# Python代码语言规范

1. **Yapf**

使用yapf使得代码得到规范处理，能够处理标准的代码风格

1. **导入**

**定义:**

模块间共享代码的重用机制。

**优点:**

命名空间管理约定十分简单。

**缺点:**

模块名仍可能冲突. 有些模块名太长, 不太方便。

**结论:**

导入时按照标准包进行导入，对于使用较多的包或者模块，直接使用 import 包名

对于只使用具体方法的模块，使用 from 包名 import 模块名 进行引用

对于导入的模块名称较长的问题，采用as进行重命名 如： import tensorflow as tf

1. **全局变量**

要尽可能的避免全局变量。

**定义:**

定义在模块级的变量.

**优点:**

偶尔有用.

**缺点:**

导入时可能改变模块行为, 因为导入模块时会对模块级变量赋值.

**结论:**

避免使用全局变量, 用类变量来代替

1. **异常**

使用异常需要小心

**定义:**

异常是一种跳出代码块的正常控制流来处理错误或者其它异常条件的方式.

**优点:**

正常操作代码的控制流不会和错误处理代码混在一起.

**缺点:**

可能在运行过程中出现错过错误的情况

1. **条件表达式**

经常用在单行函数中

**定义:**

条件表达式是对于if语句的一种更为简短的句法规则. 例如: x = 1 if cond else 2 .

**优点:**

比if语句更加简短和方便.

**缺点:**

比if语句难于阅读.

**结论:**

适用于单行函数. 其他情况下建议使用完整的if语句

1. **True/False的求值**

要尽可能少的或者隐性的使用false

**定义:**

Python在布尔上下文中会将某些值求值为false.简单来说，所有的”空”值都被认为是false. 因此0， None, [], {}, “” 都被认为是false.

**优点:**

增强代码的可读性

**缺点:**

有时候使用不恰当会出现各种错误

**结论:**

①永远不要用==或者!=来比较单件, 比如None. 使用is或者is not.

注意: 当你写下 if x: 时, 你其实表示的是 if x is not None . 例如: 当你要测试一个默认 值是None的变量或参数是否被设为其它值. 这个值在布尔语义下可能是false!

②永远不要用==将一个布尔量与false相比较. 使用 if not x: 代替. 如果你需要区分 false 和None, 你应该用像 if not x and x is not None: 这样的语句.

③对于序列(字符串, 列表, 元组), 要注意空序列是false. 因此 if not seq: 或者 if seq: 比 if len(seq): 或 if not len(seq): 要更好.

④处理整数时, 使用隐式false可能会得不偿失(即不小心将None当做0来处理). 可 以将一个已知是整型(且不是len()的返回结果)的值与0比较.

注意：字符串‘0’回作为‘true’返回

1. **默认参数值**

大部分情况下要尽可能的想到使用默认参数值

**定义:**

可以在函数参数列表的最后指定变量的值, 例如, def maths(a, b = 0): . 如果调用maths时只带一个参数, 则b被设为0.

**优点:**

在代码调用时会更加的便捷，因为在调用的过程中，并不需要你去定义过多的变量

**缺点:**

默认参数值只能在初始调用时默认一次，下次调用会被赋值，默认参数暂时作废

**结论:**

鼓励使用, 不过不要在函数或方法定义中使用可变对象作为默认值.