Python核心编程第三章笔记

3.1.2 继续( \ )

Python 语句，一般使用换行分隔，也就是说一行一个语句。（换行就相当于C++里面的分号）一行过长的语句可以使用反斜杠( \ ) 分解成几行。（相当于matlab里面的…）

例外情况一个语句不使用反斜线也可以跨行：小括号、中括号、花括号，三引号。

3.1.3 多个语句构成代码组(:)：

冒号

If while def class 等

标题行+:+子句

3.1.4 代码组由不同的缩进分隔

核心风格：缩进四个空格宽度，避免使用制表符

3.1.5 同一行书写多个语句(;)

3.1.6 模块

每一个Python 脚本文件都可以被当成是一个模块。模块以磁盘文件的形式存在。

3.2 变量赋值

3.2.1一对一赋值

注意，赋值并不是直接将一个值赋给一个变量， 尽管你可能根据其它语言编程经验认为应该如此。在Python 语言中，对象是通过引用传递的。在赋值时，不管这个对象是新创建的，还是一个已经存在的，都是将该对象的引用（并不是值）赋值给变量。

3.2.2链式赋值

>>> y = x = x + 1

3.2.3“多元”赋值

>>> x, y, z = 1, 2, 'a string'

应加括号

>>> (x, y, z) = (1, 2, 'a string')

使用多元赋值交换两个变量的值

>>> x, y = y, x

3.3 标识符

核心风格：避免用下划线作为变量名的开始

因为下划线对解释器有特殊的意义，而且是内建标识符所使用的符号，我们建议程序员避免用下划线作为变量名的开始。一般来讲，变量名\_xxx 被看作是“私有的”，在模块或类外不可以使用。当变量是私有的时候，用\_xxx 来表示变量是很好的习惯。因为变量名\_\_xxx\_\_对Python 来说有特殊含义，对于普通的变量应当避免这种命名风格。

􀁺 \_xxx 不用'from module import \*'导入

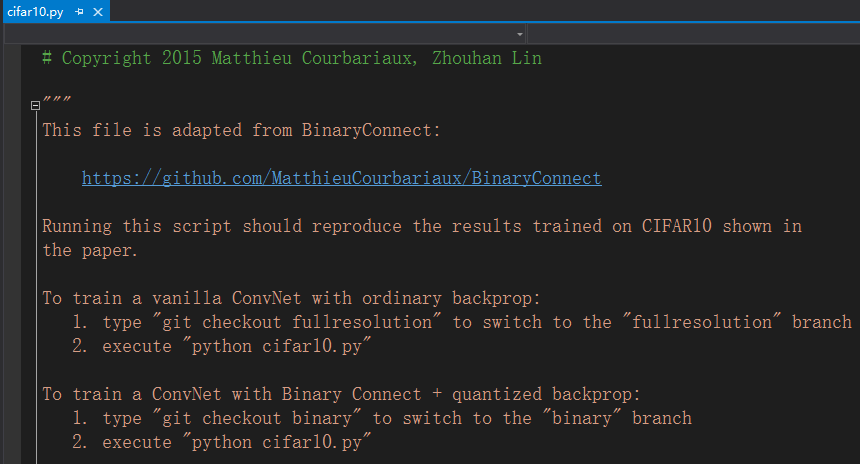
􀁺 \_\_xxx\_\_系统定义名字

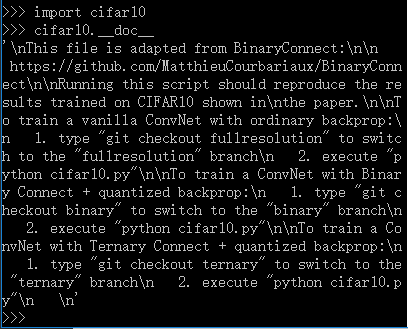
􀁺 \_\_xxx 类中的私有变量名

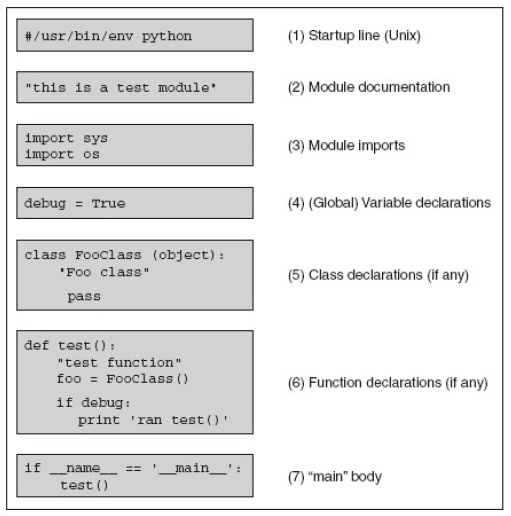
3.4 基本风格指南

文档

Python 还提供了一个机制，可以通过\_\_doc\_\_特别变量，动态获得文档字串。在模块，类声明，或函数声明中第一个没有赋值的字符串可以用属性obj.\_\_doc\_\_来进行访问，其中obj是一个模块，类，或函数的名字。这在运行时刻也可以运行。







(6)函数定义语句

此处定义的函数可以通过module.function()在外部被访问到.

当模块被导入时 def 语句会被执行， 函数也就都会定义好.

(7) 主程序

无论这个模块是被别的模块导入还是作为脚本直接执行，**都**会执行这部分代码。通常这里不会有太多功能性代码，而是根据执行的模式调用不同的函数。

很多项目都是一个主程序，由它导入所有需要的模块。所以请记住，绝大部分的模块创建的目的是为了被别人调用而不是作为独立执行的脚本。我们也很可能创建一个Python 库风格的模块，这种模块的创建目的就是为了被其他模块调用。总之，只有一个模块，也就是包含主程序的模块会被直接执行，或由用户通过**命令行执行**，或作为**批处理执行**， 或由**Unix cron 任务定时执行**，或通过**Web 服务器调用**，或**通过GUI 执行**。

时刻记住一个事实，那就是所有的模块都有能力来执行代码。最高级别的Python 语句－－也就是说， 那些**没有缩进的代码行在模块被导入时就会执行， 不管是不是真的需要执行。**由于有这样一个“特性”，比较安全的写代码的方式就是**除了那些真正需要执行的代码以外， 几乎所有的功能代码都在函数当中。**再说一遍， 通常只有主程序模块中有大量的顶级可执行代码，所有其它被导入的模块只应该有很少的顶级执行代码，所有的功能代码都应该封装在函数或类当中。

（参阅核心笔记了解更多信息）

核心笔记：\_\_name\_\_ 指示模块应如何被加载

由于主程序代码无论模块是被导入还是被直接执行都会运行， 我们必须知道模块如何决定运行方向。一个应用程序可能需要导入另一个应用程序的一个模块，以便重用一些有用的代码（否则就只能用拷贝粘贴那种非面向对象的愚蠢手段）。这种情况下，你只想访问那些位于其它应用程序中的代码，而不是想运行那个应用程序。因此一个问题出现了，“Python 是否有一种方法能在运行时检测该模块是被导入还是被直接执行呢？” (程序“主体”：无论模块被导入还是作为脚本程序执行，都会执行“程序主体”)

\_\_name\_\_ 系统变量

如果模块是被导入， \_\_name\_\_ 的值为模块名字

如果模块是被直接执行， \_\_name\_\_ 的值为 '\_\_main\_\_'

3.4.2 在主程序中书写测试代码

用if \_\_name\_\_ == ‘main’:,只有单独作为脚本执行而不是import的时候才会执行if里面的语句。

3.5 内存管理

􀁺 变量无须事先声明

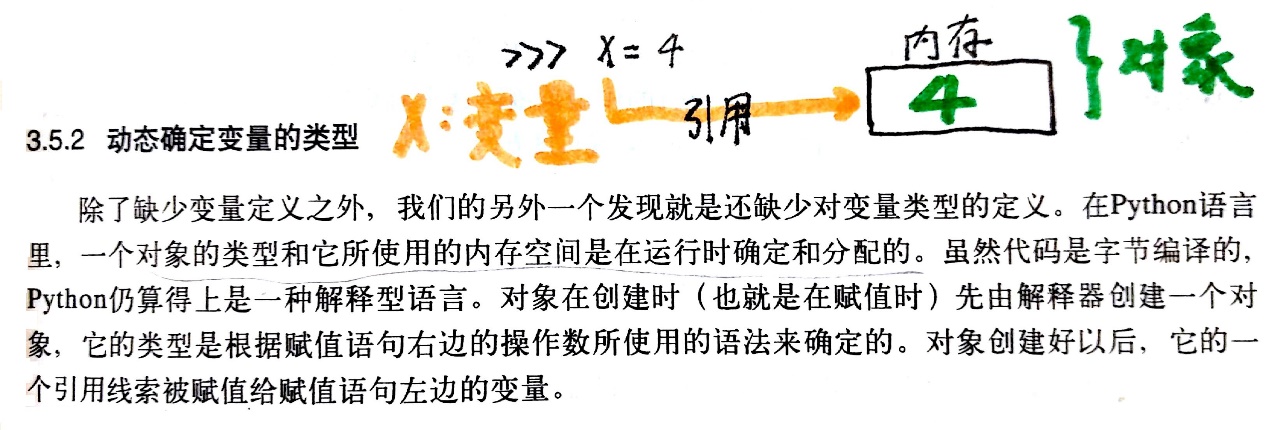
􀁺 变量无须指定类型

􀁺 程序员不用关心内存管理

􀁺 变量名会被“回收”

􀁺 del 语句能够直接释放资源

3.5.1 变量定义

变量在第一次被赋值时自动声明，只有被创建和赋值后才能被使用。

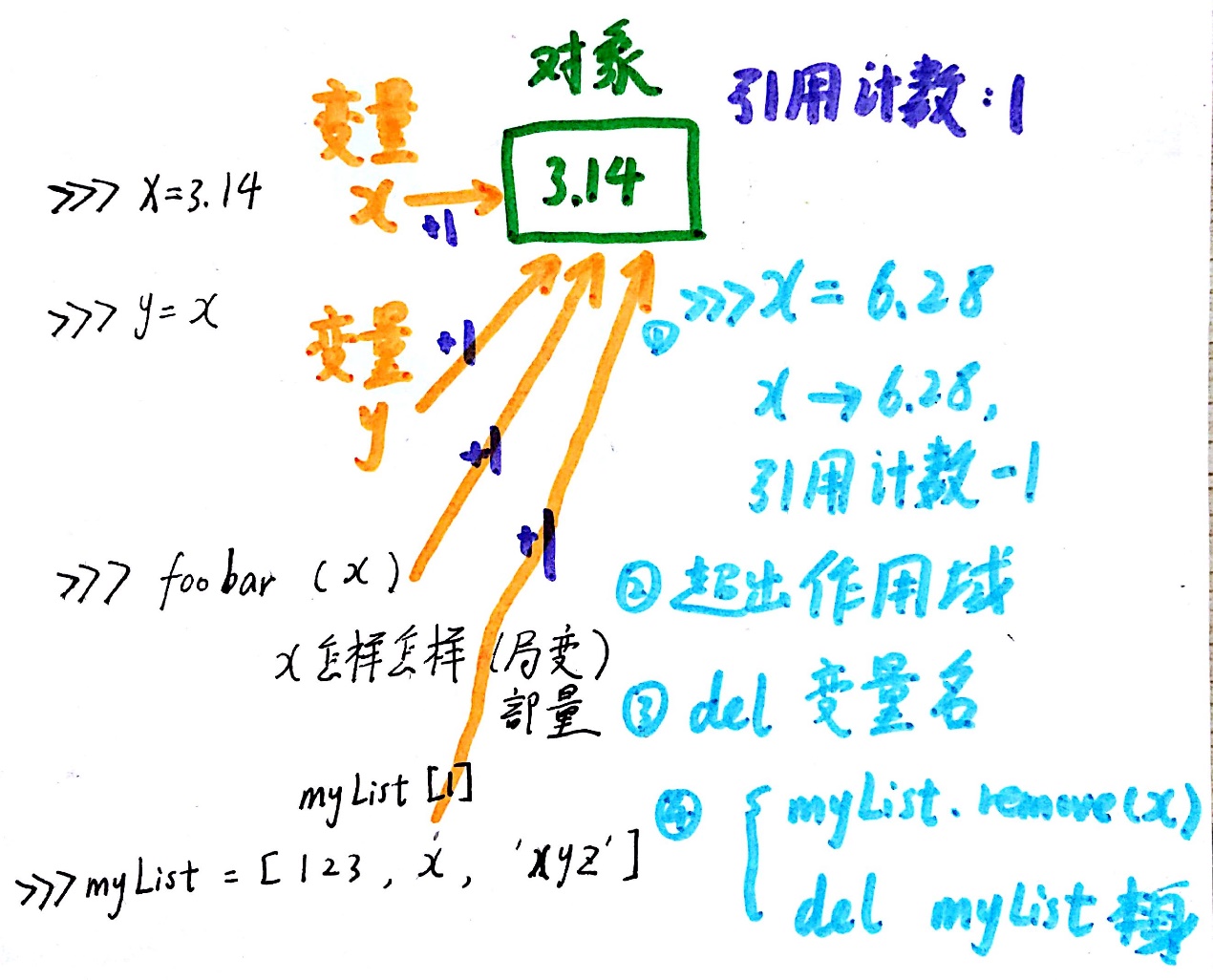
3.5.3 内存分配

Python 解释器承担了内存管理的复杂任务，我们不必考虑内存的分配，也不必考虑内存的释放。

3.5.4 引用计数

要保持追踪内存中的对象， Python 使用了引用计数这一简单技术。也就是说Python 内部记录着所有使用中的对象各有多少引用。一个内部跟踪变量，称为一个引用计数器。至于每个对象各有多少个引用， 简称引用计数。当对象被创建时， 就创建了一个引用计数， 当这个对象不再需要时， 也就是说， 这个对象的

引用计数变为0 时， 它被垃圾回收。（严格来说这不是100%正确，不过现阶段你可以就这么认为）

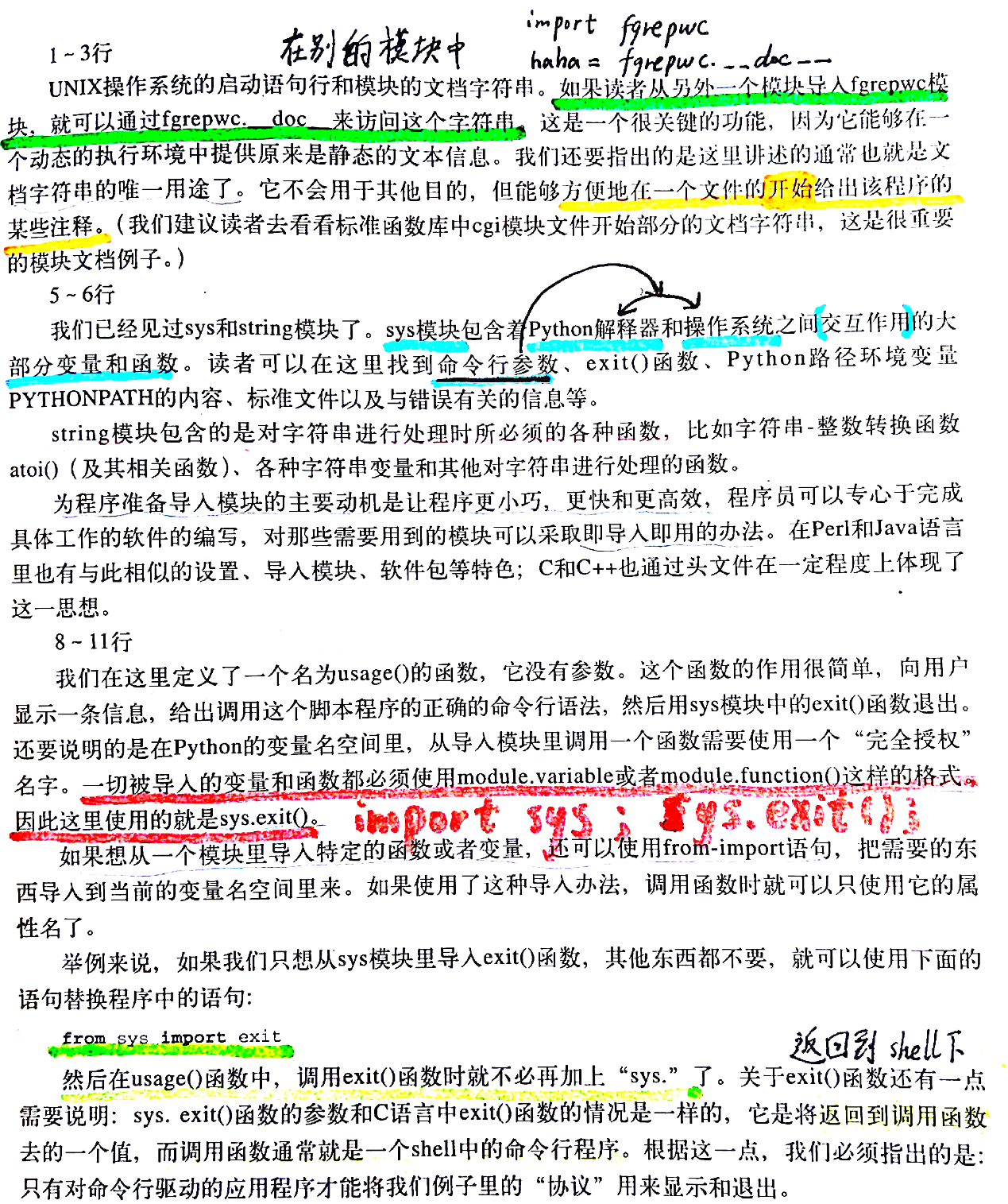


当对象被创建并（将其引用）赋值给变量时，该对象的引用计数就被设置为1。当同一个对象（的引用）又被赋值给其它变量时，或作为参数传递给函数， 方法或类实例时， 或者被赋值为一个窗口对象的成员时，该对象的一个新的引用，或者称作别名，就被创建

（则该对象的引用计数自动加1）。

对象的引用计数减为0， 这会导致该对象从此“无法访问”或“无法抵达”。 从此刻起， 该对象就成为垃圾回收机制的回收对象。 注意任何追踪或调试程序会给一个对象增加一个额外的引用， 这会推迟该对象被回收的时间。

3.6 第一个Python 程序



由下图，可以可以看出，exit函数如果参数为1，exit(1)是告诉系统“failure”

