条件运算符、递归和推导

条件运算符

```
<表达式1> ? <表达式2> : <表达式3>
```

```
max = (a > b) ? a : b
```

```
value_when_true if condition else value_when_false
```

```
In : 'True' if True else 'False'
Out: 'True'

In : 'True' if False else 'False'
Out: 'False'
```

一个不太好写法 👇

```
[value_when_false, value_when_true](bool(condition))
```

```
In : ['False', 'True'][True]
Out: 'True'

In : ['False', 'True'][False]
Out: 'False'
```

and ... or

```
condition and value_when_true or value_when_false
```

```
In : 'a' and 'b'
Out: 'b'

In : False and 'b'
Out: False

In : 'a' and False and 'b'
Out: False
```

```
In : 'a' or 'b'
Out: 'a'

In : False or 'b'
Out: 'b'

In : 'a' or False or 'b'
Out: 'a'
```

```
In : True and 1 or 0
Out: 1

In : False and 1 or 0
Out: 0
```

下列这种情况条件运算错误👇

```
In : True and '' or 0
Out: 0

In : True and None or 0
Out: 0
```

递归

1. 必定存在可使递归调用终止的条件, 否则导致出现无限递归
2. 在每一次调用自己时，在某种意义上应该更接近于解

阶乘例子

阶乘的数学定义是： n! = n * (n-1)!

例如：3! = 3 x 2 x 1 = 6

```
In : def factorial(n):
...:     if n == 1:
...:         return 1
...:     else:
...:         return n * factorial(n-1)
...:

In : def factorial(n):
...:     return 1 if n == 1 else n * factorial(n-1)
...:
```

```
In : import sys

In : sys.getrecursionlimit()
Out: 1000

In : f = factorial(1000)
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-2-f4f69c4597b9> in <module>()
----> 1 f = factorial(1000)

<ipython-input-1-d7501bb4fcb0> in factorial(n)
      3                         return 1
      4             else:
----> 5                         return n * factorial(n-1)
      6

... last 1 frames repeated, from the frame below ...

<ipython-input-1-d7501bb4fcb0> in factorial(n)
      3                         return 1
      4             else:
----> 5                         return n * factorial(n-1)
      6

RecursionError: maximum recursion depth exceeded in comparison

In : sys.setrecursionlimit(2000)

In : f = factorial(1000)
```

推导(comprehensions)

推导包含列表推导、字典推导和集合推导等，推导也叫作解析，如列表解析

```
In : l = []

In : for i in [1, 2, 3, 4]:
...:     l.append(i * i)
...:

In : l
Out: [1, 4, 9, 16]
```

列表解析可以这样用 👇

```
In : [i * i for i in [1, 2, 3, 4]]
Out: [1, 4, 9, 16]
```

带条件的解析式

```
In : [i * i for i in [1, 2, 3, 4] if i % 2]
Out: [1, 9]

In : [i * i for i in range(10) if i % 2 and i % 3]
Out: [1, 25, 49]
```

集合解析/字典解析

```
In : {i * i for i in [1, 2, 3, 4, 1]}
Out: {1, 4, 9, 16}  # 重复的元素去掉了

In : {k: v * v for k, v in [('a', 1), ('b', 2)]}
Out: {'a': 1, 'b': 4}
```

延伸阅读

1. http://python-3-patterns-idioms-test.readthedocs.io/en/latest/Comprehensions.html