嵌入式 MicroPython 开发规范

中北大学工程训练中心 A413 实验室

嵌入式 MicroPython 开发规范

Emile:1069653183@qq.com

QQ：1069653183

Tel:19834401260

修订历史

日期	版本	描述	作者
2023 年 02 月 23 日	1.0	创建开发规范文档	李子圣

一、编码

所有的 Python 脚本文件都应在文件头标上如下标识或其兼容格式的标识：

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

设置编辑器，默认保存为 utf-8 格式。

二、文件头信息

```
'''
    @Project  :
    @File     :
    @description:
    @Author   :
    @Date     :
'''
```

包括工程名字、文件名称、文件功能描述、作者以及创建日期。

三、函数注释

```
# 函数功能描述
def Function(Param1,Param2,...):
    '''
    :description                  : 函数功能
    :param Param1[数据类型]: 参数含义
    :param Param2[数据类型]: 参数含义
    : return       [数据类型]: 参数含义
    '''
    ...
    return
```

四、格式

4.1 缩进

Python 依赖缩进来确定代码块的层次，行首空白符主要有两种：tab 和空格，但严禁两者混用。我们使用 4 个空格的 tab 进行缩进。

4.2 空格

空格在 Python 代码中是有意义的，因为 Python 的语法依赖于缩进，在行首的空格称为前导空格。在这一节不讨论前导空格相关的内容，只讨论非前导空格。非前导空格在 Python 代码中没有意义，但适当地加入非前导空格可以增进代码的可读性。

在二元算术、逻辑运算符前后加空格，如：

```
a = b + c
```

“:”用在行尾时前后皆不加空格，如分枝、循环、函数和类定义语言；用在非行尾时两端加空格，如 dict 对象的定义：

```
d = {'key' : 'value'}
```

括号（含圆括号、方括号和花括号）前后不加空格，如：

```
do_something(arg1, arg2)
```

而不是

```
do_something( arg1, arg2 )
```

逗号后面加一个空格，前面不加空格；

4.3 空行

适当的空行有利于增加代码的可读性，加空行可以参考如下几个准则：

（1）在类、函数的定义间加空行；

（2）在 import 不同种类的模块间加工行；

（3）在函数中的逻辑段落间加空行，即把相关的代码紧凑写在一起，作为一个逻辑段落，段落间以空行分隔；

五、命名

一致的命名可以给开发人员减少许多麻烦，而恰如其分的命名则可以大幅提高代码的可读性，降低维护成本。

命名应当尽量使用全拼写的单词，缩写的情况有如下两种：

常用的缩写，如 XML、ID 等，在命名时也应只大写首字母，如：

```
class XmlParser(object):pass
```

命名中含有长单词，对某个单词进行缩写。这时应使用约定成俗的缩写方式，如去除元音、包含辅音的首字符等方式，例如：

function 缩写为 fn

text 缩写为 txt

object 缩写为 obj

count 缩写为 cnt

number 缩写为 num，等。

5.1 常量命名

常量名所有字母大写，由下划线连接各个单词，如：

```
WHITE = 0xffffffff
THIS_IS_A_CONSTANT = 1
```

5.2 变量命名

变量名全部小写，由下划线连接各个单词，如：

```
color = WHITE
this_is_a_variable = 1
```

变量名应带有类型信息，采用驼峰命名法。

5.3 函数命名

采用驼峰命名法，函数名的命名规则与变量名相同。

5.4 类

类名单词首字母大写，不使用下划线连接单词，也不加入 C、T 等前缀。如：

```
class ThisIsAClass(object):
passs
```

5.5 模块

模块名全部小写，对于包内使用的模块，可以加一个下划线前缀，

如：

module.py

_internal_module.py

5.6 包

包的命名规范与模块相同。

5.7 特定命名方式

主要是指 __xxx__ 形式的系统保留字命名法。项目中也可以使用这种命名，它的意义在于这种形式的变量是只读的，这种形式的类成员函数尽量不要重载。如：

```
class Base(object):
    def __init__(self, id, parent = None):
    self.__id__ = id
    self.__parent__ = parent
    def __message__(self, msgid):
    # …略
```

其中 __id__、__parent__ 和 __message__ 都采用了系统保留字命名法。

六、语句

6.1 import

import 语句有以下几个原则需要遵守：

import 的次序，先 import Python 内置模块，再 import 第三方模块，最后 import 自己开发的项目中的其它模块；这几种模块中用空行分隔开来。

一条 import 语句 import 一个模块。

当从模块中 import 多个对象且超过一行时，使用如下断行法（此语法 py2.5 以上版本才支持）：

```
from module import (obj1, obj2, obj3, obj4,obj5, obj6)
```

不要使用 from module import *，除非是 import 常量定义模块或其它你确保不会出现命名空间冲突的模块。

6.2 赋值

对于赋值语句，主要是不要做无谓的对齐，如：

```
a = 1                    # 这是一个行注释
variable = 2             # 另一个行注释
fn = callback_function   # 还是行注释
```

没有必要做这种对齐，原因有两点：一是这种对齐会打乱编程时的注意力，大脑要同时处理两件事（编程和对齐）；二是以后阅读和维护都很困难，因为人眼的横向视野很窄，把三个字段看成一行很困

难，而且维护时要增加一个更长的变量名也会破坏对齐。直接这样写为佳：

```
a = 1 # 这是一个行注释
variable = 2 # 另一个行注释
fn = callback_function # 还是行注释
```

6.3 分枝和循环

对于分枝和循环，有如下几点需要注意的：

不要写成一行，如：

```
if not flg: pass
```

和

```
for i in xrange(10): print i
```

都不是好代码，应写成

```
if not flg:
    pass
for i in xrange(10):
    print i
```

注：本文档中出现写成一行的例子是因为排版的原因，不得作为编码中不断行的依据。

条件表达式的编写应该足够 pythonic，如以下形式的条件表达式是拙劣的：

```
if len(alist) != 0: do_something()
```

```
if alist != []: do_something()

if s != "": do_something()

if var != None: do_something()

if var != False: do_something()
```

上面的语句应该写成：

```
if seq: do_somethin() # 注意，这里命名也更改了

if var: do_something()
```

用得着的时候多使用循环语句的 else 分句，以简化代码。