**Python 编码规范**

遵循良好的编码风格，可以有效的提高代码的可读性，降低出错几率和维护难度。在团队开发中，使用（尽量）统一的编码风格，还可以降低沟通成本。

网上有很多版本的编码规范，基本上都是遵循 PEP8 的规范：

* [PEP 0008 – Style Guide for Python Code](https://www.python.org/dev/peps/pep-0008/)
* [Google 的 Python 风格指南](http://zh-google-styleguide.readthedocs.org/en/latest/google-python-styleguide/contents/)
* [Python Guide - Code Style](http://docs.python-guide.org/en/latest/writing/style/)
* [Pocoo Styleguide](http://flask.pocoo.org/docs/0.10/styleguide/)

除了在编码时主动遵循规范，还有很多有用的工具：

* IntelliJ IDEA 和 PyCharm 的格式化代码功能
* Google 开源的 Python 文件格式化工具：[github.com/google/yapf](https://github.com/google/yapf)
* pyflakes, pylint 等工具及各种编辑器的插件

本文的内容主要摘自互联网上各种版本的规范，因为公司有些小伙伴代码风格不太好，所以整理了一份算是团队的编码规范。

**缩进**

* 不要使用 tab 缩进
* 使用任何编辑器写 Python，请把一个 tab 展开为 4 个空格
* 绝对不要混用 tab 和空格，否则容易出现 IndentationError

**空格**

* 在 list, dict, tuple, set, 参数列表的 , 后面加一个空格
* 在 dict 的 : 后面加一个空格
* 在注释符号 # 后面加一个空格，但是 #!/usr/bin/python 的 # 后不能有空格
* 操作符两端加一个空格，如 +, -, \*, /, |, &, =
* 接上一条，在参数列表里的 = 两端不需要空格
* 括号（(), {}, []）内的两端不需要空格

**空行**

* function 和 class 顶上两个空行
* class 的 method 之间一个空行
* 函数内逻辑无关的段落之间空一行，不要过度使用空行
* 不要把多个语句写在一行，然后用 ; 隔开
* if/for/while 语句中，即使执行语句只有一句，也要另起一行

**换行**

* 每一行代码控制在 80 字符以内
* 使用 \ 或 () 控制换行，举例：
* **def** **foo**(first, second, third, fourth, fifth,
* sixth, and\_some\_other\_very\_long\_param):
* user = User.objects.filter\_by(first=first, second=second, third=third) \
* .skip(100).limit(100) \
* .all()
* text = ('Long strings can be made up '
* 'of several shorter strings.')

**命名**

* 使用有意义的，英文单词或词组，绝对不要使用汉语拼音
* package/module 名中不要出现 -
* 各种类型的命名规范：

| **Type** | **Public** | **Internal** |
| --- | --- | --- |
| Modules | lower\_with\_under | \_lower\_with\_under |
| Packages | lower\_with\_under |   |
| Classes | CapWords | \_CapWords |
| Exceptions | CapWords |   |
| Functions | lower\_with\_under() | \_lower\_with\_under() |
| Global/Class Constants | CAPS\_WITH\_UNDER | \_CAPS\_WITH\_UNDER |
| Global/Class Variables | lower\_with\_under | \_lower\_with\_under |
| Instance Variables | lower\_with\_under | \_lower\_with\_under (protected) or \_\_lower\_with\_under (private) |
| Method Names | lower\_with\_under() | \_lower\_with\_under() (protected) or \_\_lower\_with\_under() (private) |
| Function/Method Parameters | lower\_with\_under |   |
| Local Variables | lower\_with\_under |   |

**import**

* 所有 import 尽量放在文件开头，在 docstring 下面，其他变量定义的上面
* 不要使用 from foo imort \*
* import 需要分组，每组之间一个空行，每个分组内的顺序尽量采用字典序，分组顺序是：
	1. 标准库
	2. 第三方库
	3. 本项目的 package 和 module
* 不要使用隐式的相对导入（implicit relative imports），可是使用显示的相对导入（explicit relative imports），如 from ..utils import validator，最好使用全路径导入（absolute imports）
* 对于不同的 package，一个 import 单独一行，同一个 package/module 下的内容可以写一起：
* *# bad*
* **import** sys, os, time
* *# good*
* **import** os
* **import** sys
* **import** time
* *# ok*
* **from** flask **import** Flask, render\_template, jsonify
* 为了避免可能出现的命名冲突，可以使用 as 或导入上一级命名空间
* 不要出现循环导入(cyclic import)

**注释**

* 文档字符串 docstring, 是 package, module, class, method, function 级别的注释，可以通过 \_\_doc\_\_ 成员访问到，注释内容在一对 """ 符号之间
* function, method 的文档字符串应当描述其功能、输入参数、返回值，如果有复杂的算法和实现，也需要写清楚
* 不要写错误的注释，不要无谓的注释
* *# bad 无谓的注释*
* x = x + 1 *# increase x by 1*
* *# bad 错误的注释*
* x = x - 1 *# increase x by 1*
* 优先使用英文写注释，英文不好全部写中文，否则更加看不懂

**异常**

* 不要轻易使用 try/except
* except 后面需要指定捕捉的异常，裸露的 except 会捕捉所有异常，意味着会隐藏潜在的问题
* 可以有多个 except 语句，捕捉多种异常，分别做异常处理
* 使用 finally 子句来处理一些收尾操作
* try/except 里的内容不要太多，只在可能抛出异常的地方使用，如：
* *# bad*
* **try**:
* user = User()
* user.name = "leon"
* user.age = int(age) *# 可能抛出异常*
* user.created\_at = datetime.datetime.utcnow()
* db.session.add(user)
* db.session.commit() *# 可能抛出异常*
* **except**:
* db.session.rollback()
* *# better*
* **try**:
* age = int(age)
* **except** (TypeError, ValueError):
* **return** *# 或别的操作*
* user = User()
* user.name = "leon"
* user.age = age
* user.created\_at = datetime.datetime.utcnow()
* db.session.add(user)
* **try**:
* db.session.commit()
* **except** sqlalchemy.exc.SQLAlchemyError: *# 或者更具体的异常*
* db.session.rollback()
* **finally**:
* db.session.close()
* 从 Exception 而不是 BaseException 继承自定义的异常类

**Class（类）**

* 显示的写明父类，如果不是继承自别的类，就继承自 object 类
* 使用 super 调用父类的方法
* 支持多继承，即同时有多个父类，建议使用 Mixin

**编码建议**

**字符串**

* 使用字符串的 join 方法拼接字符串
* 使用字符串类型的方法，而不是 string 模块的方法
* 使用 startswith 和 endswith 方法比较前缀和后缀
* 使用 format 方法格式化字符串

**比较**

* 空的 list, str, tuple, set, dict 和 0, 0.0, None 都是 False
* 使用 if some\_list 而不是 if len(some\_list) 判断某个 list 是否为空，其他类型同理
* 使用 is 和 is not 与单例（如 None）进行比较，而不是用 == 和 !=
* 使用 if a is not None 而不是 if not a is None
* 用 isinstance 而不是 type 判断类型
* 不要用 == 和 != 与 True 和 False 比较（除非有特殊情况，如在 sqlalchemy 中可能用到）
* 使用 in 操作：
	1. 用 key in dict 而不是 dict.has\_key()
	2. *# bad*
	3. **if** d.has\_key(k):
	4. do\_something()
	5. *# good*
	6. **if** k **in** d:
	7. do\_something()
	8. 用 set 加速 “存在性” 检查，list 的查找是线性的，复杂度 O(n)，set 底层是 hash table, 复杂度 O(1)，但用 set 需要比 list 更多内存空间

**其他**

* 使用列表表达式（[list comprehension](https://www.python.org/dev/peps/pep-0202/)），字典表达式([dict comprehension](https://www.python.org/dev/peps/pep-0274/), Python 2.7+) 和生成器(generator)
* dict 的 get 方法可以指定默认值，但有些时候应该用 [] 操作，使得可以抛出 KeyError
* 使用 for item in list 迭代 list, for index, item in enumerate(list) 迭代 list 并获取下标
* 使用内建函数 sorted 和 list.sort 进行排序
* 适量使用 map, reduce, filter 和 lambda，使用内建的 all, any 处理多个条件的判断
* 使用 defaultdict (Python 2.5+), Counter(Python 2.7+) 等 “冷门” 但好用的标准库算法和数据结构
* 使用装饰器(decorator)
* 使用 with 语句处理上下文
* 有些时候不要对类型做太过严格的限制，利用 Python 的鸭子类型（Duck Type）特性
* 使用 logging 记录日志，配置好格式和级别
* 了解 Python 的 Magic Method：[A Guide to Python’s Magic Methods](http://www.rafekettler.com/magicmethods.html), [Python 魔术方法指南](http://pycoders-weekly-chinese.readthedocs.org/en/latest/issue6/a-guide-to-pythons-magic-methods.html)
* 阅读优秀的开源代码，如 [Flask 框架](https://github.com/mitsuhiko/flask), [Requests for Humans](https://github.com/kennethreitz/requests)
* 不要重复造轮子，查看标准库、PyPi、Github、Google 等使用现有的优秀的解决方案