PEP 8 风格指南

PEP 是 Python Enhancement Proposal 的缩写，通常翻译为“Python 增强提案”。每个 PEP 都是一份为 Python 社区提供的指导 Python 往更好的方向发展的技术文档，其中的第 8 号增强提案（PEP 8）是针对 Python 语言编订的代码风格指南。尽管我们可以在保证语法没有问题的前提下随意书写 Python 代码，但是在实际开发中，采用一致的风格书写出可读性强的代码是每个专业的程序员应该做到的事情，也是每个公司的编程规范中会提出的要求，这些在多人协作开发一个项目（团队开发）的时候显得尤为重要。我们可以从 Python 官方网站的 PEP 8 链接中找到该文档，下面我们对该文档的关键部分做一个简单的总结。

空格的使用

1. 使用空格来表示缩进而不要用制表符（Tab）。这一点对习惯了其他编程语言的人来说简直觉得不可理喻，因为绝大多数的程序员都会用 Tab 来表示缩进，但是要知道 Python 并没有像 C/C++或 Java 那样的用花括号来构造一个代码块的语法，在 Python 中分支和循环结构都使用缩进来表示哪些代码属于同一个级别，鉴于此 Python 代码对缩进以及缩进宽度的依赖比其他很多语言都强得多。在不同的编辑器中，Tab 的宽度可能是 2、4 或 8 个字符，甚至是其他更离谱的值，用 Tab 来表示缩进对 Python 代码来说可能是一场灾难。
2. 和语法相关的每一层缩进都用 4 个空格来表示。
3. 每行的字符数不要超过 79 个字符，如果表达式因太长而占据了多行，除了首行之外的其余各行都应该在正常的缩进宽度上再加上 4 个空格。
4. 函数和类的定义，代码前后都要用两个空行进行分隔。
5. 在同一个类中，各个方法之间应该用一个空行进行分隔。
6. 二元运算符的左右两侧应该保留一个空格，而且只要一个空格就好。

标识符命名

PEP 8 倡导用不同的命名风格来命名 Python 中不同的标识符，以便在阅读代码时能够通过标识符的名称来确定该标识符在 Python 中扮演了怎样的角色（在这一点上，Python 自己的内置模块以及某些第三方模块都做得并不是很好）。

1. 变量、函数和属性应该使用小写字母来拼写，如果有多个单词就使用下划线进行连接。
2. 类中受保护的实例属性，应该以一个下划线开头。
3. 类中私有的实例属性，应该以两个下划线开头。
4. 类和异常的命名，应该每个单词首字母大写。
5. 模块级别的常量，应该采用全大写字母，如果有多个单词就用下划线进行连接。
6. 类的实例方法，应该把第一个参数命名为 `self` 以表示对象自身。
7. 类的类方法，应该把第一个参数命名为 `cls` 以表示该类自身。

表达式和语句

在 Python 之禅（可以使用 `import this` 查看）中有这么一句名言：“There should be one-- and preferably only one --obvious way to do it.”，翻译成中文是“做一件事应该有而且最好只有一种确切的做法”，这句话传达的思想在 PEP 8 中也是无处不在的。

1. 采用内联形式的否定词，而不要把否定词放在整个表达式的前面。例如 `if a is not b` 就比 `if not a is b` 更容易让人理解。
2. 不要用检查长度的方式来判断字符串、列表等是否为 `None` 或者没有元素，应该用 `if not x` 这样的写法来检查它。
3. 就算 `if` 分支、`for` 循环、`except` 异常捕获等中只有一行代码，也不要将代码和 `if`、`for`、`except` 等写在一起，分开写才会让代码更清晰。
4. `import` 语句总是放在文件开头的地方。
5. 引入模块的时候，`from math import sqrt` 比 `import math` 更好。
6. 如果有多个 `import` 语句，应该将其分为三部分，从上到下分别是 Python **标准模块**、**第三方模块**和**自定义模块**，每个部分内部应该按照模块名称的**字母表顺序**来排列。

1，Python 编程规范

> 编码

```
1
2
3
4
5
```

所有的 Python 脚本文件都应在文件头标上

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

用于设置编辑器，默认保存为 utf-8 格式。

> 注释

```
1
2
3
4
```

业界普遍认同 Python 的注释分为两种，

一种是由 # 开头的“真正的”注释，例如，用于表明为何选择当前实现以及这种实现的原理和难点
另一种是 docstrings，例如，用于表明如何使用这个包、模块、类、函数（方法），甚至包括使用示例和单元测试坚持适当注释原则。

对
难点的代码必须注释。但与注释不同，建议对每一个包、模块、类、函数（方法）写

docstrings，除非代码一目了然，非常简单。

> 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。如果使用 tab 缩进，设定 tab 为 4 个空格。

> 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。

```
1
```

1）在二元算术、逻辑运算符前后加空格：如

```
2
3   a = b + c;
4   2） 在一元前缀运算符后不加空格，如
5
6   if  !flg: pass;
7   3） “:”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：
8
9   dict 对象的定义
10
11  d = {'key'  :    'value'}
12  4） 括号（含圆括号、方括号和花括号）前后不加空格，如：
13
14  do_something(arg1, arg2)
15  而不是 do_something( arg1, arg2 )<br>
16  5） 不要在逗号、分号、冒号前面加空格，但应该在它们后面加（除了在行尾）
17
18  6） 不要用空格来垂直对齐多行间的标记，因为这会成为维护的负担（适用于:，#，=等）
```

> 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   1） 在类、函数的定义间加空行；
2   2） 在 import 不同种类的模块间加空行；
3   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；
```

> 断行

尽管现在的宽屏显示器已经可以单屏显示超过 256 列字符，但本规范仍然坚持行的最大长度不得超过 80 个字符的标准。折叠长行的方法有以下几种方法:

1) 为长变量名换一个短名，如:

```
this.is.a.very.long.variable_name =  this.is.another.long.variable_name
```

应改为:

```
variable_name1 =  this.is.a.very.long.variable_name
variable_name2 =  this.is.another.variable_name
variable_name1 = variable_name2s
```

2) Python 会将圆括号、中括号和花括号中的行隐式的连接起来，你可以利用这个特点。如需要，你可以在表达式外围增加一对额外的圆括号

3) 在长行加入续行符强行断行，断行的位置应在操作符前，且换行后多一个缩进，以使维护人员看代码的时候看到代码行首即可判定这里存在换行，如:

```
if  color == WHITE or color == BLACK \
        or color == BLUE: # 注意 or 操作符在新行的行首而不是旧行的行尾
do_something(color);
```

> 字符串

1. 避免在循环中用+和+=操作符来累加字符串。由于字符串是不可变的，这样做会创建不必要的临时对象，并且导致二次方而不是线性的运行时间。
表，然后在循环结束后用 .join 连接列表。（也可以将每个子串写入一个 cStringIO.StringIO 缓存中

1. 为多行字符串使用三重双引号而非三重单引号。不过要注意，通常用隐式行连接更清晰，因为多行字符串与程序其他部分的缩进方式不一致。

>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。

>> 常量

常量名所有字母大写，由下划线连接各个单词，如

```
WHITE = 0XFFFFFF
THIS_IS_A_CONSTANT = 1
```

>> 变量

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
```

私有类成员使用单一下划线前缀标识，多定义公开成员，少定义私有成员。

变量名不应带有类型信息，因为 Python 是动态类型语言。如 iValue、names_list、dict_obj 等都是不好的命名。

>> 函数

1 函数名的命名规则与变量名相同。

>> 类

对类名使用大写字母开头的单词（如 CapWords，即 Pascal 风格），不使用下划线连接单词。如：

```
class  ThisIsAClass(object):pass
```

>> 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
```

2	_internal_module.py

>> 包

1	包的命名规范与模块相同

>> 缩写

命名应当尽量使用全拼写的单词，缩写的情况有如下两种：

1） 常用的缩写，如 XML、ID 等，在命名时也应只大写首字母，如

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

2） 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：

```
1  function 缩写为 fn
2
3  text 缩写为 txt
4
5  object  缩写为 obj
6
7  count 缩写为 cnt
8
9  number 缩写为 num，等。
```

>> 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

```
1  class  Base(object):
2          def __init__(self, id, parent =None):
```

```
3                   self.__id__ = id
4                   self.__parent__ = parent
5           def __message__(self, msgid):
6                   # …略
```

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

>> 导入格式

1.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；这几种模块用空行分隔开来。

2.每个 import 应该独占一行。

3.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。

> 赋值

对于赋值语言，主要是不要做无谓的对齐，如：

```
1   a                       = 1                                                 # 这是一个行注释
2   variable = 2                                                        # 另一个行注释
3   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
```

> 语句

通常每个语句应该独占一行。不过， 如果测试结果与测试语句在一行放得下， 你也可以将它们放在同一行。如果是 if 语句， 只有在没有 else 时才能这样做。特别地，绝不要对 try/except 这样做，因为 try 和 except 不能放在同一行。

2，参考资料

python 面试题总结（1）--语言特性

1. 谈谈对 Python 和其他语言的区别

答:

Python 是一门**强类型**的**可移植、可扩展、可嵌入**的**解释型编程语言**，属于**动态语言**；其语法简洁优美、功能强大无比、应用领域非常广泛且具有强大完备的第三方库。

（注：语言有**无类型，弱类型和强类型三种**。强类型指的是程序中表达的任何对象所从属的类型都必须能在编译时刻确定。常见的强类型语言有 C++、Java、Apex 和 Python 等。）

拿 C 语言和 Python 比： Python 的第三方类库比较齐全并且使用简洁,很少代码就能实现一些功能，如果用 C 去实现相同的功能可能就比较复杂。

但是对于速度来说 Python 的运行速度相较于 C 就比较慢了。所以有利的同时也有弊端，毕竟我们的学习成本降低了。

2. 简述解释型和编译型编程语言

答:

编译型语言是指使用专门的编译器，针对特定平台（操作系统）将某种高级语言源代码，一次性“翻译”成可被该平台硬件执行的机器语言（包括机器指令和操作数），并包装成该平台所能识别的可执行程序的格式，这个转换过程称为编译（Compile）。

编译生成的可执行程序可以脱离开发环境，在特定的平台上独立运行

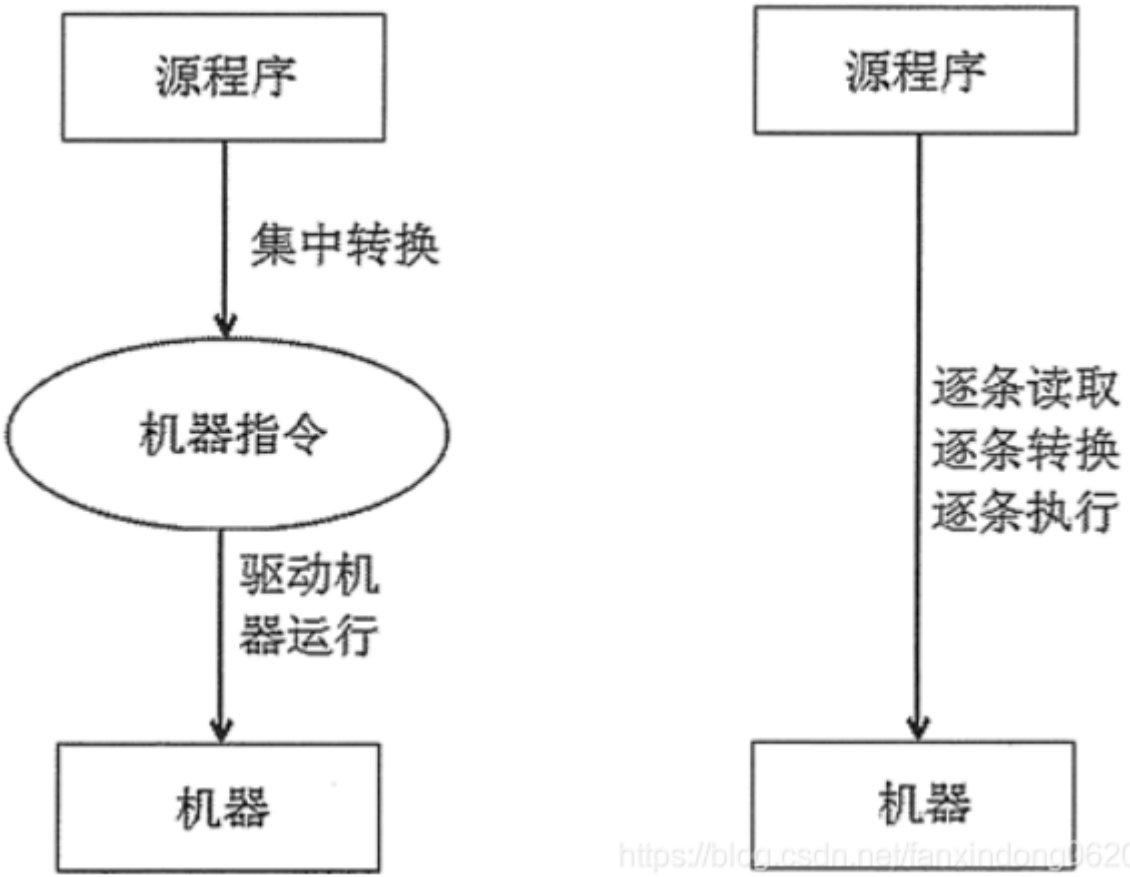
解释型语言是指使用专门的解释器，将源程序逐行解释成特定平台的机器代码并立即执行的语言。

解释型语言通常不会进行整体性的编译和链接处理，解释型语言相当于把编译型语言中的编译和解释过程混合到一起同时完成。

可以这样认为，每次执行解释型语言的程序都需要进行一次编译，因此解释型语言的程序运行效率通常较低，而且不能脱离解释器独立运行。但解释型语言有一个优势，就是跨平台比较容易，只需提供特定平台的解释器即可，每个特定平台上的解释器都负责将源程序解释成特定平台的机器指令。

也就是说，解释型语言可以方便地实现源程序级的移植，但这是以牺牲程序执行效率为代价的。

编译型语言和解释型语言的对比如图所示。



不难理解，编译型语言和解释型语言的区别在于，编译是对高级语言程序进行一次性翻译，这样的好处是，一旦源程序被彻底翻译，它就可以重复运行，且今后都不再需要编译器和源代码；而如果使用解释器，则高级语言程序每次运行，都需要借助源程序和解释器，其最大的好处就是，程序有很好的可移植性。

3. Python 的解释器种类以及相关特点?

答:

CPython–c 语言开发的，使用最广的解释器

IPython–基于 cPython 之上的一个交互式计时器，交互方式增强功能和 cPython 一样

PyPy–目标是执行效率，采用 JIT 技术。对 Python 代码进行动态编译，提高执行效率

JPython–运行在 Java 上的解释器，直接把 Python 代码编译成 Java 字节码执行

IronPython–运行在微软 .NET 平台上的解释器，把 Python 编译成 . NET 的字节码。

4. Python3 和 Python2 的区别?

答： 这里例举 5 条
print 在 Python3 中是函数，必须加括号；Python2 中 print 为 class。
Python2 中使用 xrange，Python3 使用 range。
Python2 中默认的字符串类型是 ASCII，Python3 中默认的字符串类型是 Unicode。
Python2 中/的结果是整型，Python3 中是浮点类型。
Python2 中声明元类：*metaclass* = MetaClass, Python3 中声明元类：class newclass(metaclass=MetaClass)：pass。

5. Python3 和 Python2 中 int 和 long 区别?

答：
Python2 有 int 和 long 类型。int 类型最大值不能超过 sys.maxint，而且这个最大值是平台相关的。可以通过在数字的末尾附上一个 L 来定义长整型，显然，它比 int 类型表示的数字范围更大。
在 Python3 里，只有一种整数类型 int，大多数情况下，表示为长整型，没有 python2 中的 Long。和 Python 2 中的长整型类似。

6. xrange 和 range 的区别?

答：
首先得说明一下，只有在 python2 中才有 xrange 和 range，python3 中没有 xrange，并且 python3 中的 range 和 python2 中的 range 有 本质的区别。所以这儿说的 range 和 xrange 的区别是只针对 python2 的。
在 py2 中，range 得到的是一个列表,即

```
x = range(0, 5)
print(type(x))  # 打印 x 的类型，结果是 list
print(x) # 结果是[0,1,2,3,4]
```

xrange 得到的是一个生成器对象， 即

```
x = xrange(0, 5)
print(type(x))  # 输出类型，结果为一个生成对象
print(x)  # 输出 x， 结果为 xrange(0,5)
```

那么，python3 中为什么没有了 range 了呢（额，这个怎么描述呢，是有 range，但是这个 range 其实是 py2 中的 xrange，而不是 range），因为使用生成器可以节约内存。比如现在有个代码是 for i in range(0, 10000)，如果还是使用 py2 中的 range 的话，那么就会得到一个 0 到 9999 的一个列表，这个将会占用很大的空间，但是使用生成器的话，不需要一上来就开辟一块很大的内存空间，会节省很大的资源。

python 面试题总结（2）--编码规范

1. 什么是 PEP8?

答：PEP8 --《Python Enhancement Proposal #8》（8 号 Python 增强提案），他针对的 Python 代码格式而编订的风格指南。

2. 了解 Python 之禅么?

答：通过 import this 语句可以获取其具体的内容。它告诉大家如何写出高效整洁的代码。例如：
避复就简
Simple is better than complex.
避隐就显
Explicit is better than implicit.
能不嵌套就不嵌套
Flat is better than nested.

3. 了解 DocStrings 么?

1）DocStrings 文档字符串是一个重要工具，主要是解释代码作用的，帮助你的程序文档更加简单易懂。
2）我们可以在函数体的第一行使用一对三个单引号 ‘’‘ 或者一对三个双引号 “”” 来定义文档字符串。
3）可以使用 doc（注意双下划线）调用函数中的文档字符串属性。

```
def function():
       ''' say something here!
       '''
       pass
print (function.__doc__) # 调用 doc
```

输出结果为：

```
 say something here!
```

4. 了解类型注解么?

Python 是动态语言，变量随时可以被赋值，且能赋值为不同的类型
Python 不是静态编译型语言，变量类型是在运行器决定的
动态语言很灵活，但是这种特性也是弊端

```
       def add(x, y):
           return x + y
       print(add(4, 5))
       print(add('hello', 'world'))
       add(4, 'hello')
```

难发现：由于不做任何类型检查，直到运行期问题才显现出来，或者线上运行时才能暴露出问题
难使用：函数的使用者看到函数的时候，并不知道你的函数的设计，并不知道应该传入什么类型的数据

函数注解

- Python 3.5 引入

- 对函数的参数进行类型注解
- 对函数的返回值进行类型注解
- 只对函数参数做一个辅助的说明，并不对函数参数进行类型检查
- 提供给第三方工具，做代码分析，发现隐藏的 bug
- 函数注解的信息，保存在__annotations__属性中

```
def add(x:int , y:int) -> int:
    '''
    :param x:
    :param y:
    :return:
    '''
    return x + y
print(help(add))
print(add.__annotations__)
```

运行结果

```
add(x:int, y:int) -> int
    :param x:
    :param y:
    :return:
None
{'x': <class 'int'>, 'y': <class 'int'>, 'return': <class 'int'>}
```

变量注解

Python 3.6 引入。它也只是一种对变量的说明，非强制

```
i: int = 3
```

5. 例举你知道 Python 对象的命名规范，例如方法或者类等

类：总是使用首字母大写单词串，如 MyClass。
内部类：可以使用额外的前导下划线。
变量：小写，由下划线连接各个单词。方法名类似
常量：常量名所有字母大写 等
函数&方法：函数名应该为小写，可以用下划线风格单词以增加可读性，如：myfunction， my_example_function。（混合大小写仅被允许用于这种风格已经占据优势的时候，以便保持向后兼容。）

函数和方法的参数：总使用“self”作为实例方法的第一个参数。总使用“cls”作为类方法的第一个参数。
如果一个函数的参数名称和保留的关键字冲突，通常使用一个后缀下划线好于使用缩写或奇怪的拼写。
详情见：python 命名规范

6. Python 中的注释有几种?

答：总体来说分为两种，单行注释和多行注释。
单行注释在行首是 #。
多行注释可以使用三个单引号或三个双引号，包括要注释的内容。

7. 如何优雅的给一个函数加注释?

答：可以使用 docstring 配合类型注解

8. 如何给变量加注释?

答：可以通过变量名：类型的方式如下

```
a: str = "this is string type"
```

9. Python 代码缩进中是否支持 Tab 键和空格混用。

答：不允许 tab 键和空格键混用，这种现象在使用 sublime 的时候尤为明显。一般推荐使用 4 个空格替代 tab 键。

10. 是否可以在一句 import 中导入多个库?

答：可以是可以，但是不推荐。因为一次导入多个模块可读性不是很好，所以一行导入一个模块会比较好。同样的尽量少用 from modulename import *，因为判断某个函数或者属性的来源有些困难，不方便调试，可读性也降低了。

11. 在给 Py 文件命名的时候需要注意什么?

给文件命名的时候不要和标准库中的一些模块重复，比如 abc。
另外要名字要有意义，不建议数字开头或者中文命名。

12. 例举几个规范 Python 代码风格的工具

自动检测工具 Pylint：一个检查违反 PEP8 规范和常见错误的库，它会自动查找不符合代码风格标准和有潜在问题的代码，并在控制台输出代码中违反规范和出现问题的相关信息。
自动优化工具 Black：在众多代码格式化工具中，Black 算是比较新的一个，它最大的特点是可配置项比较少，个人认为这对于新手来说是件好事，因为我们不必过多考虑如何设置 Black，让 Black 自己做决定就好。
在使用方面，black 默认读取指定 python 文件并对其进行代码规范格式化，然后输出到原文件。

13、Python2.7 与 Python3 之间的主要区别

1. 使用__future__模块

Python 3.X 引入了一些与 Python 2 不兼容的关键字和特性。在 Python 2 中，可以通过内置的__future__模块导入这些新内容。如果你希望在 Python 2 中写的代码也可以在 Python 3.X 中运行，那么建议使用__fufure__模块。

2. print 函数

虽然 print 语法是 Python 3 中一个很小的改动，但是依然值得提一下：Python 2 中的 print 语句被 Python 3 中的 print()函数取代，这意味着在 Python 3 中必须用括号将需要输出的对象括起来。在 Python 2 中使用额外的括号也可以，但是如果要在 Python 3 中以 Python 2 的形式不带括号调用 print 函数，就会触发 SyntaxError(语法错误）。

3. 整数除法

由于人们常常会忽视 Python 3 在整,数除法上的改动（写错了也不会触发 SyntaxError),因此在移植代码或在 Python 2 中执行 Python 3 的代码时需 4 要特别注意这个改动。

4. Unicode

Python 2 有基于 ASCII 的 str()类型，可通过单独的 unicode()的函数转成 unicode 类型，但没有 byte 类型。在 Python 3 中有了 Unicode(UTF-8)
字符串和两个字节类（bytes 和 bytearrays）。

5. xrange

在 Python 2.x 中，经常会用 xrang()创建一个可迭代对象，通常出现在“for”循环或“列表/集合/字典推导式”中。（在 Python 3 中使用 xrang()会触。NameError)。

6. 触发异常

Python 2 支持新旧两种异常触发语法，而 Python 3 只支持带括号的语法（不然会触发 SyntaxError）。

7. 处理异常

Python 3 中的异常处理发生了一点变化。在 Python 3 中必须使用 as 关键字，Python 2 中不需要。

8. next()函数和.net()方法

由于会经常用到 next() (.next())函数（方法），因此要提到另一个语法改动（实现方面也做了改动）：在 Python 2 中，函数形式和方法形式都可以使用；在 Python 3 中 ，只能使用 next()函数（试图调用.next()方法会触发 AttributeError) 。

9. for 循环变量与全局命名空间泄露

在 Python 3 中，for 循环中的变量不再会泄露到全局命名空间中了。

10. 比较无序类型

Python 3 中另一个优秀的改动是，如果我们试图比较无序类型，就会触发一个 TypeError。

11. 使用 input()解析输入内容

Python 3 改进了 input()函数，这样该函数就会总是将用户的输入存储为 str 对象。在 Python 2 中，为了避免读取非字符串类型会发生的一些危险行为，不得不使用 raw_input()代替 input()。

1>python3 里面已经把 raw_input()给去掉了
- 事实上是这样的：在 Python 3 内，将 raw_input() 重命名为 input()，这样一来，无须导入也能从标准输入获得数据了。如果您需要保留版本 2.x 的 input() 功能，可以使用 eval(input())，效果基本相同。

2>Python 版本 2.x 中，raw_input() 会从标准输入（sys.stdin）读取一个输入并返回一个字符串，且尾部的换行符从末尾移除
注意：使用 python2.x 版本进行相应的练习
raw_input()随便输都是字符串，而 input()必须按照 Python 的规则来~
1.raw_input()

```
name=raw_input('输入姓名：')
age=raw_input('输入年龄')
```

我们输入汉字的姓名和数字的年龄

```
输入姓名：乖乖
输入年龄：5
乖乖 5

***Repl Closed***
```

直接输，效果杠杠的~但是要注意哦，你的年龄的格式是 string

2.input()

```
name=input('输入姓名：')
age=input('输入年龄：')
```

我们还是输入汉字的姓名和数字的年龄

```
输入姓名：'乖乖
输入年龄：5
乖乖 5

***Repl Closed***
```

效果也是杠杠的~但是需要注意 input()输入严格按照 Python 的语法，是字符就自觉的加 ' ' ，数字就是数字
输入的类型为字符的时候可以用 raw_input()，当然不怕麻烦也可以用 input()手动加引号
int 类型的时候最好用 input()
总之，非常的灵活！！！

12. 返回可迭代对象，而不是列表

某些函数和方法在 Python 3 中返回的是可迭代对象，而不像在 Python 2 中返回列表。对象只遍历一次会节省很多内存，如果通过生成器多次迭代这些对象，效率就不高了。此时如果需要列表对象，可以通过 Python 3 的 list()函数简单地将可迭代对象转成列表。

```
Ideal are like the stars --- we never reach them ,but like mariners , we chart our course by them
```