第 5 章 Python 编码规范

俗话说：“没有规矩不成方圆。”编程工作往往都是一个团队协同进行，因而一致的编码规范非常有必要，这样写成的代码便于团队中的其他人员阅读，也便于编写者自己以后阅读。

关于本书的 Python 编码规范借鉴了 Python 官方的 PEP8 编码规范[①]和谷歌 Python 编码规范[②]。

5.1 命名规范

程序代码中到处都是标识符，因此取一个一致并且符合规范的名字非常重要。Python 中命名规范采用多种不同方式。不同的代码元素命名不同，下面将分类说明。

- 包名：全部小写字母，中间可以由点分隔开，不推荐使用下画线。作为命名空间，包名应该具有唯一性，推荐采用公司或组织域名的倒置，如 com.apple.quicktime.v2。
- 模块名：全部小写字母，如果是多个单词构成，可以用下画线隔开，如 dummy_threading。
- 类名：采用大驼峰法命名[③]，如 SplitViewController。
- 异常名：异常属于类，命名同类命名，但应该使用 Error 作为后缀。如 FileNotFoundError。
- 变量名：全部小写字母，如果由多个单词构成，可以用下画线隔开。如果变量应用于模块或函数内部，则变量名可以由单下画线开头；变量类内部私有使用变量名可以双下画线开头。不要命名双下画线开头和结尾的变量，这是 Python 保留的。另外，避免使用小写 L、大写 O 和大写 I 作为变量名。
- 函数名和方法名：命名同变量命名，如 balance_account、_push_cm_exit。
- 常量名：全部大写字母，如果是由多个单词构成，可以用下画线隔开，如 YEAR 和 WEEK_OF_MONTH。

命名规范示例如下：

```
_saltchars = _string.ascii_letters + _string.digits + './'

def mksalt(method=None):
```

① 参考地址 https://www.python.org/dev/peps/pep-0008。

② 参考地址 https://google.github.io/styleguide/pyguide.html。

③ 大驼峰法命名是驼峰命名的一种，驼峰命名是指混合使用大小写字母来命名。驼峰命名分为小驼峰法和大驼峰法。小驼峰法就是第一个单词全部小写，后面的单词首字母大写，如 myRoomCount；大驼峰法是第一个单词的首字母也大写，如 ClassRoom。

```
        if method is None:
            method = methods[0]
        s = '${}$'.format(method.ident) if method.ident else ''
        s += ''.join(_sr.choice(_saltchars) for char in range(method.salt_chars))
        return s

    METHOD_SHA256 = _Method('SHA256', '5', 16, 63)
    METHOD_SHA512 = _Method('SHA512', '6', 16, 106)

    methods = []
    for _method in (METHOD_SHA512, METHOD_SHA256, METHOD_MD5, METHOD_CRYPT):
        _result = crypt('', _method)
        if _result and len(_result) == _method.total_size:
            methods.append(_method)
```

5.2 注释规范


扫码看视频

Python 中注释的语法有三种：单行注释、多行注释和文档注释。本节介绍如何规范使用这些注释。

5.2.1 文件注释

文件注释就是在每一个文件开头添加注释，采用多行注释。文件注释通常包括如下信息：版权信息、文件名、所在模块、作者信息、历史版本信息、文件内容和作用等。

下面看一个文件注释的示例：

```
#
# 版权所有 2015 北京智捷东方科技有限公司
# 许可信息查看 LICENSE.txt 文件
# 描述：
#    实现日期基本功能
# 历史版本：
#    2015-7-22: 创建 关东升
#    2015-8-20: 添加 socket 库
#    2015-8-22: 添加 math 库
#
```

上述注释只是提供了版权信息、文件内容和历史版本信息等，文件注释要根据实际情况包括内容。

5.2.2 文档注释

文档注释就是文档字符串，注释内容能够生成 API 帮助文档，可以使用 Python 官方提供的 pydoc 工具从 Python 源代码文件中提取这些信息，也可以生成 HTML 文件。所有公有的模块、函数、类和方法都应该进行文档注释。

文档注释规范有些“苛刻”。文档注释推荐使用一对三重双引号“"""”包裹起来，注意不推荐使用三重单引号“'''”。文档注释应该位于被注释的模块、函数、类和方法内部的第一条

语句。如果文档注释一行能够注释完成，结束的三重双引号也在同一行。如果文档注释很长，第一行注释之后要留一个空行，然后剩下的注释内容换行要与开始三重双引号对齐，最后结束的三重双引号要独占一行，并与开始三重双引号对齐。

下面代码是 Python 官方提供的 base64.py 文件的一部分。

```
#! /usr/bin/env python3

"""Base16, Base32, Base64 (RFC 3548), Base85 and Ascii85 data encodings"""      ①

# Modified 04-Oct-1995 by Jack Jansen to use binascii module
# Modified 30-Dec-2003 by Barry Warsaw to add full RFC 3548 support
# Modified 22-May-2007 by Guido van Rossum to use bytes everywhere

import re
import struct
import binascii

bytes_types = (bytes, bytearray)  # Types acceptable as binary data

def _bytes_from_decode_data(s):                                                  ②
    if isinstance(s, str):
        try:
            return s.encode('ascii')
        except UnicodeEncodeError:
            raise ValueError('string argument should contain only ASCII characters')
    if isinstance(s, bytes_types):
        return s
    try:
        return memoryview(s).tobytes()
    except TypeError:
        raise TypeError("argument should be a bytes-like object or ASCII "
                        "string, not %r" % s.__class__.__name__) from None


# Base64 encoding/decoding uses binascii

def b64encode(s, altchars=None):
    """Encode the bytes-like object s using Base64 and return a bytes object.  ③

    Optional altchars should be a byte string of length 2 which specifies an  ④
    alternative alphabet for the '+' and '/' characters.  This allows an
    application to e.g. generate url or filesystem safe Base64 strings.
    """                                                                          ⑤
    encoded = binascii.b2a_base64(s, newline=False)
    if altchars is not None:
        assert len(altchars) == 2, repr(altchars)
        return encoded.translate(bytes.maketrans(b'+/', altchars))
return encoded
```

上述代码第①行是只有一行的文档注释，代码第③行～第⑤行是多行的文档注释，注意它的第一行后面是一个空行，代码第④行接着进行注释，它要与开始三重双引号对齐。代码第⑤行是结束三重双引号，它独占一行，而且与开始三重双引号对齐。另外，代码第②行定义的函数没有文档注释，这是因为该函数是模块私有的，通过它的命名 _bytes_from_decode_data 可知它是私有的。

5.2.3 代码注释

程序代码中处理文档注释时还需要在一些关键的地方添加代码注释，文档注释一般是给一些看不到源代码的人看的帮助文档，而代码注释是给阅读源代码的人参考的。代码注释一般采用单行注释和多行注释。示例代码如下：

```
# Base32 encoding/decoding must be done in Python                    ①
_b32alphabet = b'ABCDEFGHIJKLMNOPQRSTUVWXYZ234567'
_b32tab2 = None
_b32rev = None

def b32encode(s):
    """Encode the bytes-like object s using Base32 and return a bytes object.
    """
    global _b32tab2
    # Delay the initialization of the table to not waste memory       ②
    # if the function is never called                                 ③
    if _b32tab2 is None:
        b32tab = [bytes((i,)) for i in _b32alphabet]
        _b32tab2 = [a + b for a in b32tab for b in b32tab]
        b32tab = None

    if not isinstance(s, bytes_types):
        s = memoryview(s).tobytes()
    leftover = len(s) % 5
    # Pad the last quantum with zero bits if necessary                ④
    if leftover:
        s = s + b'\0' * (5 - leftover)  # Don't use += !
    encoded = bytearray()
    from_bytes = int.from_bytes
    b32tab2 = _b32tab2
    for i in range(0, len(s), 5):
        c = from_bytes(s[i: i + 5], 'big')
        encoded += (b32tab2[c >> 30] +           # bits 1 - 10        ⑤
                    b32tab2[(c >> 20) & 0x3ff] + # bits 11 - 20
                    b32tab2[(c >> 10) & 0x3ff] + # bits 21 - 30
                    b32tab2[c & 0x3ff]           # bits 31 - 40
                   )
    # Adjust for any leftover partial quanta
    if leftover == 1:
        encoded[-6:] = b'======'
```

```
elif leftover == 2:
    encoded[-4:] = b'===='
elif leftover == 3:
    encoded[-3:] = b'==='
elif leftover == 4:
    encoded[-1:] = b'='
return bytes(encoded)
```

上述代码第②行～第④行都是单行注释，要求与其后的代码具有一样的缩进级别。代码第①行～第③行是多行注释，注释时要求与其后的代码具有一样的缩进级别。代码第⑤行是尾端进行注释，这要求注释内容极短，应该再有足够的空白（至少两个空格）来分开代码和注释。

5.2.4　使用 TODO 注释

PyCharm 等 IDE 工具都为源代码提供了一些特殊的注释，就是在代码中加一些标识，便于 IDE 工具快速定位代码，TODO 注释就是其中的一种。TODO 注释虽然不是 Python 官方所提供的，但是主流的 IDE 工具也都支持 TODO 注释。有 TODO 注释说明此处有待处理的任务，或代码没有编写完成。示例代码如下：

```
import com.pkg2.hello as module1
from com.pkg2.hello import z

y = 20

# TODO 声明函数
print(y)                    # 访问当前模块变量 y
print(module1.y)            # 访问 com.pkg2.hello 模块变量 y
print(z)                    # 访问 com.pkg2.hello 模块变量 z
```

这些注释可以在 PyCharm 工具的 TODO 视图查看，如果没有打开 TODO 视图，可以将鼠标放到 PyCharm 左下角▣按钮上，弹出如图 5-1 所示的菜单，选择 TODO，打开如图 5-2 所示的 TODO 视图，单击其中的 TODO 可跳转到注释处。

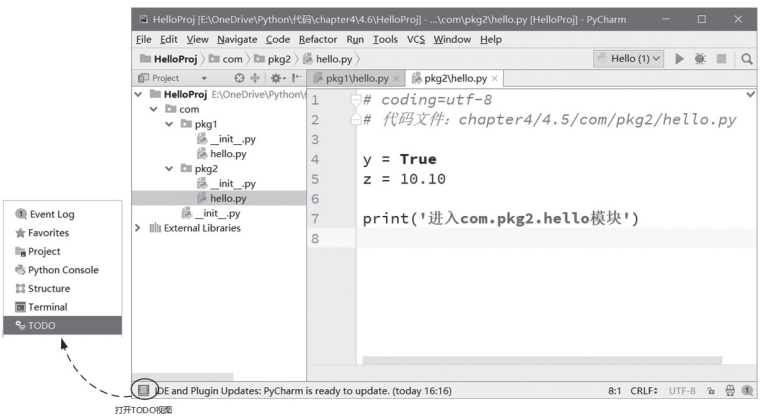

图 5-1　打开 TODO 视图

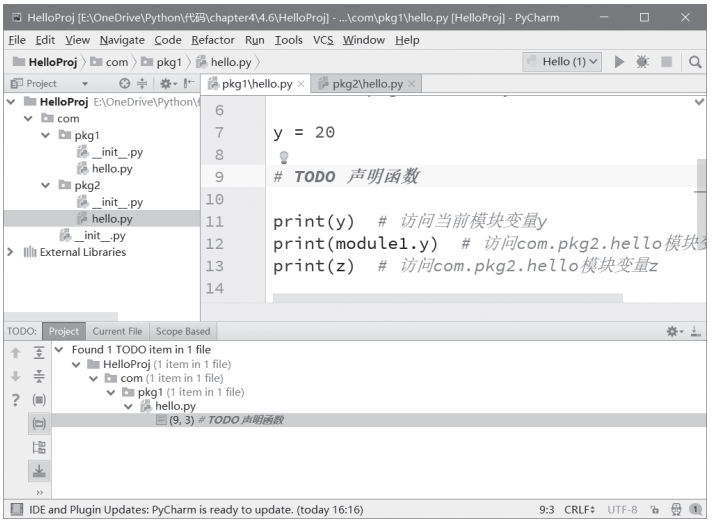

图 5-2　查看 TODO 视图

5.3　导入规范


扫码看视频

导入语句总是放在文件顶部，位于模块注释和文档注释之后，模块全局变量和常量之前。每一个导入语句只能导入一个模块，示例代码如下。

推荐：

```
import re
import struct
import binascii
```

不推荐：

```
import re, struct, binascii
```

但是如果 from import 后面跟有多个代码元素是可以的。

```
from codeop import CommandCompiler, compile_command
```

导入语句应该按照从通用到特殊的顺序分组，顺序是：标准库→第三方库→自己模块。每一组之间有一个空行，而且组中模块是按照英文字母顺序排序的。

```
import io                                    ①
import os
import pkgutil
import platform
import re
import sys
import time                                  ②

from html import unescape                    ③

from com.pkg1 import example                 ④
```

上述代码中导入语句分为三组，代码第①行和第②行是标准库中的模块，注意它的导入顺序是有序的，代码第③行是导入第三方库中的模块，代码第④行是导入自己的模块。

5.4　代码排版

代码排版包括空行、空格、断行和缩进等内容。代码排版内容比较多，工作量很大，也非常重要。


扫码看视频

5.4.1　空行

空行用以将逻辑相关的代码段分隔开，以提高可读性。下面是使用空行的规范。

（1）import 语句块前后保留两个空行，示例代码如下，其中①②处和③④处是两个空行。

```
# Copyright 2007 Google, Inc. All Rights Reserved.
# Licensed to PSF under a Contributor Agreement.

"""Abstract Base Classes (ABCs) according to PEP 3119."""
①
②
from _weakrefset import WeakSet
③
④
```

（2）函数声明之前保留两个空行，示例代码如下，其中①②处是两个空行。

```
from _weakrefset import WeakSet
①
②
def abstractmethod(funcobj):
    funcobj.__isabstractmethod__ = True
    return funcobj
```

（3）类声明之前保留两个空行，示例代码如下，其中①②处是两个空行。

```
①
②
class abstractclassmethod(classmethod):
    __isabstractmethod__ = True

    def __init__(self, callable):
        callable.__isabstractmethod__ = True
        super().__init__(callable)
```

（4）方法声明之前保留一个空行，示例代码如下，其中①处是一个空行。

```
class abstractclassmethod(classmethod):
    __isabstractmethod__ = True
    ①
    def __init__(self, callable):
        callable.__isabstractmethod__ = True
        super().__init__(callable)
```

（5）两个逻辑代码块之间应该保留一个空行，示例代码如下，其中①处是一个空行。

```
def convert_timestamp(val):
    datepart, timepart = val.split(b" ")
    year, month, day = map(int, datepart.split(b"-"))
    timepart_full = timepart.split(b".")
    hours, minutes, seconds = map(int, timepart_full[0].split(b":"))
    if len(timepart_full) == 2:
        microseconds = int('{:0<6.6}'.format(timepart_full[1].decode()))
    else:
        microseconds = 0
    ①
    val = datetime.datetime(year, month, day, hours, minutes, seconds, microseconds)
    return val
```

5.4.2 空格

代码中的有些位置是需要有空格的，这个工作量也很大。下面是使用空格的规范。

（1）赋值符号“=”前后各有一个空格。

```
a = 10
c = 10
```

（2）所有的二元运算符都应该使用空格与操作数分开。

```
a += c + d
```

（3）一元运算符：算法运算符取反“-”和运算符取反“~”。

```
b = 10
a = -b
y = ~b
```

（4）括号内不要有空格，Python 中括号包括小括号“()”、中括号“[]”和大括号“{}”。

推荐：

```
doque(cat[1], {dogs: 2}, [])
```

不推荐：

```
doque(cat[ 1 ], { dogs: 2 }, [  ])
```

（5）不要在逗号、分号、冒号前面有空格，而是要在它们后面有一个空格，除非该符号已经是行尾了。

推荐：

```
if x == 88:
    print x, y
x, y = y, x
```

不推荐：

```
if x == 88 :
```

```
    print x , y
x , y = y , x
```

（6）参数列表、索引或切片的左括号前不应有空格。

推荐：

```
doque(1)
dogs['key'] = list[index]
```

不推荐：

```
doque (1)
dict ['key'] = list [index]
```

5.4.3　缩进

4 个空格常被作为缩进排版的一个级别。虽然在开发时程序员可以使用制表符进行缩进，而默认情况下一个制表符等于 8 个空格，但是不同的 IDE 工具中一个制表符与空格对应个数会有不同，所以不要使用制表符缩进。

代码块的内容相当于首行缩进一个级别（4 个空格），示例如下：

```
class abstractclassmethod(classmethod):
    __isabstractmethod__ = True

    def __init__(self, callable):
        callable.__isabstractmethod__ = True
        super().__init__(callable)

def __new__(mcls, name, bases, namespace, **kwargs):
    cls = super().__new__(mcls, name, bases, namespace, **kwargs)
    for base in bases:
        for name in getattr(base, "__abstractmethods__", set()):
            value = getattr(cls, name, None)
            if getattr(value, "__isabstractmethod__", False):
                abstracts.add(name)
    cls.__abstractmethods__ = frozenset(abstracts)

    return cls
```

5.4.4　断行

一行代码中最多 79 个字符，对于文档注释和多行注释时一行最多 72 个字符，但是如果注释中包含 URL 地址可以不受这个限制。否则，如果超过则需断行，可以依据下面的一般规范断开。

（1）在逗号后面断开。

```
bar = long_function_name(name1, name2,
                         name3, name4)
def long_function_name(var_one, var_two,
                       var_three, var_four):
```

（2）在运算符前面断开。

```
name1 = name2 * (name3 + name4
                         - name5) + 4 * name6
```

（3）尽量不要使用续行符“\”，当有括号（包括大括号、中括号和小括号）则在括号中断开，这样可以不使用续行符。

```
def long_function_name(var_one, var_two,
                       var_three, var_four):
    return var_one + var_two + var_three \          ①
           + var_four


name1 = name2 * (name3 + name4
                         - name5) + 4 * name6

bar = long_function_name(name1, name2,
                         name3, name4)

foo = {
    long_dictionary_key: name1 + name2
                         - name3 + name4 - name5
}

c = list[name2 * name3
      + name4 - name5 + 4 * name6]
```

上述代码第①行使用了续行符进行断行，其他的断行都是在括号中实现的，所以省略了续行符。有时为了省略续行符，会将表达式用小括号括起来，如下代码所示。

```
def long_function_name(var_one, var_two,
                       var_three, var_four):
    return (var_one + var_two + var_three
           + var_four)
```

提示：在 Python 中反斜杠“\”可以作为续行符使用，告诉解释器当前行和下一行是连接在一起的。但在大括号、中括号和小括号中续行是隐式的。

本章小结

通过对本章内容的学习，读者可以了解到 Python 编码规范，包括命名规范、注释规范、导入规范和代码排版等内容。