ipythonexercise_part4.1

February 12, 2015

1 Answers for IPython Exercise Part 4.1

```
In [17]: import numpy as np
         a = np.array([1,2,3,4])
         a + 1
         print a
         2**a
         print a
         b = np.ones(4) + 1
         print a - b
         print a * b
         j = np.arange(5)
         print 2**(j+1) - j
         a = np.arange(10000)
         %timeit a+1
         l = range(10000)
         %timeit [i+1 for i in l]
```

```
[1 2 3 4]
[1 2 3 4]
[-1.  0.  1.  2.]
[ 2.  4.  6.  8.]
[ 2  3  6 13 28]
10000 loops, best of 3: 30.9 µs per loop
1000 loops, best of 3: 584 µs per loop
```

```
In [18]: c = np.ones((3,3))
         c * c
```

```
Out[18]: array([[ 1.,  1.,  1.],
                [ 1.,  1.,  1.],
                [ 1.,  1.,  1.]])
```

```
In [19]: [2**0, 2**1, 2**2, 2**3, 2**4]
```

```
Out[19]: [1, 2, 4, 8, 16]
```

```
In [20]: j = np.arange(5)
         a_j = 2^(3*j) - j
         a_j
```

```
Out[20]: array([ 2,  0,  6,  4, 10])
```

1.0.1 Other Operations

It is useful when comparing arrays to find out if they are equal within a given tolerance value.

Transpose is a view. Using a += a.T will make the matrix symmetric regardless. Though, it might not work for other operations.

1.0.2 Reductions

Product

"sum" provides the summ of all array elements while cumsum provides a sum of all array elements cumulatively.

```
In [21]: a = np.array([[1,2,3], [4,5,6]])
         print a
         print "..."
         print a.ravel()
         print "..."
         print a.T
         print "..."
         print a.T.ravel()
         print "..."
         print a
```

```
[[1 2 3]
 [4 5 6]]
...
[1 2 3 4 5 6]
...
[[1 4]
 [2 5]
 [3 6]]
...
[1 4 2 5 3 6]
...
[[1 2 3]
 [4 5 6]]
```

```
In [22]: a = np.array([[1,2,3], [4,5,6]])
         print a
         print a.flatten()
         print a
```

```
[[1 2 3]
 [4 5 6]]
[1 2 3 4 5 6]
[[1 2 3]
 [4 5 6]]
```

1.0.3 Shape Manipulations

Ravel returns a view of the input. It will make a copy only if needed. Flatten returns a copy of the elements.

```
In [23]: a = np.array([[1,2,3], [4,5,6]])
         print a
         print "..."
         print a.T
```

```
[[1 2 3]
 [4 5 6]]
...
[[1 4]
 [2 5]
 [3 6]]
```

1.0.4 Sorting

In-Place and Out-of-Place sorting

```
In [24]: a = np.array([[71,41,19], [13,85,22]])
         a.sort(axis=1)
         a
```

```
Out[24]: array([[19, 41, 71],
                [13, 22, 85]])
```

```
In [25]: a = np.array([[71,41,19], [13,85,22]])
         b = a.sort()
         d = np.array([[71,41,19], [13,85,22]])
         b = a.sort()
         c = np.sort(d)
         a, c
```

```
Out[25]: (array([[19, 41, 71],
                [13, 22, 85]]),
          array([[19, 41, 71],
                [13, 22, 85]]))
```

Creating arrays with diff. dtypes and sorting them.

```
In [26]: a = np.array([[43,6.324,'a',3.05], [9.23563566, 'lololol', 12347,54683]])
         a.sort()
         a
```

```
Out[26]: array([['3.05', '43', '6.324', 'a'],
                ['12347', '54683', '9.23563566', 'lololol']],
               dtype='|S10')
```

Look at np.random.shuffle for a way to create sortable input quicker.

```
In [27]: a = np.array([[61,51,321,211], [143,122,11,21], [421,571,131,961]])
         np.random.shuffle(a)
         a, np.sort(a)
```

```
Out[27]: (array([[421, 571, 131, 961],
                [143, 122,  11,  21],
                [ 61,  51, 321, 211]]),
          array([[131, 421, 571, 961],
                [ 11,  21, 122, 143],
                [ 51,  61, 211, 321]]))
```

Combine ravel, sort and reshape.

```
In [28]: a = np.array([[61,51,321,211], [143,122,11,21], [421,571,131,961]])
         a, a.ravel(), np.sort(a), a.reshape(4*3), np.sort(a.ravel())
```

```
Out[28]: (array([[ 61,  51, 321, 211],
                [143, 122,  11,  21],
                [421, 571, 131, 961]]),
          array([ 61,  51, 321, 211, 143, 122,  11,  21, 421, 571, 131, 961]),
          array([[ 51,  61, 211, 321],
                [ 11,  21, 122, 143],
                [131, 421, 571, 961]]),
          array([ 61,  51, 321, 211, 143, 122,  11,  21, 421, 571, 131, 961]),
          array([ 11,  21,  51,  61, 122, 131, 143, 211, 321, 421, 571, 961]))
```

Look at the ‘axis’ keyword for ‘sort’ and rewrite the previous exercise.

```
In [29]: a = np.array([[61,51,321,211], [143,122,11,21], [421,571,131,961]])
         print a
         print np.sort(a, axis=1)
         print np.sort(a, axis=0)
```

```
[[ 61  51 321 211]
 [143 122  11  21]
 [421 571 131 961]]
[[ 51  61 211 321]
 [ 11  21 122 143]
 [131 421 571 961]]
[[ 61  51  11  21]
 [143 122 131 211]
 [421 571 321 961]]
```