# day8

May 24, 2024

## 1 NUMPY

NumPy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of non-negative integers. In NumPy dimensions are called axes.

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

[2]:
```
a = np.arange(15).reshape(3, 5)
a
```

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

[3]:
```
# ndarray.ndim
# the number of axes (dimensions) of the array.
a.ndim
```

[3]:
```
2
```

[4]:
```
#  This is a tuple of integers indicating the size of the array in each␣
 ↪dimension.
a.shape
```

[4]:
```
(3, 5)
```

[5]:
```
# the total number of elements of the array.
a.size
```

[5]:
```
15
```

[6]:
```
# an object describing the type of the elements in the array.
a.dtype
```

[6]:
```
dtype('int32')
```

[7]:
```
# the size in bytes of each element of the array.
a.itemsize
```

[7]:
```
4
```

[8]:
```
# the buffer containing the actual elements of the array
a.data
```

[8]:
```
<memory at 0x00000150B0EA4E10>
```

[9]:
```
type(a)
```

[9]:
```
numpy.ndarray
```

[10]:
```
x=np.array([[1,3],[3,4]],dtype=float)
x
```

[10]:
```
array([[1., 3.],
       [3., 4.]])
```

[11]:
```
x=np.array([[1,3],[3,4]],dtype=complex)
x
```

[11]:
```
array([[1.+0.j, 3.+0.j],
       [3.+0.j, 4.+0.j]])
```

The function zeros creates an array full of zeros, the function ones creates an array full of ones, and the function empty creates an array whose initial content is random and depends on the state of the memory. By default, the dtype of the created array is float64, but it can be specified via the key word argument dtype.

[12]:
```
np.zeros((3, 4))
```

[12]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[13]:
```
np.ones((2,5))
```

[13]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

[14]:
```
np.empty((2,4))
```

[14]:
```
array([[4.67296746e-307, 1.69121096e-306, 1.29061142e-306,
        1.89146896e-307],
       [7.56571288e-307, 3.11525958e-307, 1.24610723e-306,
        1.29061142e-306]])
```

To create sequences of numbers, NumPy provides the arange function which is analogous to the Python built-in range, but returns an array.

[15]:
```
np.arange(10,50,2)
```

[15]:
```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
       44, 46, 48])
```

## 2 Basic Operations

[16]:
```
a=np.array([2,3,4,5,7])
b=np.ones((1,5))
print(a-b)
```

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

[17]:
```
a*5
```

[17]:
```
array([10, 15, 20, 25, 35])
```

[18]:
```
a**2
```

[18]:
```
array([ 4,  9, 16, 25, 49])
```

[19]:
```
a>3
```

[19]:
```
array([False, False,  True,  True,  True])
```

[20]:
```
A = np.array([[1, 1],
              [0, 1]])
B = np.array([[2, 0],
              [3, 4]])
```

[21]:
```
# elementwise product
A*B
```

[21]:
```
array([[2, 0],
       [0, 4]])
```

[22]:
```
# matric product
A@B
```

[22]:
```
array([[5, 4],
       [3, 4]])
```

[23]:
```
A.dot(B)
```

[23]:
```
array([[5, 4],
       [3, 4]])
```

[24]:
```
A.sum()
```

[24]:
```
3
```

[25]:
```
A.min()
```

[25]:
```
0
```

[26]:
```
A.max()
```

[26]:
```
1
```

[27]:
```
A.mean()
```

[27]:
```
0.75
```

[28]:
```
A.transpose()
```

[28]:
```
array([[1, 0],
       [1, 1]])
```

[64]:
```
def f(x,y):
    return x+y;
x=np.fromfunction(f,(3,3),dtype="int64")
x
```

[64]:
```
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]], dtype=int64)
```

[29]:
```
A.sort()
A
```

[29]:
```
array([[1, 1],
       [0, 1]])
```

[54]:
```
A.cumsum() # cumulative sum of elements
```

[54]:
```
array([1, 2, 2, 3])
```

[31]:
```
A.sum(axis=1)
```

[31]:
```
array([2, 1])
```

[55]:
```
a.cumprod() # cumulative product of elements
```

[55]:
```
array([  2,   6,  24, 120, 840])
```

[56]:
```
#np.compress(condition, a, axis=None, out=None)
np.compress(a>3,a)  #  a slice along that axis is returned in output for each␣
 ↪index where condition evaluates to True
```

[56]:
```
array([4, 5, 7])
```

[57]:
```
# np.extract(condition, arr)
np.extract(a>2,a)  # compress is equivalent to extract.
```

[57]:
```
array([3, 4, 5, 7])
```

## 3 Universal Functions

[32]:
```
s=np.arange(4)
s
```

[32]:
```
array([0, 1, 2, 3])
```

[33]:
```
np.exp(A)
```

[33]:
```
array([[2.71828183, 2.71828183],
       [1.        , 2.71828183]])
```

[35]:
```
np.log(B)
```

```
C:\Users\heman\AppData\Local\Temp\ipykernel_27544\4061543246.py:1:
RuntimeWarning: divide by zero encountered in log
  np.log(B)
```

[35]:
```
array([[0.69314718,       -inf],
       [1.09861229, 1.38629436]])
```

[37]:
```
np.sqrt(49)
```

[37]:
```
7.0
```

## 4 Statistics

[38]:
```
A.mean()
```

[38]:
```
0.75
```

[39]:
```
A.std()  #standard deviation
```

[39]:
```
0.4330127018922193
```

[40]:
```
A.var() #variance
```

[40]:
```
0.1875
```

[44]:
```
np.cov(A)  #covariance
```

[44]:
```
array([[0. , 0. ],
       [0. , 0.5]])
```

## 5 Questions

[47]:
```
A.all()
```

[47]:
```
False
```

[48]:
```
A.any()
```

[48]:
```
True
```

[49]:
```
A.nonzero()
```

[49]:
```
(array([0, 0, 1], dtype=int64), array([0, 1, 1], dtype=int64))
```

[51]:
```
np.where(a>3)
```

[51]:
```
(array([2, 3, 4], dtype=int64),)
```

## 6 Indexing, Slicing

[58]:
```
a[:3]
```

[58]:
```
array([2, 3, 4])
```

[59]:
```
a[::-1]
```

[59]:
```
array([7, 5, 4, 3, 2])
```

[60]:
```
w=np.array([[1,2,3],[5,6,7],[9,7,6]])
w[:,2] #all row and third col
```

[60]:
```
array([3, 7, 6])
```

## 7 Shape

[67]:
```
w.shape
```

[67]:
```
(3, 3)
```

[69]:
```
w.ravel()   # returns the array, flattened
```

[69]:
```
array([1, 2, 3, 5, 6, 7, 9, 7, 6])
```

[72]:
```
w.reshape(9,1)
```

[72]:
```
array([[1],
       [2],
       [3],
       [5],
       [6],
       [7],
       [9],
       [7],
       [6]])
```

The reshape function returns its argument with a modified shape, whereas the ndarray.resize method modifies the array itself

[75]:
```
w.resize(1,9)
w
```

[75]:
```
array([[1, 2, 3, 5, 6, 7, 9, 7, 6]])
```