Introduction to Numpy

Cole Dieckhaus

August 28, 2019

1 Fundamental Python Libraries

1.1 Numpy

Numpy Documentation
Numpy book

python3 -m pip install numpy

The main use of Numpy for Python developers is to get access to contiguous memory arrays, just like the ones commonly used in C++. Numpy is far more popular than other libraries with this same data structure, largely because of its broadcasting feature, which allows operations acting on multiple arrays of different sizes to work without any special functions.
The Numpy interpreter is also significantly faster than Python for many tasks. For example, if you had a python list with values 1 to 1 million, and you made a list comprehension to add 1 to each of the values, it takes .07 seconds. On the other hand, if you have a Numpy array with values 1 to 1 million, using broadcasting to add one to each value only takes .008 seconds.

Importing Numpy

```
>>> import numpy as np
```

Creating an ndarray

```
>>> x = np.array(['a', 'b', 'c'])
>>> x
array(['a', 'b', 'c'], dtype='<U1')

>>> x = np.arange(0, 3, 1)
>>> x
array([0, 1, 2])

>>> x = np.arange(0, 1, .1)
>>> x
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

```
>>> x = np.zeros(shape=(3, 5))
>>> x
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])

>>> x = np.random.uniform(0, 1, size=10)
>>> x
array([0.48908712, 0.21474925, 0.71589095, 0.74606879, 0.0992887 ,
       0.33601409, 0.41361811, 0.85178078, 0.21592613, 0.8155579 ])
```

Reshaping

```
>>> x = np.arange(1, 13).reshape((3, 4))
>>> x
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
>>> x.T
array([[ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11],
       [ 4,  8, 12]])

>>> x = np.random.power(7, size=4)
>>> x
array([0.73801263, 0.8526897 , 0.88994279, 0.93148266])
>>> x.T
array([0.73801263, 0.8526897 , 0.88994279, 0.93148266])
>>> x.reshape((-1, 1))
array([[0.73801263],
       [0.8526897 ],
       [0.88994279],
       [0.93148266]])
```

Combining ndarrays

```
>>> a = np.arange(3)
>>> x = np.append(a, a)
>>> x
array([0, 1, 2, 0, 1, 2])

>>> a = np.arange(6).reshape((2, -1))
>>> a
```

```
array([[0, 1, 2],
       [3, 4, 5]])
>>> x = np.append(a, a)
>>> x
array([0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5])
>>> x = np.concatenate((a, a), axis=0)
>>> x
array([[0, 1, 2],
       [3, 4, 5],
       [0, 1, 2],
       [3, 4, 5]])
>>> x = np.concatenate((a, a), axis=1)
>>> x
array([[0, 1, 2, 0, 1, 2],
       [3, 4, 5, 3, 4, 5]])

>>> x = np.vstack((a, a))
>>> x  ## Identical to concatenate: axis=0
array([[0, 1, 2],
       [3, 4, 5],
       [0, 1, 2],
       [3, 4, 5]])

>>> x = np.hstack((a, a))
>>> x  ## Identical to concatenate: axis=1
array([[0, 1, 2, 0, 1, 2],
       [3, 4, 5, 3, 4, 5]])
```

Indexing & Slicing ndarrays

```
>>> x = np.arange(12).reshape((3, 4))
>>> x
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> x[1, 2]
6

>>> ## Every new dimension index is at the front
>>> x = np.arange(12).reshape((2, 2, 3))
>>> x
array([[[ 0,  1,  2],
        [ 3,  4,  5]],
       [[ 6,  7,  8],
        [ 9, 10, 11]]]])
>>> x[0, 1, 2]
5

>>> x = np.arange(0, 18, 3)
>>> x
array([ 0,  3,  6,  9, 12, 15])
>>> x[[1, 3, 5]]
array([ 3,  9, 15])

>>> x = np.arange(10).reshape((2, -1))
>>> x
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> locs = np.where(x % 2)
>>> x[locs]
array([1, 3, 5, 7, 9])

>>> x = np.arange(5) + np.arange(5).reshape((-1, 1))
>>> x
array([[0, 1, 2, 3, 4],
       [1, 2, 3, 4, 5],
       [2, 3, 4, 5, 6],
       [3, 4, 5, 6, 7],
       [4, 5, 6, 7, 8]])
>>> x[1:4]
array([[1, 2, 3, 4, 5],
       [2, 3, 4, 5, 6],
       [3, 4, 5, 6, 7]])
>>> x[:, 1:4]
array([[1, 2, 3],
       [2, 3, 4],
       [3, 4, 5],
       [4, 5, 6],
       [5, 6, 7]])
>>> x[1:4, 1:4]
array([[2, 3, 4],
       [3, 4, 5],
       [4, 5, 6]])

>>> x = np.arange(5)
>>> x
array([0, 1, 2, 3, 4])
```

```
>>> x[::1]
array([0, 1, 2, 3, 4])
>>> x[::2]
array([0, 2, 4])
>>> x[::-1]
array([4, 3, 2, 1, 0])
```

Broadcasting

```
>>> x = np.ones(shape=3) * 6
>>> x
array([6., 6., 6.])

>>> a = np.arange(3)
>>> b = np.arange(3).reshape((-1, 1))
>>> a
array([0, 1, 2])
>>> b
array([[0],
       [1],
       [2]])
>>> x = a + b
>>> x
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]])

>>> x = np.array([-3, 0, 3]) + np.zeros(shape=(3, 1))
>>> x
array([[-3.,  0.,  3.],
       [-3.,  0.,  3.],
       [-3.,  0.,  3.]])
>>> y = np.array([-10, 0, 10])
>>> y
array([-10,   0,  10])
>>> x * y
array([[30.,  0., 30.],
       [30.,  0., 30.],
       [30.,  0., 30.]])
>>> y = y.reshape((-1, 1))
>>> y
array([[-10],
       [  0],
       [ 10]])
```

```
>>> x * y
array([[ 30.,  -0., -30.],
       [ -0.,   0.,   0.],
       [-30.,   0.,  30.]])

>>> x = np.arange(5) + np.zeros(shape=(3, 1))
>>> x
array([[0., 1., 2., 3., 4.],
       [0., 1., 2., 3., 4.],
       [0., 1., 2., 3., 4.]])
>>> y = np.array([[1], [2], [3]])
>>> y
array([[1],
       [2],
       [3]])
>>> x = x >= y
>>> x
array([[False,  True,  True,  True,  True],
       [False, False,  True,  True,  True],
       [False, False, False,  True,  True]])
>>> x = np.int_(x)
>>> x
array([[0, 1, 1, 1, 1],
       [0, 0, 1, 1, 1],
       [0, 0, 0, 1, 1]])
```

Memory Nuances

```
>>> a = np.arange(10).reshape((2, 5))
>>> a
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> b = a[::-1, ::-1]
>>> b
array([[9, 8, 7, 6, 5],
       [4, 3, 2, 1, 0]])
>>> a[0] += 1
>>> a
array([[1, 2, 3, 4, 5],
       [5, 6, 7, 8, 9]])
>>> b
array([[9, 8, 7, 6, 5],
       [5, 4, 3, 2, 1]])
```

```
>>> a = np.arange(10).reshape((2, 5))
>>> a
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> b = np.copy(a)[::-1, ::-1]
>>> b
array([[9, 8, 7, 6, 5],
       [4, 3, 2, 1, 0]])
>>> a[0] += 1
>>> a
array([[1, 2, 3, 4, 5],
       [5, 6, 7, 8, 9]])
>>> b
array([[9, 8, 7, 6, 5],
       [4, 3, 2, 1, 0]])
```

Numpy Statistics

```
>>> x = np.random.normal(loc=12, size=10000)
>>> np.mean(x)
12.004210965789426

>>> x = np.random.randint(0, 100, size=100000)
>>> np.median(x)
49.0
```

Masked Arrays

```
>>> x = np.ma.array(np.arange(10), mask=[False] * 10)
>>> x
masked_array(data=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value=999999)

>>> x.mask[np.where(x % 2)] = True
>>> x
masked_array(data=[0, --, 2, --, 4, --, 6, --, 8, --],
             mask=[False,  True, False,  True, False,  True, False,  True,
                   False,  True],
       fill_value=999999)

>>> x >= 0
```

```
masked_array(data=[True, --, True, --, True, --, True, --, True, --],
             mask=[False,  True, False,  True, False,  True, False,  True,
                   False,  True],
       fill_value=999999)

>>> np.sum(x)
20
```

General Functions

```
>>> a = list(range(5))
>>> a
[0, 1, 2, 3, 4]
>>> b = a[::-1]
>>> b
[4, 3, 2, 1, 0]
>>> for i, value in enumerate(b):
    a[i] += value

>>> a
[4, 4, 4, 4, 4]

>>> a = np.arange(10).reshape((2, 5))
>>> a
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> b = np.copy(a)[::-1, ::-1]
>>> b
array([[9, 8, 7, 6, 5],
       [4, 3, 2, 1, 0]])
>>> for idx, value in np.ndenumerate(b):
    a[idx] += value
>>> a
array([[9, 9, 9, 9, 9],
       [9, 9, 9, 9, 9]])

>>> x = np.arange(12).reshape((2, 2, 3))
>>> x
array([[[ 0,  1,  2],
        [ 3,  4,  5]],

       [[ 6,  7,  8],
        [ 9, 10, 11]]])
>>> np.ravel(x)
```

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

>>> x = np.arange(7)
>>> x
array([0, 1, 2, 3, 4, 5, 6])
>>> y = np.minimum(x, x[::-1])
>>> y
array([0, 1, 2, 3, 2, 1, 0])
>>> np.argmax(y)
3
```

1.2 Craps

```
Roll 2 dice and sum values.

Part 1: First roll
Win if 7 or 11
Loose if 2, 3 or 12
Go onto Part 2 if did not win or loose

Part 2: Roll until win/loose
Win if re-roll number from part 1 again
Loose if 7

Brute force method to find probability of winning craps. 1M iterations of the game.
This can be done with all games taking place in one array.
```