Kato_ipythonexercise_part4.1

February 12, 2015

```
In [2]: 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: 16.9 s per loop
The slowest run took 9.12 times longer than the fastest. This could mean that an intermediate result is
1000 loops, best of 3: 650 s per loop
```

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

```
Out[3]: array([[ 1.,  1.,  1.],
               [ 1.,  1.,  1.],
               [ 1.,  1.,  1.]])
```

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

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

```
In [9]: j = np.arange(5)
        a_j = 2^(3*j) - j
        a_j
```

```
Out[9]: array([ 2,  0,  6,  4, 10])
```

Exercise: Other Operations

- Look at the help for `np.allclose`. When might this be useful? When you are comparing arrays to find out if they are equal within a given tolerance value.

- Look at the help for `np.triu` and `np.tril`.
- Is the transpose a view or a copy? What implications does this have for making a matrix symmetric? The transpose is a view, however, using a += a.T seems to make the matrix symmetric regardless. It might not work for other operations, though.

Exercise: Reductions

- Given there is a `sum`, what other function might you expect to see? product
- What is the difference between `sum` and `cumsum`? sum provides the sum of all array elements. cumsum provides a sum of all array elements cumulatively.

```
In [15]: 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 [14]: 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]]
```

Exercise: Shape manipulations

- Look at the docstring for reshape, especially the notes section which has some more information about copies and views.
- Use flatten as an alternative to ravel. What is the difference? (Hint: check which one returns a view and which a copy) Ravel returns a view of the input. A copy is only made if needed. On the other hand, flatten returns a copy of the elements.

- Experiment with transpose for dimension shuffling.

```
In [16]: 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]]
```

Exercise: Sorting

- Try both in-place and out-of-place sorting.

```
In [22]: a = np.array([[7,4,9],[1,8,2]])
         a.sort(axis=1)
         a
```

```
Out[22]: array([[4, 7, 9],
                [1, 2, 8]])
```

```
In [37]: a = np.array([[7,4,9],[1,8,2]])
         b = a.sort()
         d = np.array([[7,4,9],[1,8,2]])
         b = a.sort()
         c = np.sort(d)
         a, c
```

```
Out[37]: (array([[4, 7, 9],
                 [1, 2, 8]]), array([[4, 7, 9],
                 [1, 2, 8]]))
```

- Try creating arrays with different dtypes and sorting them.

```
In [41]: a = np.array([[4,6.32,'a',3.0],[9.2356356, 'ehehehe',123,5463]])
         a.sort()
         a
```

```
Out[41]: array([['3.0', '4', '6.32', 'a'],
                ['123', '5463', '9.23563', 'ehehehe']],
               dtype='|S7')
```

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

```
In [76]: a = np.array([[6,5,32,21],[43,22,1,2],[42,57,13,96]])
         np.random.shuffle(a)
         a, np.sort(a)
```

```
Out[76]: (array([[43, 22,  1,  2],
                 [ 6,  5, 32, 21],
                 [42, 57, 13, 96]]), array([[ 1,  2, 22, 43],
                 [ 5,  6, 21, 32],
                 [13, 42, 57, 96]]))
```

- Combine `ravel`, `sort` and `reshape`.

```
In [47]: a = np.array([[6,5,32,21],[43,22,1,2],[42,57,13,96]])
         a, a.ravel(), np.sort(a), a.reshape(4*3), np.sort(a.ravel())
```

```
Out[47]: (array([[ 6,  5, 32, 21],
                 [43, 22,  1,  2],
                 [42, 57, 13, 96]]),
          array([ 6,  5, 32, 21, 43, 22,  1,  2, 42, 57, 13, 96]),
          array([[ 5,  6, 21, 32],
                 [ 1,  2, 22, 43],
                 [13, 42, 57, 96]]),
          array([ 6,  5, 32, 21, 43, 22,  1,  2, 42, 57, 13, 96]),
          array([ 1,  2,  5,  6, 13, 21, 22, 32, 42, 43, 57, 96]))
```

- Look at the 'axis' keyword for 'sort' and rewrite the previous exercise.

```
In [56]: a = np.array([[6,5,32,21],[43,22,1,2],[42,57,13,96]])
         print a
         print np.sort(a,axis=1)
         print np.sort(a,axis=0)
```

```
[[ 6  5 32 21]
 [43 22  1  2]
 [42 57 13 96]]
[[ 5  6 21 32]
 [ 1  2 22 43]
 [13 42 57 96]]
[[ 6  5  1  2]
 [42 22 13 21]
 [43 57 32 96]]
```

```
In [ ]:
```