EMS 高校教务管理系统

Coding Guideline 编码规范

当前版本： *Version1.2*

编制：夏迎琪，宋文宇

审核：罗登，朱迪迪，张云皓，蔡锐，何显

批准：罗登

完成日期： 2019 年 4 月 2 日

北京化工大学，信息科学与技术学院，计科 1605 班。

文档作者联系方式：罗登：（2016011186@mail.buct.edu.cn）.

文件修改控制

本章记录文档修改历史。

版本号说明：本文档采用的版本号格式为 num1.num2 表示。num1 表示为重大修改（初次版本发布、重大更新等），num2 表示小范围修改（修复个别错别字、排版问题、模糊不清项）。

编号	文件状态	版本	修改人	审核人	批准人	修改日期	备注
1	新建	1.0	夏迎琪	各组长	团长	2019 年 3 月 27 日	新建编码规范 Word 文档，编辑加入内容。
2	新建	1.1	罗登	各组长	团长	2019 年 3 月 27 日	用 LaTeX 重新编排，统一格式。
3	增加	1.2	宋文宇	各组长	团长	2019 年 4 月 2 日	加入 JavaScript 语言规范。

目 录

§0.1 导言

§0.1.1 本文档的读者

该文档面向的 3 类读者：

- 老师，了解项目编码规范，便于检查的阅读。
- Python 语言初学者，用于借鉴提高。
- 使用该系统，并需要维护该系统的技术人员。
- 参与开发工作的所有编码人员。

由于 Python 等语言属于脚本语言一类，既可以面向对象开发，又可以解释执行，灵活性较大，因此也很容易导致初学者代码不规范，希望所有开发人员都能仔细阅读该文档。该文档参考资料来自于个人总结和网上资料（多参考 PEP8 规则）。

§0.1.2 目的

开发和编写该文档，主要是出于一下几点考虑：

1. 统一项目编码规范，便于后期整合与生成文档。
2. 防止出现代码层面的沟通困难。
3. 供编程功底较弱或对 Python 和 Javascript 语言不熟悉的同学学习借鉴。

§0.1.3 英文缩写说明

- EMS：Educational Management System 是本开发项目高校教务管理系统的缩写。
- MVC：Model-View-Controler 一种程序设计模式，将开发分为模型层，视图层和控制层，便于解耦合。
- MVT：Model-View-Template 一种程序设计模式，将开发分为模型层，视图层和模板层，视图层负责业务逻辑的实现，模板是将 HTML 和数据结合起来的引擎。Django 中采用的设计模式。
- JS：JavaScript 的缩写，一种用于前端 HTML 界面交互的语言。
- HTML：Hypertext Markup Language，超文本标记语言。用于网页浏览器呈现内容和交互。
- PEP：Python Enhance Propouse，Python 增强议案，又 Python 委员会和众多开源的开发者提出的针对 Python 语言缺陷的增强议案。其中 8 号议案是关于 Python 语言编码规范的提议。
- IDE：Integrated Development Enviroment 集成开发环境。

§0.1.4 专有名称说明

- Python：一种高级的面向对象式的可解释执行的程序设计语言。作为开发的主语言。
- Django：一种流行的基于 Python 的 Web 开发框架。作为开发的主框架，用于处理前端界面和后端数据库连接。

- 解耦合：通过设计模式和架构设计的方法降低系统间各个模块的相互关联程度。
- MySQL：一种开源的流行的数据库名称，作为开发主数据库。
- Git：版本控制软件。
- github：源代码关联工具。
- VUE：一种渐进式的前端设计框架，可以方便操纵网页界面中的元素。

§0.2 代码格式规范

§0.2.1 缩进

1. 空格（Space）是首选的缩进方式。制表符（Tap）只能用于与同样使用制表符缩进的代码保持一致。Python3 不允许同时使用空格和制表符的缩进。混合使用制表符和空格缩进的 Python2 代码应该统一转成空格。
2. 每一级缩进使用 4 个空格。
3. 续行应该与其包裹元素对齐，要么使用圆括号、方括号和花括号内的隐式行连接来垂直对齐，要么使用挂行缩进对齐。当使用挂行缩进时，应该考虑到第一行不应该有参数，以及使用缩进以区分自己是续行。
4. 四空格的规则对于续行是可选的。
5. 当 if 语句的条件部分长到需要换行写的时候，注意可以在两个字符关键字的连接处（比如 if），增加一个空格，再增加一个左括号来创造一个 4 空格缩进的多行条件。这会与 if 语句内同样使用 4 空格缩进的代码产生视觉冲突。PEP 没有明确指明要如何区分 i 发的条件代码和内嵌代码。

§0.2.2 行长度

1. 所有行限制的最大字符数为 79。
2. 文档字符或者注释，每行的最大字符数限制在 72。
3. 限制编辑器窗口宽度可以使多个文件并行打开，并且在使用代码检查工具 (在相邻列中显示这两个版本) 时工作得很好。
4. 避免使用编辑器中默认配置的 80 窗口宽度，即使工具在帮你折行时在最后一列放了一个标记符。
5. 一些团队更喜欢较长的行宽。如果代码主要由一个团队维护，那这个问题就能达成一致，可以把行长度从 80 增加到 100 个字符（更有效的做法是将行最大长度增加到 99 个字符），前提是注释和文档字符串依然已 72 字符折行。
6. Python 标准库比较保守，需要将行宽限制在 79 个字符（文档/注释限制在 72）。
7. 较长的代码行选择 Python 在小括号，中括号以及大括号中的隐式续行方式。通过小括号内表达式的换行方式将长串折成多行。这种方式应该优先使用，而不是使用反斜杠续行。

§0.2.3 断行规则

对应二元运算符来说，总是在运算符前面断行

```
# 不推荐，代码距离操作符太远了
income = (gross_wages +
    taxable_interest +
    (dividends - qualified_dividends) -
    ira_deduction -
    student_loan_interest)
```

```
# 推荐，代码和操作数很容易匹配
income = (gross_wages
    + taxable_interest
    + (dividends - qualified_dividends)
    - ira_deduction
    - student_loan_interest)
```

§0.2.4 空行

- 顶层函数和类的定义，前后用两个空行隔开。
- 类里的方法定义用一个空行隔开。
- 相关的功能组可以用额外的空行（谨慎使用）隔开。一堆相关的单行代码之间的空白行可以省略（例如，一组虚拟实现 dummy implementations）。
- 在函数中使用空行来区分逻辑段（谨慎使用）。
- Python 接受 control-L（即 ^L）换页符作为空格；许多工具把这些字符当作页面分隔符，所以你可以在文件中使用它们来分隔相关段落。请注意，一些编辑器和基于 Web 的代码阅读器可能无法识别 control-L 为换页，将在其位置显示另一个字形。

§0.2.5 源文件编码规则

- Python 核心发布版本中的代码总是以 UTF-8 格式编码。使用 UTF-8 编码的文件不应具有编码声明。
- 在标准库中，非默认的编码应该只用于测试，或者当一个注释或者文档字符串需要提及一个包含内 ASCII 字符编码的作者名字的时候；否则，使用\x,\u,\U , 或者
  N 进行转义来包含非 ASCII 字符。
- Python 3 中，标准库规定了以下策略：Python 标准库中的所有标识符必须使用 ASCII 标识符，并在可行的情况下使用英语单词（在许多情况下，缩写和技术术语是非英语的）。
- 此外，字符串文字和注释也必须是 ASCII。唯一的例外是（a）测试非 ASCII 特征的测试用例，以及（b）作者的名称。作者的名字如果不使用拉丁字母拼写，必须提供一个拉丁字母的音译。

§0.2.6 导入规则

- 当需要多个导入时，我们通常在分开的行中实现
- 导入总是位于文件的顶部，在模块注释和文档字符串之后，在模块的全局变量与常量之前。

- 导入应该按照以下顺序分组：标准库导入、相关第三方库导入、本地应用/库特定导入
- 应该在每一组导入之间加入空行。
- 推荐使用绝对路径导入，如果导入系统没有正确的配置（比如包里的一个目录在 sys.path 里的路径后），使用绝对路径会更加可读并且性能更好（至少能提供更好的错误信息）：然而，显示的指定相对导入路径是使用绝对路径的一个可接受的替代方案，特别是在处理使用绝对路径导入不必要冗长的复杂包布局时：标准库要避免使用复杂的包引入结构，而总是使用绝对路径。不应该使用隐式相对路径导入。
- 当从一个包含类的模块中导入类时

```
from myclass import MyClass
from foo.bar.yourclass import YourClass
```

如果上述写法导致了名字冲突，那么就这么写：

```
import myclass
import foo.bar.yourclass

...
# 使用这种方式来调用
myclass.MyClass
yourclass.YourClass
```

- 避免通配符的导入（`from import *`），因为这样做会不知道命名空间中存在哪些名字，会使得读取接口和许多自动化工具之间产生混淆。对于通配符的导入，有一个防御性的做法，即将内部接口重新发布为公共 API 的一部分（例如，用可选加速器模块的定义覆盖纯 Python 实现的接口，以及重写那些事先不知道的定义）。

§0.2.7 模块级的命名问题

像`__author__`, `__version__`, `__all__` 等这样的模块级命名（也就是名字里有两个前缀下划线和两个后缀下划线），应该放在文档字符串的后面，以及除`from __future__`之外的`import`表达式前面。Python 要求将来在模块中的导入，必须出现在除文档字符串之外的其他代码之前。

§0.2.8 字符串引号问题

- 在 Python 中，单引号和双引号字符串是相同的。PEP 不会为这个给出建议。选择一条规则并坚持使用下去。（请在编码中自我统一）
- 当一个字符串中包含单引号或者双引号字符的时候，使用和最外层不同的符号来避免使用反斜杠，从而提高可读性。
- 对于三引号字符串，总是使用双引号字符来与 PEP 257 中的文档字符串约定保持一致。

§0.2.9 表达式和语句中的空格问题

- 在下列情况下，避免使用无关的空格：
  - 紧跟在小括号，中括号或者大括号后。
  - 紧贴在逗号、分号或者冒号之前。

  - 紧贴在函数参数的左括号之前。
  - 紧贴索引或者切片的左括号之前。
  - 为了和另一个赋值语句对齐，在赋值运算符附件加多个空格。

- 冒号在切片（数组中括号 [ ]）中就像二元运算符，在两边应该有相同数量的空格（把它当做优先级最低的操作符）。在扩展的切片操作中，所有的冒号必须有相同的间距。例外情况：当一个切片参数被省略时，空格就被省略了。

§0.3 注释格式规范

- 当代码更改时，优先更新对应的注释。
- 在句尾结束的时候应该使用两个空格。
- 注释应该是完整的句子。如果一个注释是一个短语或句子，它的第一个单词应该大写，除非它是以小写字母开头的标识符 (永远不要改变标识符的大小写！)。
- 如果注释很短，结尾的句号可以省略。
- 块注释一般由完整句子的一个或多个段落组成，并且每句话结束有个句号。

§0.3.1 块注释

- 块注释通常适用于跟随它们的某些（或全部）代码，并缩进到与代码相同的级别。
- 块注释的每一行开头使用一个 # 和一个空格。
- 块注释内部的段落通过只有一个 # 的空行分隔。

```
# We use a weighted dictionary search to find out where i is in
# the array.  We extrapolate position based on the largest num
# in the array and the array size and then do binary search to
# get the exact number.

if i & (i-1) == 0:        # True if i is 0 or a power of 2.
```

§0.3.2 行内注释

- 有节制地使用行内注释。
- 行内注释是与代码语句同行的注释。
- 行内注释和代码至少要有两个空格分隔。注释由 # 和一个空格开始。
- 事实上，如果状态明显的话，行内注释是不必要的，反而会分散注意力。

为了提高可读性, 注释应该至少离开代码 2 个空格.

另一方面, 绝不要描述代码。假设阅读代码的人比你更懂 Python, 他只是不知道你的代码要做什么。

```
# BAD COMMENT: Now go through the b array and make sure whenever i occurs
# the next element is i+1
```

§0.3.3 文档字符串

Python 有一种独一无二的的注释方式: 使用文档字符串. 文档字符串是**包, 模块, 类或函数里的第一个语句**。这些字符串可以通过对象的`__doc__`成员被自动提取, 并且被 pydoc 所用. (你可以在你的模块上运行 pydoc 试一把, 看看它长什么样).

```
    """Return a foobang

    Optional plotz says to frobnicate the bizbaz first.
    """
```

§0.3.4 函数

一个函数（这里指的函数，包括函数, 方法, 以及生成器.）必须要有文档字符串, 除非它满足以下条件:

1. 外部不可见
2. 非常短小
3. 简单明了

文档字符串应该包含函数做什么, 以及输入和输出的详细描述. 通常, 不应该描述“怎么做”, 除非是一些复杂的算法. 文档字符串应该提供足够的信息, 当别人编写代码调用该函数时, 他不需要看一行代码, 只要看文档字符串就可以了. 对于复杂的代码, 在代码旁边加注释会比使用文档字符串更有意义.

- **Args**: 列出每个参数的名字, 并在名字后使用一个冒号和一个空格, 分隔对该参数的描述. 如果描述太长超过了单行 80 字符, 使用 2 或者 4 个空格的悬挂缩进 (与文件其他部分保持一致). 描述应该包括所需的类型和含义. 如果一个函数接受 `*foo`(可变长度参数列表) 或者`**bar`(任意关键字参数), 应该详细列出`*foo`和`**bar`.
- **Returns: (或者 Yields: 用于生成器)**, 描述返回值的类型和语义. 如果函数返回 None, 这一部分可以省略。
- **Raises**: 列出与接口有关的所有异常。

```
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:
```

```
19
20          {'Serak': ('Rigel VII', 'Preparer'),
21              'Zim': ('Irk', 'Invader'),
22              'Lrrr': ('Omicron Persei 8', 'Emperor')}
23
24          If a key from the keys argument is missing from the dictionary,
25          then that row was not found in the table.
26
27      Raises:
28          IOError: An error occurred accessing the bigtable.Table object.
29  """
30  pass
```

§0.3.5 类

类应该在其定义下有一个用于描述该类的文档字符串. 如果你的类有公共属性 (Attributes), 那么文档中应该有一个属性 (Attributes) 段. 并且应该遵守和函数参数相同的格式.

```
1   class SampleClass(object):
2       """Summary of class here.
3
4       Longer class information....
5       Longer class information....
6
7       Attributes:
8           likes_spam: A boolean indicating if we like SPAM or not.
9           eggs: An integer count of the eggs we have laid.
10      """
11
12      def __init__(self, likes_spam=False):
13          """Inits SampleClass with blah."""
14          self.likes_spam = likes_spam
15          self.eggs = 0
16
17      def public_method(self):
18          """Performs operation blah."""
```

§0.4 命名规范

§0.4.1 包 (Packages) 和模块

模块应该用简短全小写的名字，如果为了提升可读性，下划线也是可以用的。包名也应该使用简短全小写的名字，但不建议用下划线。注：当使用 C 或者 C++ 编写了一个依赖于提供高级（更面向对象）接口的 Python 模块的扩展模块，这个 C/C++ 模块需要一个下划线前缀（例如：_socket）

§0.4.2 类 (Classes) 和异常 (Exceptions)

类名一般使用首字母大写的约定。在接口被文档化并且主要被用于调用的情况下，可以使用函数的命名风格代替。注：对于内置的变量命名有一个单独的约定：大部分内置变量是单个单词（或者两个单词连接在一起），首字母大写的命名法只用于异常名或者内部的常量。异常一般都是类，所有类的命名方法在这里也适用。然而，如果异常确实是一个错误，你需要在异常名后面加上“Error”后缀。

§0.4.3 全局变量

- 约定和函数命名规则一样。
- 规定全局变量只在模块内部使用。
- 通过 `from M import *` 导入的模块应该使用 **all** 机制去防止内部的接口对外暴露，或者使用在全局变量前加下划线的方式，表明这些全局变量是模块内非公有。

§0.4.4 函数名

函数名应该小写，如果想提高可读性可以用下划线分隔。注：大小写混合仅在为了兼容原来主要以大小写混合风格的情况下使用，保持向后兼容性。

§0.4.5 函数和方法参数

- 函数使用下划线分隔小写单词以提高可读性
- 始终要将 self 作为实例方法的的第一个参数。
- 始终要将 cls 作为类静态方法的第一个参数。
- 如果函数的参数名和已有的关键词冲突，在最后加单一下划线比缩写或随意拼写更好。因此 class_ 比 clss 更好。也可以用同义词来避免这种冲突。

§0.4.6 方法名和实例变量 (Instance Variables)

- 在非共有方法和实例变量前使用单下划线。
- 通过双下划线前缀触发 Python 的命名转换规则来避免和子类的命名冲突。
- Python 通过类名对这些命名进行转换：如果类`Foo`有一个叫 `__a`的成员变量，它无法通过`Foo.__a`访问。（执着的用户可以通过`Foo._Foo.__a`访问。）一般来说，前缀双下划线用来避免类中的属性命名与子类冲突的情况。

§0.4.7 常量 (Constants)

常量通常定义在模块级，通过下划线分隔的全大写字母命名。例如：`MAX_OVERFLOW`和`TOTAL`。

§0.4.8 继承的设计

始终要考虑到一个类的方法和实例变量（统称：属性）应该是共有还是非共有。如果存在疑问，那就选非共有；因为将一个非共有变量转为共有比反过来更容易。

公共属性是那些与类无关的客户使用的属性，并承诺避免向后不兼容的更改。非共有属性是那些不打算让第三方使用的属性；你不需要承诺非共有属性不会被修改或被删除。

我们不使用“私有（private）”这个说法，是因为在 Python 中目前还没有真正的私有属性（为了避免大量不必要的常规工作）。

另一种属性作为子类 API 的一部分（在其他语言中通常被称为“protected”）。有些类是专为继承设计的，用来扩展或者修改类的一部分行为。当设计这样的类时，要谨慎决定哪些属性时公开的，哪些是作为子类的 API，哪些只能在基类中使用。

贯彻这样的思想，以下是一些让代码 Pythonic 的准则：

- 公共属性不应该有前缀下划线。如果公共属性名和关键字冲突，在属性名之后增加一个下划线。这比缩写和随意拼写好很多。（然而，尽管有这样的规则，在作为参数或者变量时，‘cls’是表示‘类’最好的选择，特别是作为类方法的第一个参数。）注意 1：参考之前的类方法参数命名建议

§0.4.9 公共和内部的接口

任何向后兼容保证只适用于公共接口，因此，用户清晰地区分公共接口和内部接口非常重要。

文档化的接口被认为是公开的，除非文档明确声明它们是临时或内部接口，不受通常的向后兼容性保证。所有未记录的接口都应该是内部的。为了更好地支持内省（introspection），模块应该使用`__all__`属性显式地在它们的公共 API 中声明名称。将`__all__`设置为空列表表示模块没有公共 API。即使通过`__all__`设置过，内部接口（包，模块，类，方法，属性或其他名字）依然需要单个下划线前缀。如果一个命名空间（包，模块，类）被认为是内部的，那么包含它的接口也应该被认为是内部的。导入的名称应该始终被视作是一个实现的细节。其他模块必须不能间接访问这样的名称，除非它是包含它的模块中有明确的文档说明的 API，例如`os.path` 或者是一个包里从子模块公开函数接口的 `__init__` 模块。

§0.5 Javascript 语言

§0.5.1 嵌入规则

JavaScript 程序应尽量放在.js 文件中，需要调用的时候在 html 文件中以 script 标签的形式包含进来，避免在 html 文件中直接编写 JavaScript 代码。

```
<script src="filename.js">
```

§0.5.2 空格与运算符

通常运算符（`=+-*/`）前后需要添加空格。

```
let x = y + z;
let values = ["Volvo", "Saab", "Fiat"];
```

§0.5.3 代码缩进

通常使用 4 个空格符来缩进代码块，不推荐使用 TAB 键来缩进，因为不同编辑器 TAB 键的解析可能不同。

```
function toCelsius(fahrenheit) {
    return (5 / 9) * (fahrenheit - 32);
}
```

§0.5.4 语句规则

一条语句通常以分号作为结束符。

```
let values = ["Volvo", "Saab", "Fiat"];
```

复杂语句的通用规则：

- 将做花括号放在第一行的结尾。

- 做花括号前添加一个空格。
- 将有花括号独立放在一行。
- 不要以分号结束一个复杂的声明。

```
// 函数
function toCelsius(fahrenheit) {
    return (5 / 9) * (fahrenheit - 32);
}

// 循环
for (let i = 0; i < 5; i++) {
    x += i;
}

// 条件语句
if (time < 20) {
    greeting = "Good␣day";
} else {
    greeting = "Good␣evening";
}
```

§0.5.5 对象规则

短的对象代码可以直接写成一行。

```
let person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

较复杂对象定义的规则：

- 将左花括号与类名放在同一行。
- 冒号与属性值间有个空格。
- 字符串使用双引号，数字不需要。
- 最后一个键值对后面不要添加逗号。
- 将右花括号独立放在一行，并以分号作为结束符号。

```
let person = {
    firstName: "John",
    lastName: "Doe",
    age: 50,
    eyeColor: "blue"
};
```

§0.5.6 每行代码字符小于 80

- 为了方便阅读，每行字符数建议小于 80 个。
- 如果一个 JavaScript 语句超过了 80 个字符，建议在运算符或者逗号后换行。
  - 字符串过长时，利用字符串的 + 符号进行合并。

```
    let str1 = ''
        + '<article>'
        + '<h1>Title␣here</h1>'
```

```
        + '<p>This is a paragraph</p>'
        + '<footer>Complete</footer>'
        + '</article>';
```

– 三元运算符过长时，将条件以及两种结果分写在各行中。

```
    let result = condition
        ? resultA
        : thisIsAVeryVeryLongResult;
```

– 逻辑条件组合过长时，将每个条件独立一行，逻辑运算符放置在行首进行分隔。

```
    if(user.isAuthenticated()
        && user.isInRole('admin')
        && user.hasAuthority('add-admin')
        || user.hasAuthority('delete-admin')
    ) {
        // code
    }
```

– JSON 或数组过长时，进行适当换行，将每一行控制在合理的范围内。

```
    let mapping = {
        one: 1, two: 2, three: 3, four: 4, five: 5,
        six: 6, seven: 7, eight: 8, nine: 9, ten: 10,
        eleven: 11, twelve: 12, thirteen: 13, fourteen: 14,fifteen: 15
    };
```

§0.5.7 命名规则

- 变量、参数和函数使用驼峰法命名, 即除第一个单词之外，其他单词首字母大写。

```
    let lowerCamelCase = "abc";
```

- 全局变量使用全部大写的下划线命名法。

```
    let MAX = 100;
    let MAX_SIZE = 1000;
```

- 常量使用全部大写的下划线命名法。

```
    const PI = 3.14;
    const IS_DEBUG_ENABLED = 1;
```

§0.5.8 HTML 载入外部 JavaScript 文件

使用简洁的格式载入 JavaScript 文件（type 属性不是必须的）。

```
    <script src="myscript.js">
```

§0.5.9 使用 JavaScript 访问 HTML 元素

使用 JavaScript 获取 HTML 元素使用 getElementById 方法。

```
    let obj1 = getElementById("Demo")
    let obj2 = getElementById("demo")
```

§0.5.10 return 语句

return 如果用表达式的执行作为返回值，请把表达式和 return 放在同一行中，以免换行符被误解析为语句的结束而引起返回错误。return 关键字后若没有返回表达式，则返回 undefined。构造器的默认返回值为 this。

§0.5.11 注释

//用作代码的行注释

```
let itemsLayer; //礼物下落层
```

/* ... */形式用作对代码段的注释，或用于较正式的声明中，如函数参数、功能、文件功能等的描述

```
/* Zepto v1.1.6 - zepto event ajax form ie - zeptojs.com/license */
let Zepto = function(){
    //code
}
```

§0.5.12 优先使用箭头函数

使用函数时，优先使用箭头函数

```
// bad
[1, 2, 3].map( function (x) {
    const y = x + 1;
    return x * y;
});

// good
[1, 2, 3].map( (x) => {
    const y = x + 1;
    return x * y;
});
```

§0.5.13 建议每次只声明一个变量

声明变量时，建议每次只声明一个变量，避免一次声明多个变量

```
// bad
let a = 1, b = 2, c = 3;
// good
let a = 1;
let b = 2;
let c = 3;
```

§0.5.14 单引号的使用

建议使用单引号包裹普通字符串

```
let directive = 'No␣identification␣of␣self␣of␣mission.';
```

当字符串出现单引号时，使用``包裹字符串

```
let saying = `Say it ain't␣so`;
```

§0.5.15 避免使用 var 声明

var 是方法作用域，在方法内均可访问该变量

```
function discountedPrices (prices, discount){
    var discounted = [];
    for(var i = 0; i< prices.length; i++){
        var discountedPrice = prices[i] * (1 - discount);
        var finalPrice = Math.round;(discountedPrice * 100) / 100;
        discounted.push(finalPrice);
    }
    console.log(i);  //正常输出
    console.log(discountedPrice);  //正常输出
    console.log(finalPrice);  //正常输出
    return discounted;
}
```

let 是块作用域，即 let 声明的变量只能在局部代码块内被访问

```
function discountedPrices (prices, discount){
    let discounted = [];
    for(let i = 0; i< prices.length; i++){
        let discountedPrice = prices[i] * (1 - discount);
        let finalPrice = Math.round;(discountedPrice * 100) / 100;
        discounted.push(finalPrice);
    }
    console.log(i);  //报错
    console.log(discountedPrice);  //报错
    console.log(finalPrice);  //报错
    return discounted;
}
```

综上，推荐使用 let 声明变量，尽可能避免使用 var 关键字。

§0.6 开发时的一些工具技巧的建议

§0.6.1 使用 PyCharm 支持编码规范

PyCharm 是可以说是最常用也最著名的 Python IDE 了，由 JetBrains 公司开发。有社区版 (Community) 和专业版 (Professional) 两种，其中专业版收费，社区版免费，但是其同时提供了教育许可证的专业版使用，可以使用学校的 edu 邮箱注册，即可使用。专业版 IDE 具有很多开箱即用的 Python 开发效率和生产力工具。

智能提示

PyCharm 自带了很强的语法提示和内嵌的 PEP8 编码规范，编程时只需要关注 IDE 提醒的小灯泡，点击即可查看提示。

在有编码不规范的地方，PyCharm 的右侧滑块处会出现一条小黄线（如果时错误，默认是一条红线），鼠标放上去就可以看到提示。

添加注释

PyCharm 中采用了不同一般的注释方式，也是可以（并且推荐）采用的，因为在 PyCharm 中添加注释非常方便，只需要在函数名或者类名后面输入"""然后敲击回车，既可以生成出带有定义函数参数和返回值的注释模板。

```
def register(request):
    """
    To register an account.
    :param request: a http request
    :return: None
    """
    return
```

这个时候，可以使用快捷键（默认）Ctrl+Q 查看快速文档在右侧边栏看到已经渲染好的函数文档。

同时，还推荐在注释中书写使用样例 Example，在注释中，输入类似于命令行提示符>>>，这时 IDE 能够检测到并且提供智能提示。

```
def register(request):
    """
    To register an account.
    :param request: a http request
    :return: None
    Example:
    >>> import os
    >>> import re
    >>> os.listdir(".")
    """
    return
```

§0.6.2 使用 VS Code 支持编码规范

如果你偏好更加轻量级外观优美的代码编辑器，VS Code 是很多人的青睐之选。其中也不乏 Python 的专业开发者。

在插件商店里面搜索 Python 即可找的关于 Python 的语法高亮、智能提示功能的扩展。但是要获得更多的关于 PEP8 代码规范的支持，可以还需要安装 Flake8 和 PyLint。注意这个不是在 VS Code 中安装，而是通过包管理工具 pip 进行安装。在命令行 cmd 中，激活相应的环境，输入`pip install flake8` 即可安装并获取智能编码规范提示。

对于编码不规范的地方，VS Code 会在对应代码的下面显示出绿色波浪线，鼠标放在上面悬停，即可以查看具体的提示。