python之pythonic

实验背景

python是一门优雅 简洁 明确的语言。接触python两年多，处于边学边用的程度，一般是用其他语言的思路来写python，不优雅，不简洁。

有幸读到Jeff Knupp的Writing Idiomatic Python，文中列出了一些pythonic的写法。 本实验列出一些实验君觉得很有用的部分，完整大家可参看以下url。

> http://downloads.niceware.com/TECH-pdf/PythonStyle-Writing_idiomatic_python_3.pdf

实验内容

布尔型的比较

python中的None 0 [] {} ''等都是False，所以在数据异常校验逻辑操作的，不需要再做额外的操作如： if len(list) == 0。而是把==去掉。

```
alist = [1, 2, 3]
if alist:
    print('list is empty')

astr = ''
if astr:
    print('string is empty')

anone = None
if anone:
    print('have a None')

abool = False
if abool: # not abool == False
    print('have a False, ')
```

遍历访问可迭代对象

在遍历可迭代对象同时又需要获取其索引值的时候,可以使用enumerate来直接获取索引和值

之前WedO实验君经常会按照BAD的方式来写。

BAD

```
alist = [1, 2, 3]
anohter_list = [4, 2, 3]
for i in range(len(alist)):
   print(alist[i], another_list[i])
```

GOOD

```
alist = [1, 2, 3]
anohter_list = [4, 2, 3]
for index, value in enumerate(alist):
    print(index, value, anohter_list[index])
```

dict字典访问

python的字典dict简直太优秀了，特别是处理json，需要哪里get哪里。但是可能会出现key不在字典里情况，处理不当可能会报错。不建议用[key]的方式，用get(key)。get的函数默认如果可以不存在返回的是None。其实是可以指定初始化的值的。

```
configuration = {}
log_severity = configuration.get('severity', 'Info')
log_severity # Info
```

推导表达式

推导表达式WedO实验君觉得是最pyhonic的方式，简化循环判断操作。典型的样式[表达式 for 迭代变量 in 可迭代对象 [if 条件表达式]] == [op for xx in xx if xxx]。可多层嵌套

```
some_other_list = range(10)
some_list = [element + 5 for element in some_other_list if element%2 == 0]
some_list
# [5, 7, 9, 11, 13]
```

字符拼接

记住有join这个函数

```
result_list = ['True', 'False', 'File not found']
'|'.join(result_list)
# True|False|File not found
```

unpack

unpack意思把列表等的元素获取出来。 可以用*代替部分的元素集合，用_变量放弃一些元素。

```
some_list = ['a', 'b', 'c', 'd', 'e']
(first, second, *rest, _) = some_list
```

```
print(first) # a
print(rest) # ['c', 'd']
```

使用any/all函数

```
##不推荐
found = False
for item in a_list:
  if condition(item):
    found = True
    break
if found:
  # do something if found...

##推荐
if any(condition(item) for item in a_list):
  # do something if found...
```

尽量使用生成器代替列表

这个沿用python进阶-迭代器和生成器的例子

```
# bad
def normal_iter():
    ret = []
    for i in range(10):
        ret.append(i**2)
    return ret
# good
def generator():
    for i in range(10):
        yield i**2
```

类的 __str__

```
class Point():
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __str__(self):
        return '{0}, {1}'.format(self.x, self.y)
p = Point(1, 2)
print (p)
```

实验结语

总结一下，pythonic的宗旨是在不影响可读性的情况下，写尽可能少的代码。能写一行绝不写良好。希望对大家有帮助！