  [4 1 0]]]
```

```
[[[2 3 3]
  [4 0 0]
  [4 0 2]]

 [[3 3 1]
  [0 0 2]
  [0 2 2]]

 [[3 1 0]
  [0 2 1]
  [2 2 2]]

 [[1 0 0]
  [2 1 4]
  [2 2 4]]

 [[0 0 2]
  [1 4 1]
  [2 4 4]]

 [[0 2 3]
  [4 1 4]
  [4 4 1]]

 [[2 3 0]
  [1 4 1]
  [4 1 0]]

 [[3 0 3]
  [4 1 0]
  [1 0 3]]]


[[[4 0 0]
  [4 0 2]
```

```
  [1 4 0]]

 [[0 0 2]
  [0 2 2]
  [4 0 3]]

 [[0 2 1]
  [2 2 2]
  [0 3 3]]

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

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

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

 [[1 4 1]
  [4 1 0]
  [1 2 2]]

 [[4 1 0]
  [1 0 3]
  [2 2 0]]]


[[[4 0 2]
  [1 4 0]
  [2 1 4]]

 [[0 2 2]
  [4 0 3]
```

```
 [1 4 3]]

[[2 2 2]
 [0 3 3]
 [4 3 4]]

[[2 2 4]
 [3 3 4]
 [3 4 0]]

[[2 4 4]
 [3 4 1]
 [4 0 2]]

[[4 4 1]
 [4 1 2]
 [0 2 3]]

[[4 1 0]
 [1 2 2]
 [2 3 2]]

[[1 0 3]
 [2 2 0]
 [3 2 4]]]
```

**85. 构造二维数组的子类，使得 Z[i, j] = Z[j, i]**

```
In [156]: # Author: Eric O. Lebigot
          # Note: only works for 2d array and value setting using indices

          class Symetric(np.ndarray):
              def __setitem__(self, index, value):
                  i,j = index
                  super(Symetric, self).__setitem__((i,j), value)
                  super(Symetric, self).__setitem__((j,i), value)
```

```
        def symetric(Z):
            return np.asarray(Z + Z.T - np.diag(Z.diagonal())).view(Symetric)

        S = symetric(np.random.randint(0,10,(5,5)))
        S[2,3] = 42
        print(S)
```

```
[[ 8 18 10 16 17]
 [18  7  5 14  8]
 [10  5  2 42  4]
 [16 14 42  6 13]
 [17  8  4 13  0]]
```

**86. 给定 p 个 (n, n) 矩阵和 p 个 (n, 1) 向量，计算张量乘法 (tensor product)**

```
In [157]: p, n = 10, 20
          M = np.ones((p,n,n))
          V = np.ones((p,n,1))
          S = np.tensordot(M, V, axes=[[0, 2], [0, 1]])
          print(S)
```

```
[[200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
```

```
 [200.]
 [200.]
 [200.]
 [200.]]
```

### 87. 给定 16x16 的数组，将其分成 4x4 的小块，求每块的和

```
In [158]: Z = np.ones((16,16))
          k = 4
          S = np.add.reduceat(np.add.reduceat(Z, np.arange(0, Z.shape[0], k), axis=0),
                                                 np.arange(0, Z.shape[1], k), axis=1)
          print(S)
```

```
[[16. 16. 16. 16.]
 [16. 16. 16. 16.]
 [16. 16. 16. 16.]
 [16. 16. 16. 16.]]
```

### 88. 用数组实现生存游戏

```
In [159]: def iterate(Z):
              # Count neighbours
              N = (Z[0:-2,0:-2] + Z[0:-2,1:-1] + Z[0:-2,2:] +
                   Z[1:-1,0:-2]                + Z[1:-1,2:] +
                   Z[2:  ,0:-2] + Z[2:  ,1:-1] + Z[2:  ,2:])

              # Apply rules
              birth = (N==3) & (Z[1:-1,1:-1]==0)
              survive = ((N==2) | (N==3)) & (Z[1:-1,1:-1]==1)
              Z[...] = 0
              Z[1:-1,1:-1][birth | survive] = 1
              return Z

          Z = np.random.randint(0,2,(50,50))
          for i in range(100): Z = iterate(Z)
          print(Z)
```

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

### 89. 获取数组最大的 n 个值

```
In [160]: Z = np.arange(10000)
          np.random.shuffle(Z)
          n = 5

In [161]: # 较慢
          print (Z[np.argsort(Z)[-n:]])
```

```
[9995 9996 9997 9998 9999]
```

```
In [162]: # 较快
          print (Z[np.argpartition(-Z,n)[:n]])
```

```
[9999 9997 9998 9996 9995]
```

### 90. 给定任意大小的数组，计算其笛卡尔积

```
In [163]: def cartesian(arrays):
              arrays = [np.asarray(a) for a in arrays]
              shape = (len(x) for x in arrays)

              ix = np.indices(shape, dtype=int)
              ix = ix.reshape(len(arrays), -1).T

              for n, arr in enumerate(arrays):
                  ix[:, n] = arrays[n][ix[:, n]]

              return ix
```

```
print (cartesian(([1, 2, 3], [4, 5], [6, 7])))
```

```
[[1 4 6]
 [1 4 7]
 [1 5 6]
 [1 5 7]
 [2 4 6]
 [2 4 7]
 [2 5 6]
 [2 5 7]
 [3 4 6]
 [3 4 7]
 [3 5 6]
 [3 5 7]]
```

**91. 将一般的数组转化为结构化数组**

```
In [164]: Z = np.array([("Hello", 2.5, 3),
                        ("World", 3.6, 2)])
          Z

Out[164]: array([['Hello', '2.5', '3'],
                 ['World', '3.6', '2']], dtype='<U5')

In [165]: R = np.core.records.fromarrays(Z.T,
                                         names='col1, col2, col3',
                                         formats = 'S8, f8, i8')
          R

Out[165]: rec.array([(b'Hello', 2.5, 3), (b'World', 3.6, 2)],
                    dtype=[('col1', 'S8'), ('col2', '<f8'), ('col3', '<i8')])
```

**92. 用三种方法计算大向量的三次方**

```
In [166]: x = np.random.rand(int(5e7))

In [167]: %timeit np.power(x,3)
```

```
4.07 s ś 79.1 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)
```

```
In [168]: %timeit x*x*x
```

```
230 ms ś 3.23 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)
```

```
In [169]: %timeit np.einsum('i,i,i->i',x,x,x)
```

```
316 ms ś 16.9 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)
```

**93. 考虑形状（8,3）和（2,2）的两个数组 A 和 B. 如何查找包含 B 的每一行元素的 A 行，不考虑 B 中元素的顺序**

```
In [170]: A = np.random.randint(0,5,(8,3))
          B = np.random.randint(0,5,(2,2))

          C = (A[..., np.newaxis, np.newaxis] == B)
          rows = np.where(C.any((3,1)).all(1))[0]
          print(rows)
```

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

**94. 给定（10，3）的矩阵，找出包含不同元素的行**

```
In [171]: Z = np.random.randint(0,5,(10,3))
          Z
```

```
Out[171]: array([[2, 3, 4],
                 [0, 2, 2],
                 [0, 1, 1],
                 [2, 2, 3],
                 [0, 3, 0],
                 [4, 2, 2],
                 [0, 2, 4],
                 [4, 2, 1],
                 [1, 0, 3],
                 [0, 0, 0]])
```

```
In [172]: # 适用于任意数据类型
          E = np.all(Z[:,1:] == Z[:,:-1], axis=1)
          U = Z[~E]
          U

Out[172]: array([[2, 3, 4],
                 [0, 2, 2],
                 [0, 1, 1],
                 [2, 2, 3],
                 [0, 3, 0],
                 [4, 2, 2],
                 [0, 2, 4],
                 [4, 2, 1],
                 [1, 0, 3]])

In [173]: # 仅适用于数值类型
          U = Z[Z.max(axis=1) != Z.min(axis=1),:]
          U

Out[173]: array([[2, 3, 4],
                 [0, 2, 2],
                 [0, 1, 1],
                 [2, 2, 3],
                 [0, 3, 0],
                 [4, 2, 2],
                 [0, 2, 4],
                 [4, 2, 1],
                 [1, 0, 3]])
```

**95. 将给定的整数向量转化为二进制矩阵**

```
In [174]: I = np.array([0, 1, 2, 3, 15, 16, 32, 64, 128], dtype=np.uint8)
          print(np.unpackbits(I[:, np.newaxis], axis=1))

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

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

**96. 给定二维数组，抽取所有不同的行**

```
In [175]: Z = np.random.randint(0,2,(6,3))
          Z

Out[175]: array([[0, 0, 0],
                 [0, 1, 1],
                 [0, 0, 0],
                 [1, 1, 1],
                 [0, 1, 0],
                 [0, 0, 0]])

In [176]: uZ = np.unique(Z, axis=0)
          uZ

Out[176]: array([[0, 0, 0],
                 [0, 1, 0],
                 [0, 1, 1],
                 [1, 1, 1]])
```

**97. 考虑两个数组 A，B，使用 np.einsum 写出矩阵间的 outer, inner, sum, mul 函数**

```
In [177]: A = np.random.uniform(0,1,10)
          B = np.random.uniform(0,1,10)

          np.einsum('i->', A)       # np.sum(A)
          np.einsum('i,i->i', A, B) # A * B
          np.einsum('i,i', A, B)    # np.inner(A, B)
          np.einsum('i,j->ij', A, B)    # np.outer(A, B)

Out[177]: array([[0.05623017, 0.0435339 , 0.06410929, 0.00515166, 0.04477097,
                  0.00687832, 0.01534161, 0.03621964, 0.02564694, 0.01010714],
                 [0.25008018, 0.19361432, 0.28512209, 0.02291169, 0.19911611,
                  0.03059089, 0.06823086, 0.16108458, 0.11406317, 0.04495087],
```

```
       [0.37356092, 0.28921422, 0.42590529, 0.03422467, 0.29743259,
        0.04569558, 0.10192084, 0.24062245, 0.17038353, 0.06714602],
       [0.20315391, 0.15728358, 0.23162038, 0.01861243, 0.16175298,
        0.02485066, 0.05542769, 0.13085788, 0.0926598 , 0.03651607],
       [0.32736782, 0.25345111, 0.37323949, 0.02999258, 0.26065323,
        0.04004504, 0.08931771, 0.21086801, 0.14931456, 0.05884301],
       [0.45291908, 0.35065402, 0.51638333, 0.04149526, 0.36061828,
        0.05540302, 0.1235726 , 0.29173956, 0.20657929, 0.08141032],
       [0.1605851 , 0.12432643, 0.18308672, 0.01471239, 0.12785931,
        0.01964346, 0.04381338, 0.10343796, 0.07324389, 0.0288645 ],
       [0.39477812, 0.30564078, 0.45009551, 0.03616854, 0.31432592,
        0.04829096, 0.10770966, 0.25428912, 0.18006083, 0.07095973],
       [0.54263895, 0.42011596, 0.61867499, 0.04971516, 0.43205405,
        0.06637794, 0.1480514 , 0.34953097, 0.2475011 , 0.0975371 ],
       [0.38241351, 0.29606798, 0.43599833, 0.03503572, 0.3044811 ,
        0.04677847, 0.10433615, 0.24632468, 0.17442125, 0.06873724]])
```

**98. 用两个向量 (X, Y) 描述一条轨道，如何对其进行等距抽样**

```
In [178]: phi = np.arange(0, 10*np.pi, 0.1)
          a = 1
          x = a*phi*np.cos(phi)
          y = a*phi*np.sin(phi)

          dr = (np.diff(x)**2 + np.diff(y)**2)**.5 # segment lengths
          r = np.zeros_like(x)
          r[1:] = np.cumsum(dr)                # integrate path
          r_int = np.linspace(0, r.max(), 200) # regular spaced path
          x_int = np.interp(r_int, r, x)       # integrate path
          y_int = np.interp(r_int, r, y)
```

**99. 给定一个二维数组和一个整数 n，提取所有仅包含整数，且元素和为 n 的行**

```
In [179]: X = np.asarray([[1.0, 0.0, 3.0, 8.0],
                          [2.0, 0.0, 1.0, 1.0],
                          [1.5, 2.5, 1.0, 0.0]])
          n = 4
          M = np.logical_and.reduce(np.mod(X, 1) == 0, axis=-1)
```

```
        M &= (X.sum(axis=-1) == n)
        print(X[M])
```

```
[[2. 0. 1. 1.]]
```

**100. 给定一个向量，计算其均值的 95% 的置信区间**

```
In [180]: X = np.random.randn(100)
          N = 1000  # 再抽样次数
          idx = np.random.randint(0, X.size, (N, X.size))
          means = X[idx].mean(axis=1)
          confint = np.percentile(means, [2.5, 97.5])
          print(confint)
```

```
[-0.29266392  0.11041998]
```

```
In [ ]:
```