4 列表与元组的速度比较

IPython 中用 `magic` 命令 `%timeit` 来计时。

4.1 比较生成速度

```
%timeit [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25]
```

```
180 ns ± 2.66 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

```
%timeit (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25)
```

```
16 ns ± 0.227 ns per loop (mean ± std. dev. of 7 runs, 100000000 loops each)
```

可以看到，元组的生成速度要比列表的生成速度快得多，相差大概一个数量级。

```
%timeit list(range(10000))
```

```
187 µs ± 18.3 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
%timeit tuple(range(10000))
```

```
174 µs ± 7.53 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

4.2 比较遍历速度

产生内容相同的随机列表和元组：

```
from numpy.random import rand
values = rand(10000,4)
```

```
lst = [list(row) for row in values]
tup = tuple(tuple(row) for row in values)
```

```
 %timeit for row in lst: list(row)
```

```
2.13 ms ± 214 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
%timeit for row in tup: tuple(row)
```

```
1.33 ms ± 44.2 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

在遍历上，元组和列表的速度表现差不多。

4.3 比较遍历和索引速度：

```
%timeit for row in lst: a = row[0] + 1
```

```
The slowest run took 12.20 times longer than the fastest. This could mean that an i
ntermediate result is being cached
100 loops, best of 3: 3.73 ms per loop
```

```
%timeit for row in tup: a = row[0] + 1
```

```
100 loops, best of 3: 3.82 ms per loop
```

元组的生成速度会比列表快很多，迭代速度快一点，索引速度差不多。