百度python编码规范

修改历史：

・[强制改为建议] 禁止使用from xxx import yyy语法直接导入类或函数(即yyy只能是module或package，不能是类或函数)。
・[强制改为建议] [PY002] 禁止使用from xxx import *
・[强制改为建议] [PY003] import时必须使用package全路径名(相对PYTHONPATH)，禁止使用相对路径(相对当前路径)，禁止使用sys.path.append('../../')等类似操作改变当前环境变量。
・[强制改为建议] [PY011] 可以使用列表推导。mapping、loop、filter部分单独成行，且最多只能写一行。禁止多层loop或filter。
・[强制改为建议] [PY014] 可以使用lambda函数，但仅限一行之内。
・[强制改为建议] 条件表达式仅用于一行之内，禁止嵌套使用
・[强制改为建议] [PY016] 仅可使用以下基本类型字面常量或常量作为默认参数：整数、bool、浮点、字符串、None
・[强制改为建议] [PY021] 禁止以分号结束语句
・[强制改为建议] [PY022] 一行只能写一条语句，没有例外情况
・[强制改为建议] [PY027] 文件级定义(类或全局函数)之间隔两个空行，类方法之间隔一个空行
・[强制改为建议] [PY028] 圆括号、方括号、花括号内侧都不加空格
・[强制改为建议] [PY029] 参数列表, 索引或切片的左括号前不应加空格
・[强制改为建议] protected成员使用单下划线前缀，private成员使用双下划线前缀

・[强制改为建议] [PY043] 禁止使用双下划线开头，双下划线结尾的名字(类似__init__)
・[强制改为建议] [PY046] 如果一个类没有基类，必须继承自object类。
・[强制改为建议] 除了a+b这种最简单的情况外，应该使用%或format格式化字符串。
・[强制改为建议] 不要使用+=拼接字符串列表，应该使用join。但需确保列表中全是Strings类型，如果有Numbers等其它类型，会报TypeError错误，当不确定时建议仍使用+=拼接字符串。
・[强制改为建议] 用完文件或socket后必须显式关闭句柄。建议使用with语法简化开发
・[强制改为建议] 所有module都必须可导入。如需要执行主程序，必须检查__name__ == '__main__'

1. 前言

1.1. 一般信息[重要必读]

此编码风格指南主要基于Google Python Style Guide[中译版]，结合百度python使用习惯和实际开发情况制定。

这份文档存在的意义是让大家写出统一风格的代码，让百度的模块可维护性和可读性更好；

文档内容可能会与您的喜好冲突， 请尽量用包容的心态来接受; 不合理之处， 请反馈给py-styleguide@baidu.com

1.2. 如何使用本编程规范

1.2.1. 本规范的层次结构

本规范可分为三大部分，分别对Python语法、风格、编程实践作出规定与建议。

每一部分有若干专题，每一专题下有若干条目。

条目是规范的基本组成部分，每一条目由规定、定义、解释、示例、参考等项组成。

1.2.2. 条目的级别和编号

本规范的条目分两个级别:

- [强制]：要求所有程序必须遵守，不得违反
- [建议]：建议遵守，除非确有特殊情况

1.3. The Zen of Python

建议每位python开发人员细读”python之禅”，理解规范背后的思想

```
"""
The Zen of Python

 Beautiful is better than ugly.
 Explicit is better than implicit.
 Simple is better than complex.
 Complex is better than complicated.
 Flat is better than nested.
 Sparse is better than dense.
 Readability counts.
 Special cases aren't special enough to break the rules.
 Although practicality beats purity.
 Errors should never pass silently.
 Unless explicitly silenced.
 In the face of ambiguity, refuse the temptation to guess.
 There should be one-- and preferably only one --obvious way to do it.
 Although that way may not be obvious at first unless you're Dutch.
 Now is better than never.
 Although never is often better than |right| now.
 If the implementation is hard to explain, it's a bad idea.
 If the implementation is easy to explain, it may be a good idea.
 Namespaces are one honking great idea -- let's do more of those!

-- by Tim Peters
"""
```

2. 语言规范

2.1. import

・[建议] 禁止使用fromxxximportyyy语法直接导入类或函数(即yyy只能是module或package，不能是类或函数)。

解释

避免冲突。调用关系简单明了，x.obj表示obj对象定义在模块x中。

示例

YES

```
from Crypto.Cipher import AES

import os

os.unlink(path)
```

NO

```
# os

from os import unlink

unlink(path)
```

・[[建议] [PY002] 禁止使用fromxxximport*

・[[建议] [PY003] import时必须使用package全路径名(相对PYTHONPATH)，禁止使用相对路径(相对当前路径)，禁止使用sys.path.append('../../')等类似操作改变当前环境变量。

解释

避免模块名冲突，查找包更容易。部署时保持项目结构，使用virtualenv等创建隔离的Python环境，并配置需要的PYTHONPATH。

2.2. 异常

・[建议] 可以使用异常。但使用前请务必详细了解异常的行为，谨慎使用

・[强制] [PY004] 禁止使用双参数形式（raiseMyException,'ErrorMessage'）或字符串形式（raise'ErrorMessage'）语法抛异常。

・[强制] 如果需要自定义异常，应该在模块内定义名为Error的异常基类。该基类必须继承自Exception。其它异常类都从该Error类派生而来。

・[强制] 除非重新抛出异常，禁止使用except:捕获所有异常。

・[建议] 除非重新抛出异常，否则不建议捕获Exception或StandardError。如果捕获，必须在日志中记录所捕获异常信息。

・[强制] [PY007] 捕捉异常时，应当使用as语法，禁止使用逗号语法。

・[建议] 建议try中的代码尽可能少。避免catch住未预期的异常，掩藏掉真正的错误。

・[建议] 建议使用finally子句来执行那些无论try块中有没有异常都应该被执行的代码，这对于清理资源常常很有用。例如：文件关闭。

解释

raiseMyException,'ErrorMessage'或raise'ErrorMessage'语法将废弃
用户经常需要以统一的方式处理某模块抛出的所有异常，如果模块开发者自定义的异常没有统一基类，用户处理起来就很麻烦
except:或exceptException:或exceptStandardError:会捕获类似语法错误、Ctrl+C中断等底层异常。而一般情况下这不是用户代码要处理的
逗号式异常捕获语法将废弃

示例

YES

```
raise MyException
raise MyException('Error Message')

class Error(Exception):
    pass

try:
    raise Error
except Error as error:
    pass
```

NO

```
raise 'Error Message'
raise MyException, 'Error Message'

try:
    raise Error
except Error, error:
    pass
```

2.3. 全局变量

・[强制] 禁止使用全局变量。除了以下例外：脚本默认参数模块级常量

・[强制] 如果定义全局变量，必须写在文件头部。

解释

全局变量破坏程序封装性，使代码维护困难

2.4. 构造函数

・[建议] 类构造函数应该尽量简单，不能包含可能失败或过于复杂的操作

解释

复杂的、可能出异常的构造函数通常语义不明确，与调用者的期望不符。且难以维护，容易出错。一般建议把复杂部分拆分到独立的函数中。
什么情况算”过于”复杂，由代码owner和reviewer根据经验自行判定

2.5. 函数返回值

・[强制] [PY004] 函数返回值必须小于等于3个。3个以上时必须通过class/namedtuple/dict等具名形式进行包装

解释

虽然python支持以tuple的形式返回多个值，但返回值过多时需要用户记住返回值顺序，使接口难以使用，容易用错

示例

YES

```
def get_numbers():
    return 1, 2, 3

a, b, c = get_numbers()

class Person(object):
    def __init__(self, name, gender, age, weight):
        self.name = name
        self.gender = gender
        self.age = age
        self.weight = weight
```

YES

```
import collections
Person = collections.namedtuple('Person', 'name gender age weight')

def get_person_info():
    return Person('jjp', 'MALE', 30, 130)

person = get_person_info()
```

NO

```
def get_numbers():
    return 1, 2, 3, 4

a, b, c, d = get_numbers()

def get_person_info():
    return 'jjp', 'MALE', 30, 130

name, gender, age, weight = get_person_info()
```

2.6. 嵌套/局部/内部类或函数

・[建议] 不推荐使用嵌套/局部/内部类或函数

解释

这些机制增加了代码理解难度，考虑到公司的平均python水平，不推荐使用

2.7. 列表推导

・[建议] [PY011] 可以使用列表推导。mapping、loop、filter部分单独成行，且最多只能写一行。禁止多层loop或filter。

解释

复杂的列表推导难以理解，建议转换成对应的for循环

示例

YES

```
result = []
for x in range(10):
    for y in range(5):
        if x * y > 10:
            result.append((x, y))

for x in xrange(5):
    for y in xrange(5):
        if x != y:
            for z in xrange(5):
                if y != z:
                    yield (x, y, z)

return ((x, complicated_transform(x))
      for x in long_generator_function(parameter)
      if x is not None)

squares = [x * x for x in range(10)]

eat(jelly_bean for jelly_bean in jelly_beans
    if jelly_bean.color == 'black')
```

NO

```
result = [(x, y) for x in range(10) for y in range(5) if x * y > 10]

return ((x, y, z)
        for x in xrange(5)
        for y in xrange(5)
        if x != y
        for z in xrange(5)
        if y != z)
```

2.8. 默认迭代器和操作符

・[强制] 对容器或文件的只读遍历，应该使用内置的迭代方法，不要使用返回list的方式遍历。

・[强制] [PY013] 对容器类型，使用in或notin判断元素是否存在。而不是has_key。

解释

可读性和性能更好。例如：file.readlines()会一次性读入文件中所有行，存储到一个list中返回。当文件很大时，性能开销较大
当文件或容器较小时，调用返回list的函数也可以接受。请reviewer视具体情况决定是否通过。
如果遍历过程中需要对容器进行增删，请使用返回list的方式遍历

示例

YES

```
for key in adict: ...
if key not in adict: ...
if obj in alist: ...
for line in afile: ...
for k, v in dict.iteritems(): ...
#
for id in students.keys():
    if students[id].graduated:
        del students[id]
```

NO

```
for key in adict.keys(): ...
if not adict.has_key(key): ...
for line in afile.readlines(): ...
#
for id in students:
    if students[id].graduated:
        del students[id]      # RuntimeError
```

2.9. 生成器

・[建议] 当返回较长列表数据时建议使用yield和generator函数。

解释

生成器函数可以避免返回大列表，占用过多内存、影响性能。同时还可以保持代码的优雅易读。

示例

YES

```
#n
def odds(n):
    for i in xrange(1, n + 1):
        if i % 2 == 1:
            yield i

for i in odds(1000):
    print i
```

NO

```
#n
def odds(n):
    ret = []
    for i in xrange(1, n + 1):
        if i % 2 == 1:
            ret.append(i)
    return ret

for i in odds(1000):
    print i
```

2.10. lambda函数

・[建议] [PY014] 可以使用lambda函数，但仅限一行之内。

解释

lambda函数可以简化开发。但复杂的lambda函数可读性很差，应该避免

2.11. 条件表达式

・[[建议] 条件表达式仅用于一行之内，禁止嵌套使用

解释

语法略小众，其它语言背景开发者(如C++)看起来比较困惑。写复杂了难以阅读维护

示例

```
YES:

#
x = true_value if cond else false_value

# if
if cond:
    x = true_value
else
    x = false_value

# if
if t < 60:
    unit = "seconds"
elif t < 3600:
    unit = "minutes"
else
    unit = "hours"

NO:

#
unit = "seconds" if t < 60 else "minutes" if t < 3600 else "hours"
```

2.12. 参数

2.12.1 默认参数

・[建议] [PY016] 仅可使用以下基本类型字面常量或常量作为默认参数：整数、bool、浮点、字符串、None

解释

以可修改的对象(如list、dict、object等)作为默认参数，可能会被不小心改掉，导致默认值发生变化，产生难以追查的错误
默认值在module加载时求值，而不是每次调用时求值。复杂的默认值可能和预期不符合(例如下边例子中的time.time()和FLAGS.my_things)

示例

```
YES:

def foo(a, b=None):
    if b is None:
        b = []

No:  def foo(a, b=[]):
         ...
No:  def foo(a, b=time.time()):  # The time the module was loaded???
         ...
No:  def foo(a, b=FLAGS.my_thing):  # sys.argv has not yet been parsed...
         ...
```

2.12.2可变参数、关键字参数

・[建议] 善于使用可变参数（*args）、关键字参数（**kwargs），将使你的代码更加简洁

示例

任意的参数

```
def rm_files(*args):
  for file_path in args:
    # do something
    pass


rm_files('a.txt')
rm_files('a.txt', 'b.txt')
files = ['a.txt', 'b.txt', 'c.txt']
rm_files(*files)
```

关键字参数（比如常用的requests库）

```
importrequests

requests.request('GET', 'https://www.baidu.com', cookies={}, headers={})
```

函数定义

```
def request(method, url, **kwargs):

        :param method: method for the new :class:`Request` object.

        :param url: URL for the new :class:`Request` object.

        ……

        :param headers: (optional) Dictionary of HTTP Headers to send with the :class:`Request`.

        :param cookies: (optional) Dict or CookieJar object to send with the :class:`Request`.
```

2.13. 属性(properties)

・[强制] 可以使用property。但禁止在派生类里改写property实现。

解释

由于property是在基类中定义的，默认绑定到基类的实现函数。若允许在派生类中改写property实现，则需要在基类中通过间接方式调用property实现函数。这个方法技巧性太强，可读性差，所以禁止使用。

示例

YES:

```
import math

class Square(object):

    """A square with two properties: a writable area and a read-only perimeter.

    To use:
    >>> sq = Square(3)
    >>> sq.area
    9
    >>> sq.perimeter
    12
    >>> sq.area = 16
    >>> sq.side
    4
    >>> sq.perimeter
    16
    """

    def __init__(self, side):
        self.side = side

    def __get_area(self):
        """Calculates the 'area' property."""
        return self.side ** 2

    def __set_area(self, area):
        """Sets the 'area' property."""
        self.side = math.sqrt(area)

    area = property(__get_area, __set_area,
                    doc="""Gets or sets the area of the square.""")

    @property
    def perimeter(self):
        return self.side * 4
```

NO:

```
class MySquare(Square):
    def __get_area(self):
        return math.pi * self.side ** 2       # overwrite doesn't work
    def __set_area(self, area):
        self.side = math.sqrt(area / math.pi) # overwrite doesn't work
```

2.14. True/False求值

・[建议] 建议显式转换到bool类型，慎用到bool类型的隐式转换。如使用隐式转换，你需要确保充分了解其语义

・[强制] [PY018] 禁止使用==或!=判断表达式是否为None，应该用is或isnotNone

・[强制] 当明确expr为bool类型时，禁止使用==或!=与True/False比较。应该替换为expr或notexpr

・[强制] 判断某个整数表达式expr是否为零时，禁止使用notexpr，应该使用expr==0

解释

python中None、空字符串、0、空tuple、list、dict都会被隐式转换为False，这可能和用户预期的行为不一致。
为便于不太熟悉python语言的其它语言开发者理解python代码，建议”显式”表明到bool类型的转换语义
运算符==或!=的结果取决于__eq__函数，可能出现objisnotNone，但obj==None的情况
ifexpr！=False当expr为None时，会通过检测。一般这不是用户期望的行为from PEP：Yes:ifgreeting:No:ifgreeting==True:Worse:ifgreetingisTrue:
当判断expr是否为0时，若expr为None，not expr也会返回True。一般这不是用户期望的行为

示例

```
YES:

if users is None or len(users) == 0:
    print 'no users'

if foo == 0:
    self.handle_zero()

if i % 10 == 0:
    self.handle_multiple_of_ten()

Cautious:

if not users:
    print "no users"

NO:

if len(users) == 0:
    print 'no users'

if foo is not None and not foo:
    self.handle_zero()

if not i % 10:
    self.handle_multiple_of_ten()
```

2.15. 函数和方法装饰器

・[建议] 建议仅在以下几种情况下使用函数方法装饰器。其它情况下如有明显好处，且不影响代码可维护性，可以谨慎使用@property、@classmethod、@staticmethod自动化测试第三方库要求使用的装饰器

解释

decorator太灵活，可以任意改变函数参数和返回值，容易产生非预期行为。滥用容易使代码可维护性性变差
decorator在import时执行，decorator代码执行中的错误很难处理和恢复

3. 风格规范

3.1. 分号

・[建议] [PY021] 禁止以分号结束语句

・[建议] [PY022] 一行只能写一条语句，没有例外情况

3.2. 行列长度

・[强制] [PY023] 每行不得超过120个字符

・[强制] [PY024] 函数长度不得超过120行

解释

现在宽屏比较流行，所以从传统的80个字符限制扩展到120个字符
函数太长一般说明函数定义不明确，程序结构划分不合理，不利于维护

示例

字符串太长时可以分成两个字符串，python会自动join相临的字符串

```
x = ('This will build a very long long '
     'long long long long long long string')
```

3.3. 括号

・[建议] 除非用于明确算术表达式优先级、tuple或者隐式行连接，否则尽量避免冗余的括号。

解释

python倾向于直观、简洁的代码

示例

```
YES:

if foo:
    bar()
while x:
    x = bar()
if x and y:
    bar()
if not x:
    bar()
return foo
for (x, y) in dict.iteritems(): ...

NO:

if (x):
    bar()
if not(x):
    bar()
return (foo)
```

3.4. 缩进

・[强制] 使用4个空格缩进，禁止使用tab缩进。

・[强制] 把单行内容拆成多行写时，要么与首行保持对齐；要么首行留空，从第二行起统一缩进 4 个空格；为与后面的代码区分，可以使用8空格缩进。

解释

不同编辑器对TAB的设定可能不同，使用TAB容易造成在一些编辑器下代码混乱，所以建议一率转换成空格。
在vim下，建议打开如下设置：:settabstop=4设定tab宽度为4个字符:setshiftwidth=4设定自动缩进为4个字符:setexpandtab用space自动替代tab

示例

```
YES:

# Aligned with opening delimiter
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Aligned with opening delimiter in a dictionary
foo = {
    long_dictionary_key: value1 +
                         value2,
    ...
}

# 4-space hanging indent; nothing on first line
foo = long_function_name(
    var_one, var_two, var_three,
    var_four)

# 4-space hanging indent in a dictionary
foo = {
    long_dictionary_key:
        long_dictionary_value,
    ...
}
```

```
NO:

# Stuff on first line forbidden
foo = long_function_name(var_one, var_two,
    var_three, var_four)

# 2-space hanging indent forbidden
foo = long_function_name(
  var_one, var_two, var_three,
  var_four)

# No hanging indent in a dictionary
foo = {
    long_dictionary_key:
        long_dictionary_value,
        ...
}
```

3.5. 空行

・[建议] [PY027] 文件级定义(类或全局函数)之间隔两个空行，类方法之间隔一个空行

3.6. 空格

・[建议] [PY028] 圆括号、方括号、花括号内侧都不加空格

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

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

・[建议] [PY029] 参数列表, 索引或切片的左括号前不应加空格

```
Yes: spam(1)

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

No:  spam (1)

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

・[强制] [PY030] 逗号、分号、冒号前不加空格，后边加一个空格

```
Yes:

if x == 4:
    print x, y
x, y = y, x

No:

if x == 4 :
    print x , y
x , y = y , x
```

・[强制] [PY031] 所有二元运算符前后各加一个空格

```
Yes: x == 1

No:  x<1
```

・[强制] [PY032] 关键字参数或参数默认值里的等号前后不加空格

```
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)
```

3.7. 注释

・[强制] [PY033] 使用docstring描述module、function、class和method接口。docstring必须用三个双引号括起来。

・[强制] 对外接口部分必须用docstring描述，内部接口视情况自行决定是否写docstring。

・[强制][PY034] 接口的docstring描述至少包括功能简介、参数、返回值。如果可能抛出异常，必须注明。

・[强制] 每个文件都必须有文件声明，文件声明必须包括以下信息：版权声明，功能和用途简介，修改人及联系方式。

・[建议]TODO注释格式必须为：

```
# TODO: $()$(YYYY-MM-DD)$
```

定义

模块、类、函数的第一个逻辑行的字符串称为文档字符串(docstring)。

解释

docstring可以通过help查看，可以通过pydoc自动提取文档。编写docstring是个非常好的习惯。

示例

module注释

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
Copyright (c) 2014 Baidu.com, Inc. All Rights Reserved

This module provide configure file management service in i18n environment.

Authors: jiangjinpeng(jiangjinpeng@baidu.com)
Date: 2014/04/05 17:23:06
"""
```

function注释

```
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
```

class注释

```
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."""
```

TODO注释

```
# TODO: Improve performance using concurrent operation. $jiangjinpeng$2014-04-05$
```

3.8. import格式

- [建议] [PY037] 每行只能导入一个库
- [建议] 必须按如下顺序排列import，每部分之间留一个空行：标准库, 第三方库, 应用程序自有库

```
YES:

import os
import sys

from third.party import lib
from third.party import foobar as fb

import my.own.module

NO:

import os, sys
from third.party import lib, foobar
```

3.9. 命名规则

- [强制] [PY039] 类(包括异常)名使用首字母大写驼峰式命名
- [强制] 常量使用全大写字母，单词间用下划线分隔
- [强制] 其它情况(目录/文件/package/module/function/method/variable/parameter)一律使用全小写字母，单词间用下划线分隔
- [建议]protected成员使用单下划线前缀，private成员使用双下划线前缀
- [建议] [PY043] 禁止使用双下划线开头，双下划线结尾的名字(类似__init__)

解释

protected/private名字解释单/双下划线开头的函数或类不会被frommoduleimport*导入双下划线开头的类成员函数/变量会被python内部改写，加上类名前缀。以避免与派生类同名成员冲突单/双下划线都不能真正意义上阻止用户访问，只是module开发者与使用者之间的”约定”双下划线开头、结尾的名字对python解释器有特殊意义，可能与内部关键字冲突

示例

```
ClassName, ExceptionName

GLOBAL_CONSTANT_NAME, CLASS_CONSTANT_NAME,

module_name, package_name, method_name, function_name, global_var_name, instance_var_name, function_pa
rameter_name, local_var_name

_InternalClassName, _INTERNAL_CONSTANT_NAME, _internal_function_name, _protected_member_name, __privat
e_member_name
```

4. 编程实践

4.1. python解释器

・[建议] 模块的主程序必须以#!/usr/bin/env python开头。如果明确只支持某个python版本，请带上python版本号

・[建议] 模块可以自带某个特定版本的python环境一起发布。需要在程序的启动脚本中指定具体使用的python解释器程序

・[建议] 推荐使用2.7版本(含)以上的python解释器

解释

> python的安装位置在不同服务器上可能有差异，所以建议用env python的方式启动，使用当前环境下的默认python
> 百度不同服务器上安装的python版本可能有差异，如果对python程序的可移植性要求很高，可以自带python环境发布
> Python官网的说明：Python 2.x is legacy, Python 3.x is the present and future of the language。但python2和3的差异很大，还有很多程序只支持2.x。而2.7是2.x系列的最后一个版本。因此，我们推荐如果使用2.x系列，尽量选择2.7。

4.2. 文件编码

・[强制] 如果文件包含非ASCII字符，必须在文件前两行标明字符编码。

・[强制] 只能使用UTF-8或GB18030编码。推荐使用UTF-8编码，如果项目确有需要，可以使用GB18030

解释

> python默认使用ASCII编码解析代码文件。若代码中含有非ASCII字符，无论在字符串中还是注释中，都会导致python异常。必须在文件头标明，告知python解释器正确的字符编码
> UTF8编码是自同步编码，进行字符串处理更方便、通用。但百度内部各种代码和数据都以GB18030编码为主，因此也允许使用GB18030编码。但需要注意GB18030字符串在子串查找时可能匹配到非字符边界。进行此种操作时建议转换到unicode或utf-8处理。

```
# -*- coding: utf-8 -*-
```

・[建议] 如非必要，应尽量避免使用reload(sys);sys.setdefaultencoding('utf-8')

解释

reload(sys)，虽然看似解决了编码问题，但只是将问题隐藏了，还带来了其它风险：

1. 字符串拼接会有隐式的类型转换，变成unicode，因为系统不会报错，不容易发现

2. py2中大部分系统、第三方库方法都只支持传入str类型，unicode会出乎意料的结果，比如urllib2下会导致content-length计算错误，导致接受方只接收到了部分数据

3. 运行时可能影响第三方库

其它问题可参考社区讨论：https://stackoverflow.com/questions/3828723/why-should-we-not-use-sys-setdefaultencodingutf-8-in-a-py-script

https://stackoverflow.com/questions/28657010/dangers-of-sys-setdefaultencodingutf-8

建议做法：程序内部统一使用unicode，外部输入参数尽早转换为unicode，最后输出/调用第三方方法时再转换为str

4.3. 类继承

・[建议] [PY046] 如果一个类没有基类，必须继承自object类。

解释

> 若不继承自object，property将不能正常工作，且与python3不兼容。

示例

```
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
```

4.4. 字符串格式化与拼接

・[建议] 除了a+b这种最简单的情况外，应该使用%或format格式化字符串。

解释

复杂格式化使用%或format更直观

```
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)
```

・[建议] 不要使用+=拼接字符串列表，应该使用join。但需确保列表中全是Strings类型，如果有Numbers等其它类型，会报TypeError错误，当不确定时建议仍使用+=拼接字符串。

解释

python中字符串是不可修改对象。每次+=会创建一个新的字符串，性能较差。

```
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>'
```

4.5. 文件和socket

・[建议] 用完文件或socket后必须显式关闭句柄。建议使用with语法简化开发

解释

虽然文件或socket对象析构时会自动关闭文件，但python什么时候清理对象是不确定的。依赖自动回收可能导致文件句柄耗尽

示例

```
with open("hello.txt") as hello_file:
    for line in hello_file:
        print line
```

4.6. 主程序

・[建议] 所有module都必须可导入。如需要执行主程序，必须检查__name__=='__main__'

解释

若模块不可导入会导致pydoc或单测框架失败

示例

```
def main():
    ...

if __name__ == '__main__':
    main()
```

4.7. 单元测试

・[建议] 推荐使用UnitTests做单元测试。是否需要做单元测试以及目标单测覆盖率由项目负责人自行决定。

・[建议] 推荐测试代码放在单独的test目录中。如果被测试代码文件名为xxx.py，那么测试代码文件应该被命名为xxx_test.py

示例

简单的sqrt测试：math_test.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import unittest

import math

class MathTestCase(unittest.TestCase):
    def test_sqrt(self):
        self.assertEqual(math.sqrt(4) * math.sqrt(4), 4)

if __name__ == "__main__":
    unittest.main()
```

4.8. 日志输出

・[建议] 推荐使用python自带的logging库打印日志。

・[建议] 推荐默认日志格式："%(levelname)s:%(asctime)s:%(filename)s:%(lineno)d*%(thread)d%(message)s", 时间格式："%Y-%m-%d%H:%M:%S"

・[建议] 推荐线上程序使用两个日志文件：一个专门记录warning/error/critical日志，另一个记录所有日志。

解释

日志格式尽量与百度传统的ullog/comlog保持一致。方便统一日志处理
独立的warning/error/critical日志方便发现、追查线上问题，符合百度习惯

示例

log.py

```
import os
import logging
import logging.handlers

def init_log(log_path, level=logging.INFO, when="D", backup=7,
             format="%(levelname)s: %(asctime)s: %(filename)s:%(lineno)d * %(thread)d %(message)s",
             datefmt="%m-%d %H:%M:%S"):
    """
 init_log - initialize log module

 Args:
 log_path - Log file path prefix.
 Log data will go to two files: log_path.log and log_path.log.wf
 Any non-exist parent directories will be created automatically
 level - msg above the level will be displayed
 DEBUG < INFO < WARNING < ERROR < CRITICAL
 the default value is logging.INFO
 when - how to split the log file by time interval
 'S' : Seconds
 'M' : Minutes
 'H' : Hours
 'D' : Days
 'W' : Week day
 default value: 'D'
 format - format of the log
 default format:
 %(levelname)s: %(asctime)s: %(filename)s:%(lineno)d * %(thread)d %(message)s
 INFO: 12-09 18:02:42: log.py:40 * 139814749787872 HELLO WORLD
 backup - how many backup file to keep
 default value: 7

 Raises:
 OSError: fail to create log directories
 IOError: fail to open log file
 """
    formatter = logging.Formatter(format, datefmt)
    logger = logging.getLogger()
    logger.setLevel(level)

    dir = os.path.dirname(log_path)
    if not os.path.isdir(dir):
        os.makedirs(dir)

    handler = logging.handlers.TimedRotatingFileHandler(log_path + ".log",
                                                        when=when,
                                                        backupCount=backup)
    handler.setLevel(level)
    handler.setFormatter(formatter)
    logger.addHandler(handler)

    handler = logging.handlers.TimedRotatingFileHandler(log_path + ".log.wf",
                                                        when=when,
                                                        backupCount=backup)
    handler.setLevel(logging.WARNING)
    handler.setFormatter(formatter)
    logger.addHandler(handler)
```

你可以把上面的代码拷贝到自己的项目中。在程序初始化时，调用init_log即可使日志打印符合规范

```
import log
def main():
    log.init_log("./log/my_program")  # ./log/my_program.log./log/my_program.log.wf7
    logging.info("Hello World!!!")
```

参考

*logging — Logging facility for Python

5. 工具支持

```
    for key in adict.keys(): ...
    if not adict.has_key(key): ...
    for line in afile.readlines(): ...
    #
    for id in students:
        if students[id].graduated:
            del students[id]      # RuntimeError
```

对所有标记的有“eagle支持”的规则，大家可以本地调用客户端检查代码是否符合这部分规则，另外在发起代码评审（cooder）时也将自动触发检查，结果会以行间评论的形式插入到代码评审中，帮助作者检查代码、帮助评审人评审代码，详细内容点击查看