numpy1

2018 年 2 月 1 日

1 9.

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

```
In [2]: world_alcohol = numpy.genfromtxt("world_alcohol.txt",delimiter = ",",dtype = str)
        print(type(world_alcohol))
        print(world_alcohol)
        print(help(numpy.genfromtxt))
```

```
<class 'numpy.ndarray'>
[['Year' 'WHO region' 'Country' 'Beverage Types' 'Display Value']
 ['1986' 'Western Pacific' 'Viet Nam' 'Wine' '0']
 ['1986' 'Americas' 'Uruguay' 'Other' '0.5']
 ...
 ['1987' 'Africa' 'Malawi' 'Other' '0.75']
 ['1989' 'Americas' 'Bahamas' 'Wine' '1.5']
 ['1985' 'Africa' 'Malawi' 'Spirits' '0.31']]
Help on function genfromtxt in module numpy.lib.npyio:

genfromtxt(fname, dtype=<class 'float'>, comments='#', delimiter=None, skip_header=0, skip_footer=
    Load data from a text file, with missing values handled as specified.

    Each line past the first `skip_header` lines is split at the `delimiter`
    character, and characters following the `comments` character are discarded.

    Parameters
    ----------
    fname : file, str, pathlib.Path, list of str, generator
        File, filename, list, or generator to read.  If the filename
        extension is `.gz` or `.bz2`, the file is first decompressed. Note
        that generators must return byte strings in Python 3k.  The strings
        in a list or produced by a generator are treated as lines.
    dtype : dtype, optional
        Data type of the resulting array.
        If None, the dtypes will be determined by the contents of each
        column, individually.
    comments : str, optional
        The character used to indicate the start of a comment.
        All the characters occurring on a line after a comment are discarded
    delimiter : str, int, or sequence, optional
        The string used to separate values.  By default, any consecutive
        whitespaces act as delimiter.  An integer or sequence of integers
        can also be provided as width(s) of each field.
    skiprows : int, optional
        `skiprows` was removed in numpy 1.10. Please use `skip_header` instead.
    skip_header : int, optional
        The number of lines to skip at the beginning of the file.
    skip_footer : int, optional
        The number of lines to skip at the end of the file.
    converters : variable, optional
        The set of functions that convert the data of a column to a value.
        The converters can also be used to provide a default value
        for missing data: ``converters = {3: lambda s: float(s or 0)}``.
    missing : variable, optional
        `missing` was removed in numpy 1.10. Please use `missing_values`
        instead.
    missing_values : variable, optional
        The set of strings corresponding to missing data.
    filling_values : variable, optional
        The set of values to be used as default when the data are missing.
    usecols : sequence, optional
        Which columns to read, with 0 being the first.  For example,
        ``usecols = (1, 4, 5)`` will extract the 2nd, 5th and 6th columns.
    names : {None, True, str, sequence}, optional
        If `names` is True, the field names are read from the first line after
        the first `skip_header` lines.  This line can optionally be proceeded
        by a comment delimeter. If `names` is a sequence or a single-string of
        comma-separated names, the names will be used to define the field names
        in a structured dtype. If `names` is None, the names of the dtype
        fields will be used, if any.
```

```
excludelist : sequence, optional
    A list of names to exclude. This list is appended to the default list
    ['return','file','print']. Excluded names are appended an underscore:
    for example, `file` would become `file_`.
deletechars : str, optional
    A string combining invalid characters that must be deleted from the
    names.
defaultfmt : str, optional
    A format used to define default field names, such as "f%i" or "f_%02i".
autostrip : bool, optional
    Whether to automatically strip white spaces from the variables.
replace_space : char, optional
    Character(s) used in replacement of white spaces in the variables
    names. By default, use a '_'.
case_sensitive : {True, False, 'upper', 'lower'}, optional
    If True, field names are case sensitive.
    If False or 'upper', field names are converted to upper case.
    If 'lower', field names are converted to lower case.
unpack : bool, optional
    If True, the returned array is transposed, so that arguments may be
    unpacked using ``x, y, z = loadtxt(...)``
usemask : bool, optional
    If True, return a masked array.
    If False, return a regular array.
loose : bool, optional
    If True, do not raise errors for invalid values.
invalid_raise : bool, optional
    If True, an exception is raised if an inconsistency is detected in the
    number of columns.
    If False, a warning is emitted and the offending lines are skipped.
max_rows : int,  optional
    The maximum number of rows to read. Must not be used with skip_footer
    at the same time.  If given, the value must be at least 1. Default is
    to read the entire file.

    .. versionadded:: 1.10.0
encoding : str, optional
    Encoding used to decode the inputfile. Does not apply when `fname` is
    a file object.  The special value 'bytes' enables backward compatibility
    workarounds that ensure that you receive byte arrays when possible
    and passes latin1 encoded strings to converters. Override this value to
    receive unicode arrays and pass strings as input to converters.  If set
    to None the system default is used. The default value is 'bytes'.

    .. versionadded:: 1.14.0

Returns
-------
out : ndarray
    Data read from the text file. If `usemask` is True, this is a
    masked array.

See Also
--------
numpy.loadtxt : equivalent function when no data is missing.

Notes
-----
* When spaces are used as delimiters, or when no delimiter has been given
  as input, there should not be any missing data between two fields.
* When the variables are named (either by a flexible dtype or with `names`,
  there must not be any header in the file (else a ValueError
  exception is raised).
* Individual values are not stripped of spaces by default.
  When using a custom converter, make sure the function does remove spaces.

References
----------
.. [1] NumPy User Guide, section `I/O with NumPy
       <http://docs.scipy.org/doc/numpy/user/basics.io.genfromtxt.html>`_.

Examples
---------
>>> from io import StringIO
>>> import numpy as np

Comma delimited file with mixed dtype
```

```
    >>> s = StringIO("1,1.3,abcde")
    >>> data = np.genfromtxt(s, dtype=[('myint','i8'),('myfloat','f8'),
    ... ('mystring','S5')], delimiter=",")
    >>> data
    array((1, 1.3, 'abcde'),
          dtype=[('myint', '<i8'), ('myfloat', '<f8'), ('mystring', '|S5')])

    Using dtype = None

    >>> s.seek(0) # needed for StringIO example only
    >>> data = np.genfromtxt(s, dtype=None,
    ... names = ['myint','myfloat','mystring'], delimiter=",")
    >>> data
    array((1, 1.3, 'abcde'),
          dtype=[('myint', '<i8'), ('myfloat', '<f8'), ('mystring', '|S5')])

    Specifying dtype and names

    >>> s.seek(0)
    >>> data = np.genfromtxt(s, dtype="i8,f8,S5",
    ... names=['myint','myfloat','mystring'], delimiter=",")
    >>> data
    array((1, 1.3, 'abcde'),
          dtype=[('myint', '<i8'), ('myfloat', '<f8'), ('mystring', '|S5')])

    An example with fixed-width columns

    >>> s = StringIO("11.3abcde")
    >>> data = np.genfromtxt(s, dtype=None, names=['intvar','fltvar','strvar'],
    ...     delimiter=[1,3,5])
    >>> data
    array((1, 1.3, 'abcde'),
          dtype=[('intvar', '<i8'), ('fltvar', '<f8'), ('strvar', '|S5')])

None
```

1.1 numpy.array 输入一个矩阵，矩阵输入的每个元素的数据类型必须是一样的

```
In [3]: #The numpy.array() function can take a list or list of lists as input.
        #When we input a list, we get a one-dimensional array as a result:
        vector = numpy.array([5, 10, 15, 20])
        #When we input a list of lists, we get a matrix as a result
        matrix = numpy.array([[5, 10, 15], [20, 25, 30], [35, 40, 45]])

        print (vector)
        print (matrix)
```

```
[ 5 10 15 20]
[[ 5 10 15]
 [20 25 30]
 [35 40 45]]
```

1.2 vector.shape 显示矩阵是几行几列

```
In [4]: #We can use the ndarray.shape property to figure out how many elements are in the array
        vector = numpy.array([1, 2, 3, 4])
        print(vector.shape)
        #For matrices, the shape property contains a tuple with 2 elements.
        matrix = numpy.array([[5, 10, 15], [20, 25, 30]])
        print(matrix.shape)
```

```
(4,)
(2, 3)
```

1.3 numbers.dtype 数据类型

```
In [5]: #Each value in a NumPy array has to have the same data type
        #NumPy will automatically figure out an appropriate data type when reading in data or conv
        #You can check the data type of a NumPy array using the dtype property.
        numbers = numpy.array([1, 2, 3, 4])
        print(numbers)
        numbers.dtype
```

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

```
Out[5]: dtype('int32')
```

1.4 矩阵元素从 0 开始算起，[1,4] 代表第二行第五列的值

```
In [6]: world_alcohol = numpy.genfromtxt("world_alcohol.txt", delimiter=",", dtype="str", skip_hea
        print(world_alcohol)
        uruguay_other_1986 = world_alcohol[1,4]
        third_country = world_alcohol[2,2]
        print (uruguay_other_1986)
        print (third_country)
```

```
[['1986' 'Western Pacific' 'Viet Nam' 'Wine' '0']
 ['1986' 'Americas' 'Uruguay' 'Other' '0.5']
 ['1985' 'Africa' "Cte d'Ivoire" 'Wine' '1.62']
 ...
 ['1987' 'Africa' 'Malawi' 'Other' '0.75']
 ['1989' 'Americas' 'Bahamas' 'Wine' '1.5']
 ['1985' 'Africa' 'Malawi' 'Spirits' '0.31']]
0.5
Cte d'Ivoire
```

1.5 切片 “:” 在矩阵中代表从第 n+1 取到第 m 个

```
In [7]: vector = numpy.array([5, 10, 15, 20])
        print(vector[0:3])

        matrix = numpy.array([[5, 10, 15],[20, 25, 30],[35, 40, 45]])
        print(matrix[:,1])

        matrix = numpy.array([[5, 10, 15],[20, 25, 30],[35, 40, 45]])
        print(matrix[:,0:2])
```

```
[ 5 10 15]
[10 25 40]
[[ 5 10]
 [20 25]
 [35 40]]
```

2 10.

2.1 逻辑判断，布尔值

```
In [8]: #it will compare the second value to each element in the vector
        # If the values are equal, the Python interpreter returns True; otherwise, it returns Fals
        vector = numpy.array([5, 10, 15, 20])
        vector == 10
```

```
Out[8]: array([False,  True, False, False])
```

```
In [9]: matrix = numpy.array([
                            [5, 10, 15],
                            [20, 25, 30],
                            [35, 40, 45]
                         ])
        matrix == 25
```

```
Out[9]: array([[False, False, False],
               [False,  True, False],
               [False, False, False]])
```

```
In [10]: #Compares vector to the value 10, which generates a new Boolean vector [False, True, Fals
         vector = numpy.array([5, 10, 15, 20])
         equal_to_ten = (vector == 10)
         print (equal_to_ten)
         print(vector[equal_to_ten])
```

```
[False  True False False]
[10]
```

```
In [11]: matrix = numpy.array([
                         [5, 10, 15],
                         [20, 25, 30],
                         [35, 40, 45]
                      ])
         second_column_25 = (matrix[:,1] == 25)
         print (second_column_25)
         print(matrix[second_column_25, :])
```

```
[False  True False]
[[20 25 30]]
```

3 11.

3.1 逻辑判断 & 和 | 的运用

```
In [12]: #We can also perform comparisons with multiple conditions
         vector = numpy.array([5, 10, 15, 20])
         equal_to_ten_and_five = (vector == 10) & (vector == 5)
         print (equal_to_ten_and_five)
```

```
[False False False False]
```

```
In [13]: vector = numpy.array([5, 10, 15, 20])
         equal_to_ten_or_five = (vector == 10) | (vector == 5)
         print (equal_to_ten_or_five)
```

```
[ True  True False False]
```

```
In [14]: vector = numpy.array([5, 10, 15, 20])
         equal_to_ten_or_five = (vector == 10) | (vector == 5)
         vector[equal_to_ten_or_five] = 50
         print(vector)
```

```
[50 50 15 20]
```

3.2 修改某值

```
In [15]: matrix = numpy.array([
                     [5, 10, 15],
                     [20, 25, 30],
                     [35, 40, 45]
                  ])
         second_column_25 = matrix[:,1] == 25
         print (second_column_25)
         matrix[second_column_25, 1] = 10
         print (matrix)
```

```
[False  True False]
[[ 5 10 15]
 [20 10 30]
 [35 40 45]]
```

3.3 dtype 修改矩阵数据的值类型

```
In [16]: #We can convert the data type of an array with the ndarray.astype() method.
         vector = numpy.array(["1", "2", "3"])
         print (vector.dtype)
         vector = vector.astype(float)
         print (vector.dtype)
```

```
<U1
float64
```

3.4 矩阵元素求和、最值、均值

```
In [17]: vector = numpy.array([5, 10, 15, 20])
         print (vector.sum())
         print (vector.min())
         print (vector.max())
         print (vector.mean())
```

```
50
5
20
12.5
```

3.5 横向相加求和

```
In [18]: # The axis dictates which dimension we perform the operation on
         #1 means that we want to perform the operation on each row, and 0 means on each column
         matrix = numpy.array([
                         [5, 10, 15],
                         [20, 25, 30],
                         [35, 40, 45]
                      ])
         matrix.sum(axis=1)
```

```
Out[18]: array([ 30,  75, 120])
```

3.6 纵向相加求和

```
In [19]: matrix = numpy.array([
                         [5, 10, 15],
                         [20, 25, 30],
                         [35, 40, 45]
                     ])
      matrix.sum(axis=0)

Out[19]: array([60, 75, 90])
```

4 12

```
In [20]: import numpy as np
```

4.1 生成矩阵

```
In [21]: a = np.arange(15).reshape(3, 5)
         a

Out[21]: array([[ 0,  1,  2,  3,  4],
                [ 5,  6,  7,  8,  9],
                [10, 11, 12, 13, 14]])

In [22]: print (a.shape)
         print (a.ndim)
         print (a.dtype.name)
         print (a.size)

(3, 5)
2
int32
15


In [23]: np.zeros ((3,4))

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

In [24]: np.ones( (2,3,4), dtype=np.int32 )

Out[24]: 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]]])
```

```
In [25]: np.arange( 10, 30, 5 )
```

```
Out[25]: array([10, 15, 20, 25])
```

```
In [26]: np.arange( 0, 2, 0.3 )
```

```
Out[26]: array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])
```

```
In [27]: np.arange(12).reshape(4,3)
```

```
Out[27]: array([[ 0,  1,  2],
                [ 3,  4,  5],
                [ 6,  7,  8],
                [ 9, 10, 11]])
```

```
In [28]: np.random.random((2,3))
```

```
Out[28]: array([[0.26248255, 0.33581691, 0.9170368 ],
                [0.77015327, 0.23354984, 0.87764818]])
```

```
In [29]: from numpy import pi
         np.linspace( 0, 2*pi, 100 )
```

```
Out[29]: array([0.        , 0.06346652, 0.12693304, 0.19039955, 0.25386607,
               0.31733259, 0.38079911, 0.44426563, 0.50773215, 0.57119866,
               0.63466518, 0.6981317 , 0.76159822, 0.82506474, 0.88853126,
               0.95199777, 1.01546429, 1.07893081, 1.14239733, 1.20586385,
               1.26933037, 1.33279688, 1.3962634 , 1.45972992, 1.52319644,
               1.58666296, 1.65012947, 1.71359599, 1.77706251, 1.84052903,
               1.90399555, 1.96746207, 2.03092858, 2.0943951 , 2.15786162,
               2.22132814, 2.28479466, 2.34826118, 2.41172769, 2.47519421,
               2.53866073, 2.60212725, 2.66559377, 2.72906028, 2.7925268 ,
               2.85599332, 2.91945984, 2.98292636, 3.04639288, 3.10985939,
               3.17332591, 3.23679243, 3.30025895, 3.36372547, 3.42719199,
               3.4906585 , 3.55412502, 3.61759154, 3.68105806, 3.74452458,
               3.8079911 , 3.87145761, 3.93492413, 3.99839065, 4.06185717,
               4.12532369, 4.1887902 , 4.25225672, 4.31572324, 4.37918976,
               4.44265628, 4.5061228 , 4.56958931, 4.63305583, 4.69652235,
               4.75998887, 4.82345539, 4.88692191, 4.95038842, 5.01385494,
               5.07732146, 5.14078798, 5.2042545 , 5.26772102, 5.33118753,
               5.39465405, 5.45812057, 5.52158709, 5.58505361, 5.64852012,
               5.71198664, 5.77545316, 5.83891968, 5.9023862 , 5.96585272,
               6.02931923, 6.09278575, 6.15625227, 6.21971879, 6.28318531])
```

```
In [30]: np.sin(np.linspace( 0, 2*pi, 100 ))
```

```
Out[30]: array([ 0.00000000e+00,  6.34239197e-02,  1.26592454e-01,  1.89251244e-01,
                 2.51147987e-01,  3.12033446e-01,  3.71662456e-01,  4.29794912e-01,
                 4.86196736e-01,  5.40640817e-01,  5.92907929e-01,  6.42787610e-01,
                 6.90079011e-01,  7.34591709e-01,  7.76146464e-01,  8.14575952e-01,
                 8.49725430e-01,  8.81453363e-01,  9.09631995e-01,  9.34147860e-01,
                 9.54902241e-01,  9.71811568e-01,  9.84807753e-01,  9.93838464e-01,
                 9.98867339e-01,  9.99874128e-01,  9.96854776e-01,  9.89821442e-01,
                 9.78802446e-01,  9.63842159e-01,  9.45000819e-01,  9.22354294e-01,
                 8.95993774e-01,  8.66025404e-01,  8.32569855e-01,  7.95761841e-01,
                 7.55749574e-01,  7.12694171e-01,  6.66769001e-01,  6.18158986e-01,
                 5.67059864e-01,  5.13677392e-01,  4.58226522e-01,  4.00930535e-01,
                 3.42020143e-01,  2.81732557e-01,  2.20310533e-01,  1.58001396e-01,
                 9.50560433e-02,  3.17279335e-02, -3.17279335e-02, -9.50560433e-02,
                -1.58001396e-01, -2.20310533e-01, -2.81732557e-01, -3.42020143e-01,
                -4.00930535e-01, -4.58226522e-01, -5.13677392e-01, -5.67059864e-01,
                -6.18158986e-01, -6.66769001e-01, -7.12694171e-01, -7.55749574e-01,
                -7.95761841e-01, -8.32569855e-01, -8.66025404e-01, -8.95993774e-01,
                -9.22354294e-01, -9.45000819e-01, -9.63842159e-01, -9.78802446e-01,
                -9.89821442e-01, -9.96854776e-01, -9.99874128e-01, -9.98867339e-01,
                -9.93838464e-01, -9.84807753e-01, -9.71811568e-01, -9.54902241e-01,
                -9.34147860e-01, -9.09631995e-01, -8.81453363e-01, -8.49725430e-01,
                -8.14575952e-01, -7.76146464e-01, -7.34591709e-01, -6.90079011e-01,
                -6.42787610e-01, -5.92907929e-01, -5.40640817e-01, -4.86196736e-01,
                -4.29794912e-01, -3.71662456e-01, -3.12033446e-01, -2.51147987e-01,
                -1.89251244e-01, -1.26592454e-01, -6.34239197e-02, -2.44929360e-16])
```

4.2 矩阵运算

```
In [31]: #the product operator * operates elementwise in NumPy arrays
         a = np.array( [20,30,40,50] )
         b = np.arange( 4 )
         #print a
         #print b
         #b
         c = a-b
         #print c
         b**2
         #print b**2
         print (a<35)
```

```
[ True  True False False]
```

```
In [32]: #The matrix product can be performed using the dot function or method
         A = np.array( [[1,1],
                        [0,1]] )
         B = np.array( [[2,0],
                        [3,4]] )
         #print A
         #print B
         print (A*B)
         print (A.dot(B))
         print (np.dot(A, B) )
```

```
[[2 0]
 [0 4]]
[[5 4]
 [3 4]]
[[5 4]
 [3 4]]
```

5 13.

5.1 exp/sqrt

```
In [33]: import numpy as np
         B = np.arange(3)
         print (B)
         print (np.exp(B))
         print (np.sqrt(B))
```

```
[0 1 2]
[1.         2.71828183 7.3890561 ]
[0.         1.         1.41421356]
```

5.2 ravel/resize

```
In [34]: #Return the floor of the input
         a = np.floor(10*np.random.random((3,4)))
         print (a)
         print ('----------')

         a.shape
         ## flatten the array
         print (a.ravel())
         print ('----------')
         a.shape = (6, 2)
         print (a)
         print ('----------')
         print (a.T)
         print ('----------')
         print (a.resize((2,6)))
         print ('----------')
         print (a)

         #If a dimension is given as -1 in a reshaping operation, the other dimensions are automat
         a.reshape(3,-1)
```

```
[[0. 1. 1. 3.]
 [9. 5. 8. 4.]
 [6. 4. 8. 1.]]
----------
[0. 1. 1. 3. 9. 5. 8. 4. 6. 4. 8. 1.]
----------
[[0. 1.]
 [1. 3.]
 [9. 5.]
 [8. 4.]
 [6. 4.]
 [8. 1.]]
----------
[[0. 1. 9. 8. 6. 8.]
 [1. 3. 5. 4. 4. 1.]]
----------
None
----------
```

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

```
Out[34]: array([[0., 1., 1., 3.],
                [9., 5., 8., 4.],
                [6., 4., 8., 1.]])
```

5.3 hstack/vstack 横向/纵向拼接

```
In [35]: a = np.floor(10*np.random.random((2,2)))
         b = np.floor(10*np.random.random((2,2)))
         print (a)
         print ('----------')
         print (b)
         print ('----------')
         print (np.hstack((a,b)))
         print ('----------')
         print (np.vstack((a,b)))
         #np.hstack((a,b))
```

```
[[8. 4.]
 [1. 8.]]
----------
[[8. 7.]
 [0. 7.]]
----------
[[8. 4. 8. 7.]
 [1. 8. 0. 7.]]
----------
[[8. 4.]
 [1. 8.]
 [8. 7.]
 [0. 7.]]
```

5.4 hsplit/vsplit 横向/纵向切分

```
In [36]: a = np.floor(10*np.random.random((2,12)))
         print (a)
         print ('----------')
         print (np.hsplit(a,3))
         print ('----------')
         print (np.hsplit(a,(3,4)))   # Split a after the third and the fourth column
         print ('----------')
         a = np.floor(10*np.random.random((12,2)))
         print (a)
         np.vsplit(a,3)
```

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

```
Out[36]: [array([[6., 9.],
                 [3., 8.],
                 [1., 9.],
                 [5., 8.]]), array([[3., 1.],
                 [9., 7.],
                 [3., 3.],
                 [2., 7.]]), array([[2., 0.],
                 [2., 8.],
                 [3., 9.],
                 [4., 9.]])]
```

6 14.

6.1 copy

不知道 ID 除了检查两个矩阵是否相同之外还有什么作用

```
In [37]: #Simple assignments make no copy of array objects or of their data.
         a = np.arange(12)
         b = a
         # a and b are two names for the same ndarray object
         b is a
         b.shape = 3,4
         print (a.shape)
         print ('----------')
         print (id(a))
         print ('----------')
         print (id(b))
```

```
(3, 4)
----------
2032967751200
----------
2032967751200
```

```
In [38]: #The view method creates a new array object that looks at the same data.
         c = a.view()
         c is a
         c.shape = 2,6
         #print a.shape
         c[0,4] = 1234
         a
         print ('----------')
         print (id(c))
         print ('----------')
         print (id(b))
```

```
----------
2032967752400
----------
2032967751200
```

```
In [39]: #The copy method makes a complete copy of the array and its data.
         d = a.copy()
         d is a
         d[0,0] = 9999
         print (d)
         print ('----------')
         print (a)
```

```
[[9999    1    2    3]
 [1234    5    6    7]
 [   8    9   10   11]]
----------
[[   0    1    2    3]
 [1234    5    6    7]
 [   8    9   10   11]]
```

```
In [40]: import numpy as np
         data = np.sin(np.arange(20)).reshape(5,4)
         print (data)
         print ('----------')
         ind = data.argmax(axis=0)
         print (ind)
         print ('----------')
         data_max = data[ind, range(data.shape[1])]
         print (data_max)
         print ('----------')
         all(data_max == data.max(axis=0))
```

```
[[ 0.          0.84147098  0.90929743  0.14112001]
 [-0.7568025  -0.95892427 -0.2794155   0.6569866 ]
 [ 0.98935825  0.41211849 -0.54402111 -0.99999021]
 [-0.53657292  0.42016704  0.99060736  0.65028784]
 [-0.28790332 -0.96139749 -0.75098725  0.14987721]]
----------
[2 0 3 1]
```

```
----------
[0.98935825 0.84147098 0.99060736 0.6569866 ]
----------
```

```
Out[40]: True
```

6.2 np.tile 追尾复制

```
In [41]: a = np.arange(0, 40, 10)
         b = np.tile(a, (3, 5))
         print (b)
```

```
[[ 0 10 20 30  0 10 20 30  0 10 20 30  0 10 20 30  0 10 20 30]
 [ 0 10 20 30  0 10 20 30  0 10 20 30  0 10 20 30  0 10 20 30]
 [ 0 10 20 30  0 10 20 30  0 10 20 30  0 10 20 30  0 10 20 30]]
```

定义为 y=array([3,0,2,1,4,5]); argsort() 函数是将 x 中的元素从小到大排列，提取其对应的 index(索引)，然后输出到 y。

```
In [42]: a = np.array([[4, 3, 5], [1, 2, 1]])
         print (a)
         print ('----------')
         b = np.sort(a, axis=1)
         print (b)
         print ('----------')
         b
         a.sort(axis=1)
         print (a)
         print ('----------')
         a = np.array([4, 3, 1, 2])
         j = np.argsort(a)
         print (j)
         print ('----------')
         print (a[j])
```

```
[[4 3 5]
 [1 2 1]]
----------
[[3 4 5]
 [1 1 2]]
```

```
----------
[[3 4 5]
 [1 1 2]]
----------
[2 3 1 0]
----------
[1 2 3 4]
```

```
In [ ]:
```