# 自动化测试项目说明文档

## 1前言

本文档说明了自动化测试编码过程中的规范，使测试代码风格保持一致，容易被理解和被维护。

## 2命名规范

### 2.1命名约定

#### modulename, packagename, ClassName, methodname, ExceptionName, functionname, GLOBALVARNAME, instancevarname, functionparametername, localvarname

归纳

Classes和Exceptions使用CapWords；

Global/Class Constants使用CAPS\_WITH\_UNDER；

其他使用lower\_with\_under；

Internal类在前面加下划线

释义

所谓"内部(Internal)"表示仅模块内可用, 或者, 在类内是保护或私有的.

用单下划线(\_)开头表示模块变量或函数是protected的(使用import \* from时不会包含).

用双下划线(\_\_)开头的实例变量或方法表示类内私有.

将相关的类和顶级函数放在同一个模块里. 不像Java, 没必要限制一个类一个模块.

对类名使用大写字母开头的单词(如CapWords, 即Pascal风格), 但是模块名应该用小写加下划线的方式(如lower\_with\_under.py). 尽管已经有很多现存的模块使用类似于CapWords.py这样的命名, 但现在已经不鼓励这样做, 因为如果模块名碰巧和类名一致, 这会让人困扰.

示例：

文件夹：main\_component

文件：main\_page.py

函数（参数）：def set\_value(test\_value):

变量：loading\_modules = ""

Internal函数：def \_set\_value(test\_value):

Internal变量：\_loading\_modules = ""

### 2.2应该避免的名称

a.单字符名称, 除了计数器和迭代器.

b.包/模块名中的连字符(-)

c.双下划线开头并结尾的名称(Python保留, 例如\_\_init\_\_)

### 2.3类名 使用 名词

class Engine(options):

### 2.4函数名 使用 动宾短语

def get\_style():

### 2.5boolean 类型的变量使用 is 或 has 开头

is\_ready = false

## 3括号

宁缺毋滥的使用括号

除非是用于实现行连接, 否则不要在返回语句或条件语句中使用括号. 不过在元组两边使用括号是可以的

Yes:

if foo:

bar()

while x:

x = bar()

if x and y:

bar()

if not x:

bar()

return foo

for (x, y) in dict.items(): ...

No:

if (x):

bar()

if not(x):

bar()

return (foo)

## 4空行

顶级定义之间空两行, 方法定义之间空一行

顶级定义之间空两行, 比如函数或者类定义. 方法定义, 类定义与第一个方法之间, 都应该空一行. 函数或方法中, 某些地方要是你觉得合适, 就空一行.

## 5空格

按照标准的排版规范来使用标点两边的空格

括号内不要有空格

Yes: spam(ham[1], {eggs: 2}, [])

No: spam( ham[ 1 ], { eggs: 2 }, [ ] )

不要在逗号, 分号, 冒号前面加空格, 但应该在它们后面加(除了在行尾)

Yes: if x == 4:

print x, y

x, y = y, x

No: if x == 4 :

print x , y

x , y = y , x

参数列表, 索引或切片的左括号前不应加空格.

Yes: spam(1)

no: spam (1)

Yes: dict['key'] = list[index]

No: dict ['key'] = list [index]

在二元操作符两边都加上一个空格, 比如赋值(=), 比较(==, <, >, !=, <>, <=, >=, in, not in, is, is not), 布尔(and, or, not). 至于算术操作符两边的空格该如何使用, 需要你自己好好判断. 不过两侧务必要保持一致

Yes: x == 1

No: x<1

当’=’用于指示关键字参数或默认参数值时, 不要在其两侧使用空格

Yes: def complex(real, imag=0.0): return magic(r=real, i=imag)

No: def complex(real, imag = 0.0): return magic(r = real, i = imag)

不要用空格来垂直对齐多行间的标记, 因为这会成为维护的负担(适用于:, #, =等):

Yes:

foo = 1000 # comment

long\_name = 2 # comment that should not be aligned

dictionary = {

"foo": 1,

"long\_name": 2,

}

No:

foo = 1000 # comment

long\_name = 2 # comment that should not be aligned

dictionary = {

"foo" : 1,

"long\_name": 2,

}

## 6注释

确保对模块, 函数, 方法和行内注释使用正确的风格

### 文档字符串

Python有一种独一无二的的注释方式: 使用文档字符串. 文档字符串是包, 模块, 类或函数里的第一个语句.这些字符串可以通过对象的**doc**成员被自动提取, 并且被pydoc所用. (你可以在你的模块上运行pydoc试一把, 看看它长什么样). 我们对文档字符串的惯例是使用三重双引号”“”( PEP-257 ).一个文档字符串应该这样组织: 首先是一行以句号, 问号或惊叹号结尾的概述(或者该文档字符串单纯只有一行). 接着是一个空行. 接着是文档字符串剩下的部分, 它应该与文档字符串的第一行的第一个引号对齐.

### 模块

每个文件应该包含一个许可样板. 根据项目使用的许可(例如, Apache 2.0, BSD, LGPL, GPL), 选择合适的样板.

### 函数和方法

下文所指的函数,包括函数, 方法, 以及生成器.

一个函数必须要有文档字符串, 除非它满足以下条件:

1.外部不可见

2.非常短小

3.简单明了

文档字符串应该包含函数做什么, 以及输入和输出的详细描述. 通常, 不应该描述”怎么做”, 除非是一些复杂的算法. 文档字符串应该提供足够的信息, 当别人编写代码调用该函数时, 他不需要看一行代码, 只要看文档字符串就可以了. 对于复杂的代码, 在代码旁边加注释会比使用文档字符串更有意义.

关于函数的几个方面应该在特定的小节中进行描述记录， 这几个方面如下文所述. 每节应该以一个标题行开始. 标题行以冒号结尾. 除标题行外, 节的其他内容应被缩进2个空格.

#### Args:

列出每个参数的名字, 并在名字后使用一个冒号和一个空格, 分隔对该参数的描述.如果描述太长超过了单行80字符,使用2或者4个空格的悬挂缩进(与文件其他部分保持一致). 描述应该包括所需的类型和含义. 如果一个函数接受foo(可变长度参数列表)或者bar (任意关键字参数), 应该详细列出foo和bar.

#### Returns: (或者 Yields: 用于生成器)

描述返回值的类型和语义. 如果函数返回None, 这一部分可以省略.

#### Raises:

列出与接口有关的所有异常.

def fetch\_bigtable\_rows(big\_table, keys, other\_silly\_variable=None):

"""Fetches rows from a Bigtable.

Retrieves rows pertaining to the given keys from the Table instance

represented by big\_table. Silly things may happen if

other\_silly\_variable is not None.

Args:

big\_table: An open Bigtable Table instance.

keys: A sequence of strings representing the key of each table row

to fetch.

other\_silly\_variable: Another optional variable, that has a much

longer name than the other args, and which does nothing.

Returns:

A dict mapping keys to the corresponding table row data

fetched. Each row is represented as a tuple of strings. For

example:

{'Serak': ('Rigel VII', 'Preparer'),

'Zim': ('Irk', 'Invader'),

'Lrrr': ('Omicron Persei 8', 'Emperor')}

If a key from the keys argument is missing from the dictionary,

then that row was not found in the table.

Raises:

IOError: An error occurred accessing the bigtable.Table object.

"""

pass

### 类

类应该在其定义下有一个用于描述该类的文档字符串. 如果你的类有公共属性(Attributes), 那么文档中应该有一个属性(Attributes)段. 并且应该遵守和函数参数相同的格式.

class SampleClass(object):

"""Summary of class here.

Longer class information....

Longer class information....

Attributes:

likes\_spam: A boolean indicating if we like SPAM or not.

eggs: An integer count of the eggs we have laid.

"""

def \_\_init\_\_(self, likes\_spam=False):

"""Inits SampleClass with blah."""

self.likes\_spam = likes\_spam

self.eggs = 0

def public\_method(self):

"""Performs operation blah."""

### 块注释和行注释

最需要写注释的是代码中那些技巧性的部分. 如果你在下次 代码审查 的时候必须解释一下, 那么你应该现在就给它写注释. 对于复杂的操作, 应该在其操作开始前写上若干行注释. 对于不是一目了然的代码, 应在其行尾添加注释.

# We use a weighted dictionary search to find out where i is in

# the array. We extrapolate position based on the largest num

# in the array and the array size and then do binary search to

# get the exact number.

if i & (i-1) == 0: # true iff i is a power of 2

为了提高可读性, 注释应该至少离开代码2个空格.

另一方面, 绝不要描述代码. 假设阅读代码的人比你更懂Python, 他只是不知道你的代码要做什么.

# BAD COMMENT: Now go through the b array and make sure whenever i occurs

# the next element is i+1

## 7类

如果一个类不继承自其它类, 就显式的从object继承. 嵌套类也一样.

Yes: class SampleClass(object):

pass

class OuterClass(object):

class InnerClass(object):

pass

class ChildClass(ParentClass):

"""Explicitly inherits from another class already."""

No: class SampleClass:

pass

class OuterClass:

class InnerClass:

pass

继承自 object 是为了使属性(properties)正常工作, 并且这样可以保护你的代码, 使其不受 PEP-3000 的一个特殊的潜在不兼容性影响. 这样做也定义了一些特殊的方法, 这些方法实现了对象的默认语义, 包括

\_\_new\_\_, \_\_init\_\_, \_\_delattr\_\_, \_\_getattribute\_\_, \_\_setattr\_\_, \_\_hash\_\_, \_\_repr\_\_, and \_\_str\_\_ .

## 8字符串

即使参数都是字符串, 使用%操作符或者格式化方法格式化字符串. 不过也不能一概而论, 你需要在+和%之间好好判定.

Yes: x = a + b

x = '%s, %s!' % (imperative, expletive)

x = '{}, {}!'.format(imperative, expletive)

x = 'name: %s; score: %d' % (name, n)

x = 'name: {}; score: {}'.format(name, n)

No: x = '%s%s' % (a, b) # use + in this case

x = '{}{}'.format(a, b) # use + in this case

x = imperative + ', ' + expletive + '!'

x = 'name: ' + name + '; score: ' + str(n)

避免在循环中用+和+=操作符来累加字符串. 由于字符串是不可变的, 这样做会创建不必要的临时对象, 并且导致二次方而不是线性的运行时间. 作为替代方案, 你可以将每个子串加入列表, 然后在循环结束后用 .join 连接列表. (也可以将每个子串写入一个 cStringIO.StringIO 缓存中.)

Yes: items = ['<table>']

for last\_name, first\_name in employee\_list:

items.append('<tr><td>%s, %s</td></tr>' % (last\_name, first\_name))

items.append('</table>')

employee\_table = ''.join(items)

No: employee\_table = '<table>'

for last\_name, first\_name in employee\_list:

employee\_table += '<tr><td>%s, %s</td></tr>' % (last\_name, first\_name)

employee\_table += '</table>'

在同一个文件中, 保持使用字符串引号的一致性. 使用单引号’或者双引号”之一用以引用字符串, 并在同一文件中沿用. 在字符串内可以使用另外一种引号, 以避免在字符串中使用. GPyLint已经加入了这一检查.

Yes:

Python('Why are you hiding your eyes?')

Gollum("I'm scared of lint errors.")

Narrator('"Good!" thought a happy Python reviewer.')

No:

Python("Why are you hiding your eyes?")

Gollum('The lint. It burns. It burns us.')

Gollum("Always the great lint. Watching. Watching.")

为多行字符串使用三重双引号”“”而非三重单引号’‘’. 当且仅当项目中使用单引号’来引用字符串时, 才可能会使用三重’‘’为非文档字符串的多行字符串来标识引用. 文档字符串必须使用三重双引号”“”. 不过要注意, 通常用隐式行连接更清晰, 因为多行字符串与程序其他部分的缩进方式不一致.

Yes:

print ("This is much nicer.\n"

"Do it this way.\n")

No:

print """This is pretty ugly.

Don't do this.

"""

## 9导入格式

每个导入应该独占一行

Yes: import os

import sys

No: import os, sys

导入总应该放在文件顶部, 位于模块注释和文档字符串之后, 模块全局变量和常量之前. 导入应该按照从最通用到最不通用的顺序分组:

1.标准库导入 2.第三方库导入 3.应用程序指定导入

每种分组中, 应该根据每个模块的完整包路径按字典序排序, 忽略大小写.

import foo

from foo import bar

from foo.bar import baz

from foo.bar import Quux

from Foob import ar

## 10语句

通常每个语句应该独占一行

不过, 如果测试结果与测试语句在一行放得下, 你也可以将它们放在同一行. 如果是if语句, 只有在没有else时才能这样做. 特别地, 绝不要对 try/except 这样做, 因为try和except不能放在同一行.

Yes:

if foo: bar(foo)

No:

if foo: bar(foo)

else: baz(foo)

try: bar(foo)

except ValueError: baz(foo)

try:

bar(foo)

except ValueError: baz(foo)