100 Numpy Exercises

小小白

2019 年 7 月 28 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)]

331 ţs ś 11.3 ţs per loop (mean ś std. dev. of 7 runs, 1000 loops each)


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

5.01 ţs ś 254 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([[[0.22054713, 0.01791274, 0.3550555 ],
                 [0.20214311, 0.00522642, 0.62212663],
                 [0.80310102, 0.78426245, 0.64338167]],

                [[0.04983986, 0.99051989, 0.96583312],
                 [0.67899828, 0.40544719, 0.2075603 ],
                 [0.53011682, 0.41751142, 0.49965018]],

                [[0.07533365, 0.54417015, 0.73985242],
                 [0.21655224, 0.52242945, 0.14863604],
                 [0.9944412 , 0.52479639, 0.70733954]]])
```

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([[0.49654847, 0.00219032, 0.01928795, 0.68354638, 0.25143502,
                 0.87047557, 0.33418658, 0.08344583, 0.14716511, 0.66685776],
                [0.84700908, 0.17645969, 0.80250215, 0.83365758, 0.62144565,
                 0.14628858, 0.52633805, 0.65990219, 0.15255923, 0.88613291],
                [0.18785017, 0.96808259, 0.92771838, 0.31652174, 0.19515929,
                 0.62583915, 0.11111072, 0.93796816, 0.12287252, 0.06630076],
                [0.73253763, 0.05073095, 0.46410998, 0.68652359, 0.7942978 ,
                 0.40258988, 0.94901401, 0.10250282, 0.52754037, 0.32848858],
                [0.43608491, 0.29603079, 0.46594833, 0.81236375, 0.56355187,
                 0.44058517, 0.54252728, 0.97368373, 0.26206988, 0.88193702],
                [0.04931272, 0.0692593 , 0.51130276, 0.69347158, 0.63110598,
                 0.97391473, 0.44107031, 0.8887    , 0.23145875, 0.65929483],
                [0.74542968, 0.06126814, 0.27929092, 0.84068879, 0.06739453,
                 0.25118233, 0.20734746, 0.943424  , 0.20121617, 0.24301881],
                [0.21579846, 0.77548184, 0.41855549, 0.70742247, 0.03775403,
                 0.01837671, 0.82136684, 0.09007986, 0.08984528, 0.45901313],
                [0.05431855, 0.23788422, 0.05581697, 0.83253138, 0.42118333,
```

```
                0.20588476, 0.76396372, 0.34261765, 0.71292646, 0.80738538],
               [0.3010015 , 0.27702542, 0.12476186, 0.19323018, 0.2650162 ,
                0.10140178, 0.53426627, 0.20364003, 0.35542423, 0.52218861]])
```

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

```
Out[18]: (0.9739147305275095, 0.002190315643831653)
```

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([0.27282741, 0.13786767, 0.31249929, 0.23900839, 0.07780329,
                0.65559556, 0.31410045, 0.12355781, 0.67591137, 0.84324563])
```

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

```
Out[20]: 0.3652416851133987
```

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([[ 0.95554048, -0.39443724, -1.41679362,  1.49485148,  0.30409986],
                [ 0.4912484 ,  0.60577121, -0.95613421,  0.55744299,  0.99533824],
                [-1.41217036, -0.37359879,  1.19692679,  0.4485745 , -0.87167414],
                [ 1.78892886, -0.81031419,  0.08107744, -1.56219798,  0.86303453],
                [ 1.37452144, -0.78100091, -0.8490149 , -1.35505593, -0.37496397]])
```

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 [52]: # 6
         z < z > z


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

         ValueError                                Traceback (most recent call last)

         <ipython-input-52-856b68e674c4> 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 [ ]: l1 = [1, 2]
        l1 < l1 > l1
```

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

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 [ ]: z < z
```

```
In [ ]: z > z
```

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 all，即所有的元素全是True才判定为True，否则为False；另外一种方法是any，即只要数组中有一个True，我们就判定其为True，否则判定为False. 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 [ ]: np.array(0) / np.array(0)
```

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

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

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

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

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

根据文档就可以直接使用了。

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

31. 如何忽视所有 Numpy 的警告（不推荐）

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

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

32. 下面的表达式会返回 True 吗

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

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

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

33. 如何获取昨天，今天，明天的日期

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

```
In [ ]: yesterday, today, tomorrow
```

34. 如何获取 2016 年 7 月全部 31 天的日期

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

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

```
In [ ]: 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)
```

36. 用 5 种方法提取随机数组中的整数部分

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

In [ ]: # 1
        z - z % 1

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

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

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

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

37. 创建一个 5x5 的矩阵，每行均为 0 到 4

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

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

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

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

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

39. 创建一个长度为 10，范围从 0 到 1 的向量（不包括 0，1）

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

40. 创建一个长度为 10 的随机数组并排序

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

```
In [ ]: z.sort()
        z
```

41. 对于长度较小的数组，如何更高效地求和（相对 np.sum）

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

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

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

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

In [ ]: # 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)

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

43. 限制数组为不可变数组（read only）

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

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

In [ ]: z[0] = 1
```

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 [ ]: 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
```

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 [ ]: z = np.random.random(10)
        z

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

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

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

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

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

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

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

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

```
In [ ]: 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}")
```

49. 打印数组所有元素（不省略）

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

50. 给定一个数，在数组中找出距离其最近的数

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

In [ ]: # 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]
```

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

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 [ ]: 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
```

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 [ ]: import scipy
        import scipy.spatial

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

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

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

54. 如何读取下面的文件

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

```
In [ ]: 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
```

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

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

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

56. 生成二维高斯分布

```
In [ ]: 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
```

57. 随机地在二维数组中放置 p 个元素

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

58. 矩阵每行进行中心化（减去均值）

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

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

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

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

59. 根据某列数据来排列数组

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

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

60. 判断二维数组是否含有空列 (全为 0)

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

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

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 [ ]: 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])
```

63. 创建一个带有名称属性的数组类

```
In [ ]: 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)
```

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 [ ]: # 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
```

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

65. 根据索引列表 I，对数值列表 X 进行累加，得到 F

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

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 [ ]: # Author: Nadav Horesh


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

```
In [ ]: # 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)
```

67. 考虑一个四维数组，计算后两个轴上的元素和

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

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 [ ]: 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)
```

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 [ ]: # 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)
```

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

```
In [ ]: 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
```

71. 维度分别为（5，5，3），（5，5）的两个数组相乘

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

72. 交换数组的两行

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

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

73. 给定 10 个三元组描述 10 个三角形，找出所有边的集合

```
In [ ]: 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)
```

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

75. 用滑动窗口计算平均值

```
In [ ]: 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))
```

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 [ ]: 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)
```

77. 如何原地对布尔值取反，如何原地改变数字的正负

```
In [ ]: 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)
```

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 [ ]: 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))
```

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 [ ]: # 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]))
```

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 [ ]: # 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)
```

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 [ ]: # 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)
```

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

82. 计算矩阵的秩

```
In [ ]: # 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)
```

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

83. 找出数组中的众数

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

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

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

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

84. 从 10x10 的矩阵中抽取出所有的 3x3 矩阵

```
In [ ]: 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)
```

85. 构造二维数组的子类，使得 Z[i, j] = Z[j, i]

```
In [ ]: # 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)
```

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 [ ]: 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)
```

87. 给定 16x16 的数组，将其分成 4x4 的小块，求每块的和

```
In [ ]: 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)
```

88. 用数组实现生存游戏

```
In [ ]: 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)
```

89. 获取数组最大的 n 个值

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

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

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

90. 给定任意大小的数组，计算其笛卡尔积

```
In [ ]: 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])))
```

91. 将一般的数组转化为结构化数组

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

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

92. 用三种方法计算大向量的三次方

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

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

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

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

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 [ ]: 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)
```

94. 给定（10，3）的矩阵，找出包含不同元素的行

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

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

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

95. 将给定的整数向量转化为二进制矩阵

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

96. 给定二维数组，抽取所有不同的行

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

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

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 [ ]: 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)
```

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 [ ]: 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 [ ]: 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])
```

100. 给定一个向量，计算其均值的 95% 的置信区间

```
In [ ]: 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)
```

```
In [ ]:
```