Python Course

NumPy

Feb 2023

Matin KarimPour

Welcome to NumPy!

NumPy (Numerical Python) is an open source Python library that's used in almost every field of science and engineering. It's the universal standard for working with numerical data in Python, and it's at the core of the scientific Python and PyData ecosystems. NumPy users include everyone from beginning coders to experienced researchers doing state-of-the-art scientific and industrial research and development. The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.

Installing NumPy

```
pip install numpy
```

What’s the difference between a Python list and a NumPy array?

NumPy arrays are faster and more compact than Python lists. An array consumes less memory and is convenient to use. NumPy uses much less memory to store data and it provides a mechanism of specifying the data types. This allows the code to be optimized even further.

An array is a central data structure of the NumPy library. An array is a grid of values and it contains information about the raw data, how to locate an element, and how to interpret an element. It has a grid of elements that can be indexed in various ways. The elements are all of the same type, referred to as the array dtype.

Python array

python :)

```python
import numpy as np

a = np.array([1, 2, 3, 4, 5, 6])
print(a)
# array([1, 2, 3, 4, 5, 6])

a = np.array([[1, 2, 3, 4], [5, 6, 7, 8],
[9, 10, 11, 12]])

#array([[ 1,  2,  3,  4],
#       [ 5,  6,  7,  8],
#       [ 9, 10, 11, 12]]

print(a[0])
# [1 2 3 4]
```

Basic array

python :)

```python
np.zeros(2)
# array([0., 0.])

np.ones(2)
# array([1., 1.])

np.empty(2)
# array([2.000e+000, 4.67226695e-310])

np.arange(4)
# array([0, 1, 2, 3])

np.arange(2, 9, 2)
# array([2, 4, 6, 8])

array([2, 4, 6, 8])
# array([ 0. ,  2.5,  5. ,  7.5, 10. ])

x = np.ones(2, dtype=np.int64)
x
# array([1, 1])
```

Adding, removing, and sorting elements

In order to remove elements from an array, it's simple to use indexing to select the elements that you want to keep.

python :)

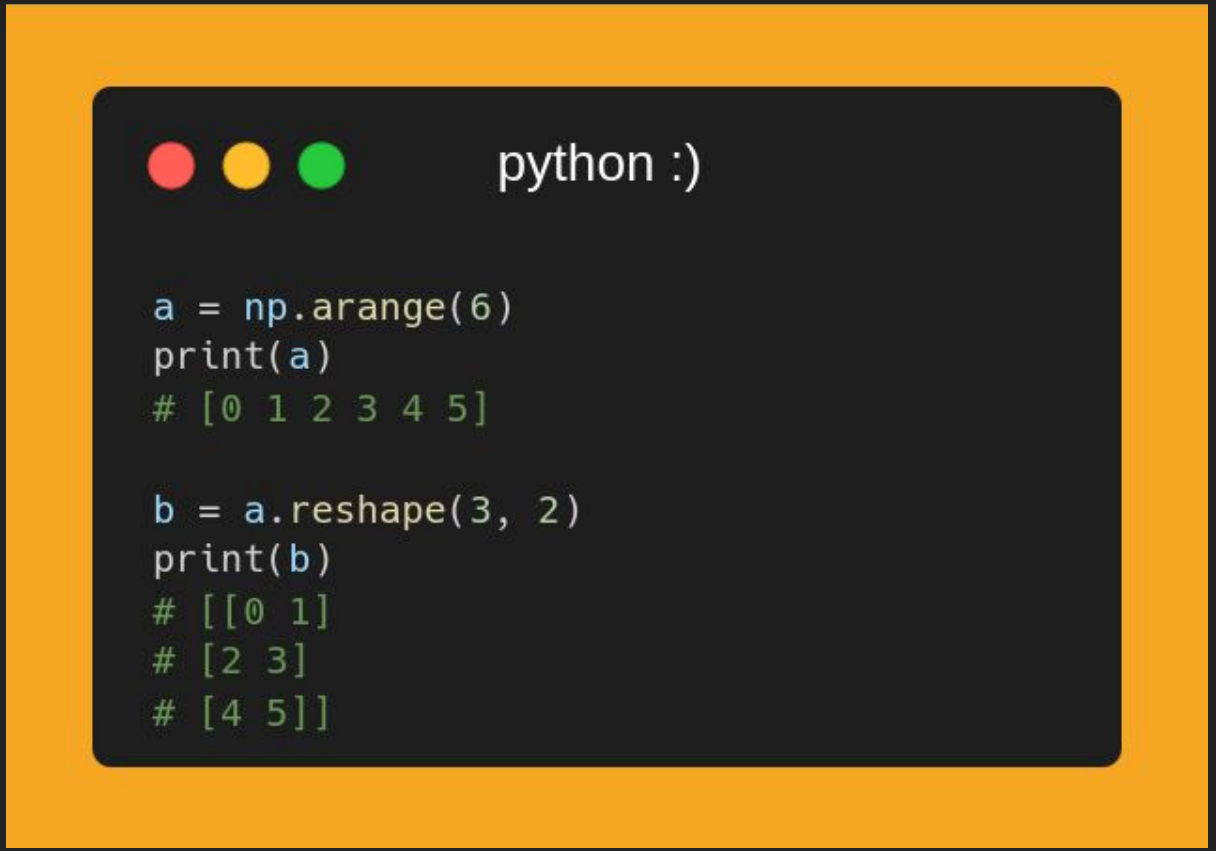
```
arr = np.array([2, 1, 5, 3, 7, 4, 6, 8])
np.sort(arr)
# array([1, 2, 3, 4, 5, 6, 7, 8])

a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])
np.concatenate((a, b))
# array([1, 2, 3, 4, 5, 6, 7, 8])

x = np.array([[1, 2], [3, 4]])
np.concatenate((x, y), axis=0)
#array([[1, 2],
#       [3, 4],
#       [5, 6]])
```

Shape and size of an array

python :)

```
array_example = np.array([[[0, 1, 2, 3],
                           [4, 5, 6, 7]],

                          [[0, 1, 2, 3],
                           [4, 5, 6, 7]],

                          [[0 ,1 ,2, 3],
                           [4, 5, 6,7]]])

array_example.ndim
# 3

array_example.size
# 24

array_example.shape
# (3, 2, 4)
```

Reshape an array



Convert a 1D array into a 2D array

python :)

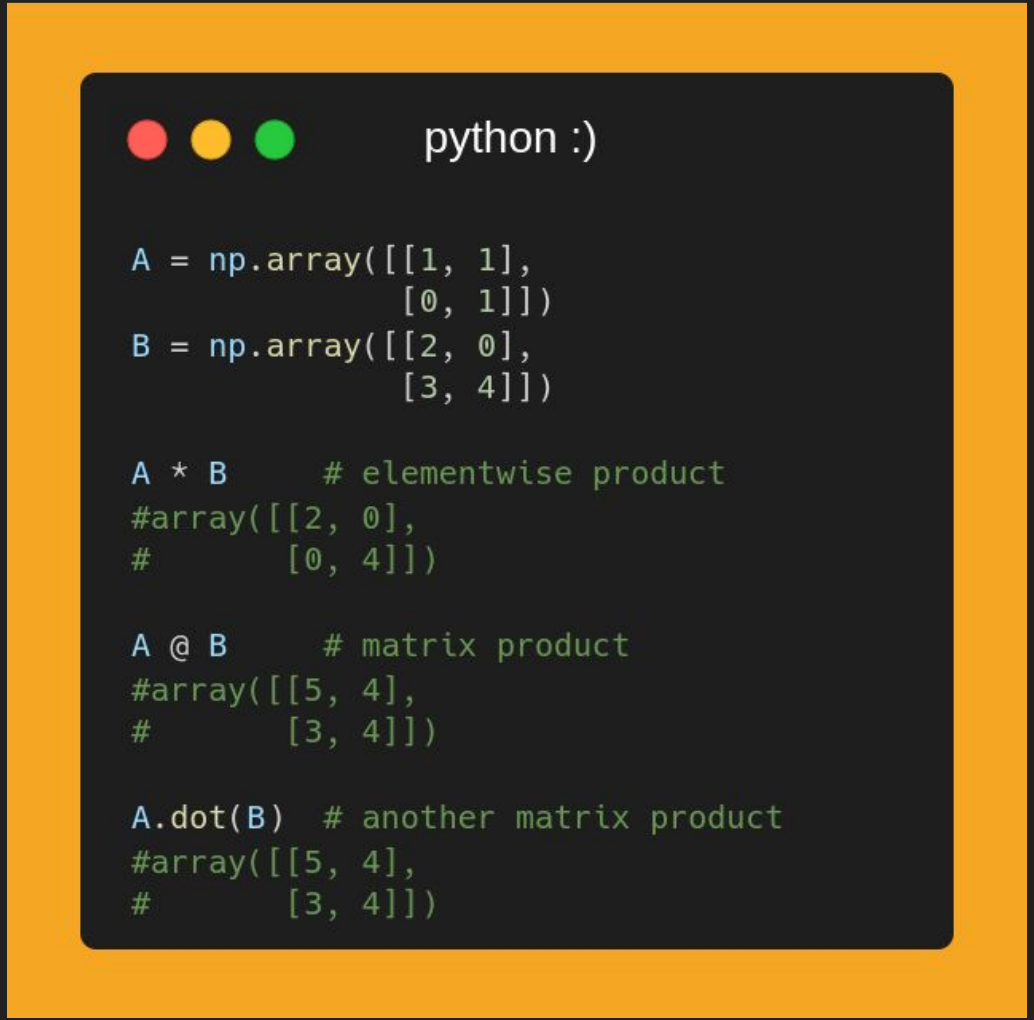
```python
a = np.array([1, 2, 3, 4, 5, 6])
a.shape
# (6,)

a2 = a[np.newaxis, :]
a2.shape
# (1, 6)
a2
# array([[1, 2, 3, 4, 5, 6]])

col_vector = a[:, np.newaxis]
col_vector.shape
# (6, 1)

col_vector
"""
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6]])
"""

b = np.expand_dims(a, axis=1)
b.shape
# (6, 1)
```

Basic Operations

python :)

```
a = np.array([20, 30, 40, 50])
b = np.arange(4)
b
#array([0, 1, 2, 3])

c = a - b
c
#array([20, 29, 38, 47])

b**2
#array([0, 1, 4, 9])

10 * np.sin(a)
#array([ 9.12945251, -9.88031624,
7.4511316 , -2.62374854])

a < 35
#array([ True,  True, False, False])
```

Basic Operations



Basic Operations

Universal Functions

python :)

```
B = np.arange(3)
B
#array([0, 1, 2])

np.exp(B)
#array([1.        , 2.71828183, 7.3890561
])

np.sqrt(B)
#array([0.        , 1.        ,
1.41421356])

C = np.array([2., -1., 4.])
np.add(B, C)
#array([2., 0., 6.])
```

Indexing, Slicing and Iterating

python :)

```
a = np.arange(10)**3
a
# array([  0,   1,   8,  27,  64, 125,
216, 343, 512, 729])

a[2]
# 8

a[2:5]
#array([ 8, 27, 64])

a[:6:2] = 1000
a
# array([1000,    1, 1000,   27, 1000,
125,  216,  343,  512,  729])

a[::-1]  # reversed a
# array([ 729,  512,  343,  216,  125,
1000,   27, 1000,    1, 1000])
```

Indexing, Slicing and Iterating

python :)

```
def f(x, y):
    return 10 * x + y

b = np.fromfunction(f, (5, 4), dtype=int)
b
#array([[ 0,  1,  2,  3],
#       [10, 11, 12, 13],
#       [20, 21, 22, 23],
#       [30, 31, 32, 33],
#       [40, 41, 42, 43]])

b[2, 3]
# 23

b[0:5, 1]  # each row in the second
column of b
# array([ 1, 11, 21, 31, 41])

b[:, 1]    # equivalent to the previous
example
# array([ 1, 11, 21, 31, 41])

b[1:3, :]  # each column in the second
and third row of b
# array([[10, 11, 12, 13],
#      [20, 21, 22, 23]])
```

Indexing, Slicing and Iterating

python :)

```
c = np.array([[[  0,  1,  2],
               [ 10, 12, 13]],
              [[100, 101, 102],
               [110, 112, 113]]])
c.shape
#(2, 2, 3)

c[1, ...]  # same as c[1, :, :] or c[1]
#array([[100, 101, 102],
#       [110, 112, 113]])

c[..., 2]  # same as c[:, :, 2]
#array([[  2,  13],
#       [102, 113]])
```

Any Question?