python

列表推导式

result = [expr for val in collection if condition]

列表推导式

```
In  [1]: strings = ['a', 'b', 'c', 'as', 'bat', 'car', 'dove', 'python', 'love', 'pythove']
         l = [val for val in strings if 'ove' in val]
         d = {key:val for val, key in enumerate(strings) if 'ove' in key}
         s = {val for val in strings if 'ove' in val}
         print('list is {0} \ndict is {1} \nset  is {2}'.format(l, d, s))

         list is ['dove', 'love', 'pythove']
         dict is {'dove': 6, 'love': 8, 'pythove': 9}
         set  is {'pythove', 'dove', 'love'}
```

列表嵌套推导式

```
In  [2]: tuples = [(1,2,3),(4,5,6),(7,8,9)]
         [val2 for val1 in tuples for val2 in val1]

Out[2]:  [1, 2, 3, 4, 5, 6, 7, 8, 9]

In  [3]: [[val2 for val2 in val1] for val1 in tuples ]

Out[3]:  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

函数

def function(vars) : ... return expr : 一般函数
function = lambda vars : expr : ***lambda*** 函数

一般函数

```
In  [4]: def f(x, y, z=2):
             if z > 1:
                 return z*(x+y)
             else:
                 return z*(x-y)
         f(5, 4, 1), f(5, 4)

Out[4]:  (1, 18)

In  [5]: def f():
             a = 5
             b = 6
             c = 7
             return a, b, c
         f()

Out[5]:  (5, 6, 7)
```

函数的嵌套调用

In [6]:
```
import re

#常规函数
def clean_strings_v1(strings):
    result = []
    for string in strings:
        string = string.strip()        # 等价于string = str.strip(string)
        string = re.sub('[!#?]', '', string)
        string = string.title()
        result.append(string)
    return result

# 函数嵌套调用
def remove_marks(string):               # 正则化函数
    return re.sub('[!#?]', '', string) # 正则表达式
def clean_strings_v2(strings, opts):  # 清理操作函数, strings:操作对象, opts:操作函数
    result = []
    for string in strings:
        for function in opts:           # 遍历opts中的函数依次使用
            string = function(string)  # 调用函数function, function的使用等价于function所指代的函数
        result.append(string)          # function可以是现成的, 也可以是自己定义的
    return result

# 主函数
strings = ['  A???laba#ma ', 'Geo#r?gia!', 'Geor?#gia', 'georgia',
           'FlOrIda', 'south      carolina##', 'West    virginia?']
clearn_opts = [str.strip, remove_marks, str.title]  # 函数指令依次是: 去除空白符, 正则化处理, 正确的大小写

print(clean_strings_v1(strings=strings))
print(clean_strings_v2(strings=strings, opts=clearn_opts))
```

```
['Alabama', 'Georgia', 'Georgia', 'Georgia', 'Florida', 'South      Carolina', 'West    Virginia']
['Alabama', 'Georgia', 'Georgia', 'Georgia', 'Florida', 'South      Carolina', 'West    Virginia']
```

lambda 函数

In [7]:
```
import re
remove_marks = lambda string: re.sub('[!#?]', '', string)
print(clean_strings_v2(strings, opts = [remove_marks]))
```

```
['  Alabama ', 'Georgia', 'Georgia', 'georgia', 'FlOrIda', 'south      carolina', 'West    virginia']
```

In [8]:
```
strings = ['foo', 'card', 'app', 'zzzzz', 'abab']
strings.sort(key = lambda x: len(set(x))) # 根据各字符串不同字母数量进行排序
strings
# 在排序之前, strings里的所有元素都会执行key的函数, 这里指的就是lambda函数
# 计算出值之后, 赋值给key, 然后sort()是根据key值进行排序
```

Out[8]:
```
['zzzzz', 'foo', 'app', 'abab', 'card']
```

生成器及生成器表达式

def generator(vars) : for ... : yield expr
gen = (expr for val in collection if condition)

生成器

构造一种可迭代的对象，从而可对其进行迭代，迭代的过程才是函数被执行的过程

In [9]:
```
def generator(var=10):
    print('Generating squares from 1 to {0}'.format(var ** 2))
    for i in range(1, var+1):
        yield i ** 2

gen = generator(var=3) #生成可迭代的对象——生成器gen
for e in gen: # 迭代生成器时，才开始执行程序
    print(e)
```

```
Generating squares from 1 to 9
1
4
9
```

生成器表达式

In [10]:
```
gen = (x**2 for x in range(1, 3+1))
for e in gen:
    print(e)
```

```
1
4
9
```

In [11]:
```
sum(i**2 for i in range(100))
```

Out[11]:
```
328350
```

In [12]:
```
dict((i, i**2) for i in range(5))
```

Out[12]:
```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

异常处理

try ... except ...

In [13]:
```
def attempt_float(x):
    try:
        return float(x)
    except(ValueError):
        return 'ValueError!'
    except(TypeError):
        return 'TypeError!'
attempt_float('something'), attempt_float((1, 2))
```

Out[13]:
```
('ValueError!', 'TypeError!')
```

In [14]:
```
def attempt_float(x):
    try:
        return float(x)
    except(ValueError, TypeError):
        return 'Error!'
attempt_float('something'), attempt_float((1, 2))
```

Out[14]:
```
('Error!', 'Error!')
```

In [15]:
```
f = open('try.txt', 'w')
try:
    f.writelines(x for x in ['x', 'y', 'z'])
except:
    print('Failed')
else:
    print('Succeeded')
finally:
    f.close()
```

```
Succeeded
```

文件

f = open(path, mode, encoding) ... f.close() : 文件打开

In [16]:
```
import os
path = os.getcwd()+'\\file.txt'
```

In [17]:
```
f = open(path)
print([line for line in f])
f.close()
```

```
['致橡树：\n', '我如果爱你——\n', '绝不像攀援的凌霄花，\n', '借你的高枝炫耀自己；\n', '我如果爱你——\n', '绝不学痴情
的鸟儿，\n', '为绿荫重复单调的歌曲；\n', '也不止像泉源，\n', '常年送来清凉的慰藉；\n', '也不止像险峰，\n', '增加你的高
度，衬托你的威仪。\n', '甚至日光，\n', '甚至春雨。']
```

In [18]:
```
f = open(path)
print([line.rstrip() for line in f]) # rstrip删除字符串末尾的指定字符
f.close()
```

```
['致橡树：', '我如果爱你——', '绝不像攀援的凌霄花，', '借你的高枝炫耀自己；', '我如果爱你——', '绝不学痴情的鸟儿，',
'为绿荫重复单调的歌曲；', '也不止像泉源，', '常年送来清凉的慰藉；', '也不止像险峰，', '增加你的高度，衬托你的威仪。',
'甚至日光，', '甚至春雨。']
```

with open(path) as f : : 文件打开

In [19]:
```
with open(path) as f:
    print([line.rstrip() for line in f])
```

```
['致橡树：', '我如果爱你——', '绝不像攀援的凌霄花，', '借你的高枝炫耀自己；', '我如果爱你——', '绝不学痴情的鸟儿，',
'为绿荫重复单调的歌曲；', '也不止像泉源，', '常年送来清凉的慰藉；', '也不止像险峰，', '增加你的高度，衬托你的威仪。',
'甚至日光，', '甚至春雨。']
```

读取模式

jupyter

文件读写方法

jupyter

f.tell( ), f.seek(position), f.read(size), f.readlines(hint)

In [20]:
```
with open(path) as f:
    print(f.tell()) # 从0开始
    l = f.read(9) # 从当前位置向前读取9个字符
    print(f.tell()) #读取的字节数
    f.seek(8)  #定位文件中的位置，单位：字节
    print(f.tell())
    print(f.read(5))
```

```
0
18
8

我如果爱
```

f.write(text), f.writelines(lines)

In [21]:
```
path_out = os.getcwd() + '\\out.txt'
with open(path_out, mode='w') as f_out:
    f_in = open(path)
    f_out.write('ZHI XIANG SHU\n')
    f_out.writelines(x for x in f_in)
    f_in.close()
with open(path_out) as f:
    lines = f.readlines(14) # 从当前位置向前读到第14个字符所在的行
    print(lines)
```

```
['ZHI XIANG SHU\n', '致橡树：\n']
```

字符模式（str: unicode）与字节模式（编码: encode）

> 字符：'……'
> 字节：*b*'……'

In [22]:
```
with open(path) as f_str:
    data_str = f_str.read(10)  ##向前读10个字符
with open(path, 'rb') as f_gbk:
    data_gbk = f_gbk.read(10)  ##向前读10个字节
print(data_str)
print(data_str.encode('gbk'))
print(data_gbk)

path_out = os.getcwd() + '\\out_utf8.txt'
with open(path_out, mode='w', encoding='utf8') as f_out:
    f_in = open(path)
    f_out.writelines(x for x in f_in)
    f_in.close()
```

```
致橡树：
我如果爱你
b'\xd6\xc2\xcf\xf0\xca\xf7\xa3\xba\n\xce\xd2\xc8\xe7\xb9\xfb\xb0\xae\xc4\xe3'
b'\xd6\xc2\xcf\xf0\xca\xf7\xa3\xba\r\n'
```

注意：文件是什么编码写的，就应该用对应编码格式读

In [23]:
```
path_gbk = os.getcwd() + '\\file.txt'
path_utf8 = os.getcwd() + '\\out_utf8.txt'
with open(path_utf8, 'r+', encoding='utf8') as f_utf_utf:
    data_utf_utf = f_utf_utf.read(3)
with open(path_utf8, 'r+', encoding='gbk', errors='ignore') as f_utf_gbk:
    data_utf_gbk = f_utf_gbk.read(3)
with open(path_gbk, 'r+', encoding='utf8', errors='ignore') as f_gbk_utf:
    data_gbk_utf= f_gbk_utf.read(3)
with open(path_gbk, 'r+', encoding='gbk') as f_gbk_gbk:
    data_gbk_gbk = f_gbk_gbk.read(3)
print(data_utf_utf)
print(data_utf_gbk)
print(data_gbk_utf)
print(data_gbk_gbk)
```

```
致橡树
鑷存℃

幔

致橡树
```