7 Python推导式

推导式comprehensions（又称解析式），是Python的一种独有特性。推导式是可以从一个数据序列构建另一个新的数据序列的结构体。 共有三种推导，在Python2和3中都有支持：

- 列表(list)推导式
- 字典(dict)推导式
- 集合(set)推导式

7.1 列表推导式

使用[]和循环可以用来生成列表：

基本格式

```
variable = [out_exp_res for out_exp in input_list if out_exp == 2]
  out_exp_res:    列表生成元素表达式，可以是有返回值的函数。
  for out_exp in input_list:     迭代input_list将out_exp传入out_exp_res表达式中。
  if out_exp == 2:     根据条件过滤哪些值可以。
```

```
#help(range)
#range?
#range??
```

```
[x * x for x in range(1, 100)]
```

```
[(x,y) for x in range(1, 4) for y in range(1, 4)]
```

```
[(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)]
```

用推导列表式生成元组，

理解一下：先用range（）生成一个列表，然后迭代，这个操作执行了两次，一次是x，一次是

y，然后再用元组的定义生成一个列表。

再来看一个对比的例子

```
values = [10, 21, 4, 7, 12]
squares = []
for x in values:
    squares.append(x**2)
print(squares)
```

```
[100, 441, 16, 49, 144]
```

列表推导式可以使用更简单的方法来创建这个列表：

```
values = [10, 21, 4, 7, 12]
squares = [x**2 for x in values]
print(squares)
```

```
[100, 441, 16, 49, 144]
```

还可以在列表推导式中加入条件进行筛选。

例如在上面的例子中，假如只想保留列表中不大于`10`的数的平方：

```
values = [10, 21, 4, 7, 12]
squares = [x**2 for x in values if x <= 10]
print squares
```

```
[100, 16, 49]
```

使用()生成generator

将俩表推导式的[]改成()即可得到生成器。 生成器的概念，后面深入讲解。

```
multiples = (i for i in range(30) if i % 3 is 0)
print(type(multiples))
```

```
<class 'generator'>
```

7.2 集合推导式&字典推导式

它们跟列表推导式也是类似的。 唯一的区别在于它使用大括号{}。

```
square_set = {x**2 for x in values if x <= 10}
print(square_set)

square_dict = {x: x**2 for x in values if x <= 10}
print(square_dict)
```

```
{16, 49, 100}
{10: 100, 4: 16, 7: 49}
```

大小写key合并

```
mcase = {'a': 10, 'b': 34, 'A': 7, 'Z': 3}
mcase_frequency = {
    k.lower(): mcase.get(k.lower(), 0) + mcase.get(k.upper(), 0)
    for k in mcase.keys()
    if k.lower() in ['a','b']
}
print(mcase_frequency)
```

```
{'a': 17, 'b': 34}
```

快速更换key和value

```
mcase = {'a': 10, 'b': 34}
mcase_frequency = {v: k for k, v in mcase.items()}
print(mcase_frequency)
```

```
{10: 'a', 34: 'b'}
```

再如，计算上面例子中生成的列表中所有元素的和：

```
total = sum([x**2 for x in values if x <= 10])
print(total)
```

```
165
```

但是，**Python**会生成这个列表，然后在将它放到垃圾回收机制中（因为没有变量指向它），这毫无疑问是种浪费。

为了解决这种问题，**Python**使用产生式表达式来解决这个问题：

```
total = sum(x**2 for x in values if x <= 10)
print(total)
```

```
165
```

与上面相比，只是去掉了括号，但这里并不会一次性的生成这个列表。

比较一下两者的用时：

```
x = range(1000000)
```

```
%timeit total = sum([i**2 for i in x])
```

```
514 ms ± 58.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
%timeit total = sum(i**2 for i in x)
```

```
487 ms ± 34.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

7.3 enumerate

这是一个有意思的内置函数，本来我们可以通过for i in range(len(list))的方式得到一个list的每个元素编号，然后在用list[i]的方式得到该元素。如果要同时得到元素编号和元素怎么办？就是这样

了:

```
week = ["monday", "sunday", "friday"]
for i in range(len(week)):
    print(week[i]+' is '+str(i))   #注意，i是int类型，如果和前面的用+连接，必须是str类型
```

```
monday is 0
sunday is 1
friday is 2
```

python中提供了一个内置函数enumerate，能够实现类似的功能,算是一个有意思的内置函数了，主要是提供一个简单快捷的方法

```
for (i,day) in enumerate(week):
    print(day +' is '+ str(i))
```

```
monday is 0
sunday is 1
friday is 2
```

顺便抄录几个例子，供看官欣赏，最好实验一下

```
seasons = ['Spring', 'Summer', 'Fall', 'Winter']
list(enumerate(seasons))
```

```
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
```

```
list(enumerate(seasons, start=1))
```

```
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

用列表生成器打印九九乘法表

```
print('\n'.join(['\t'.join(['%d * %d = %d'%(y,x,x*y) for y in range(1,x+1)])for x i
```

```
n range(1,10)]))
```

```
1 * 1 = 1
1 * 2 = 2   2 * 2 = 4
1 * 3 = 3   2 * 3 = 6   3 * 3 = 9
1 * 4 = 4   2 * 4 = 8   3 * 4 = 12  4 * 4 = 16
1 * 5 = 5   2 * 5 = 10  3 * 5 = 15  4 * 5 = 20  5 * 5 = 25
1 * 6 = 6   2 * 6 = 12  3 * 6 = 18  4 * 6 = 24  5 * 6 = 30  6 * 6 = 36
1 * 7 = 7   2 * 7 = 14  3 * 7 = 21  4 * 7 = 28  5 * 7 = 35  6 * 7 = 42  7 * 7 = 49
1 * 8 = 8   2 * 8 = 16  3 * 8 = 24  4 * 8 = 32  5 * 8 = 40  6 * 8 = 48  7 * 8 = 56
 8 * 8 = 64
1 * 9 = 9   2 * 9 = 18  3 * 9 = 27  4 * 9 = 36  5 * 9 = 45  6 * 9 = 54  7 * 9 = 63
 8 * 9 = 72  9 * 9 = 81
```