Python 程序设计 - Python 基础

李正华

October 25, 2016

1 本章主题

- 语句和语法
- 变量赋值
- 基本代码风格（暂时不讲）
- 内存管理
- 看几个 Python 程序

2 语句和语法

2.1 注释

2.2 继续（\）

通常一个 Python 语句对应一行。

用 \ 可以将一个 Python 语句分为多行写。（如何将多个 Python 语句写到一行？）

两种特殊情况：

- 三引号内的字符串可以跨行书写
- 给多个变量同时赋值（元组形式）

```
str = ''' a
bc de
x
'''
x, y = (1,
2)
```

2.3 多个语句构成代码组 (:)

缩进相同的一组语句构成一个代码块，称为一个代码组。

if、while、def 和 class 这样的复合语句

2.4 代码组由不同的缩进分隔

代码的层次关系通过同样深度的缩进体现。

为了避免不同文本编辑器中制表符代表的空白宽度不一致，推荐用 4 个空白符作为一个缩进。

2.5 同一行书写多个语句（;）

虽然允许，但是不提倡。

3 变量赋值

3.1 赋值操作符

等号（=）是主要的赋值操作符

不是直接将一个值赋给一个变量,而是让变量指向将对象。(此处需要慢慢理解,体会)

```
x = 1
y = (x = x + 1)
y = x = x + 1
```

3.2 增量赋值

```
x += 1
-= *= /= %= **=
<<= >>= &= ^= |=
```

增量赋值（x += y）时，Python 的工作原理：

- 如果 x 是可变对象（思考什么是可变对象？什么情况下是可变对象?），那么 x 所指的对象就地修改。
- 如果 x 是不可变对象，那么新创建一个对象，让其值为 x+y，然后让 x 指向这个新的对象。

Python 不支持 x++ 或 ++x 语法。

```
x = '123'
print(x, id(x))
x += '456'
print(x, id(x))

x = [1, 2, 3]
print(x, id(x))
x += [4, 5, 6]
print(x, id(x))
```

3.3 多重赋值

```
x = y = z = 1
z = 3
```

思考其中涉及的对象、引用的概念。

3.4 “多元”赋值

```
x, y, z = 1, 1, 'a string' #
(x, y, z) = (1, 1, 'a string') # 建议这样写
```

两个变量交换值:

```
(x, y) = (1, 2)
(x, y) = (y, x)
```

4 标识符

标识符是编程语言中允许作为名字（变量名、函数名、类名等）的有效字符串集合。合法的名字必须满足如下要求:

- 大小写字母或下划线开始
- 其他可以为字母、数字和下划线
- 注意: 名字是大小写敏感的

有一部分是编程语言保留的关键字，不允许用于其他用途。

还有称为“内建”(built-in) 的标识符集合，虽然不是保留的关键字，但是不推荐使用。

4.1 保留的关键字

```
import keyword
print(keyword.kwlist)
```

and, as, assert, break
class, continue, def, del
elif, else, except, exec
finally, for, from, global
if, import, in, is
lambda, not, or, pass
print, raise, return, try
while, with, yield, None
True, False （这两个其实是两个变量，对应整数值分别为 1 和 0，对应字符串输出是’True’ 和’False’

4.2 内建（暂时不讲）

4.3 专用下划线标识符（暂时不讲）

5 基本风格指南（暂时不讲）

6 内存管理

变量无需提前声明，赋值时自动声明。

动态类型

搞清楚对象（object）、引用（reference）= 别名（alias）的概念。
内存管理包括两方面：分配和释放

写程序时，创建一个对象时，系统会为对象分配内存，使用完对象之后，需要释放内存。[1]

Python 解释器承担了内存管理的复杂任务（有好处、也有缺点）。

引用计数机制：Python 解释器会记录每一个对象目前有多少个引用。

创建对象时，引用为 1；当引用为 0 时，就会被垃圾回收。（暂时可以这么简单理解）

增加引用计数的情况：

- 对象被创建：x = 3.14
- 别的别名被创建：y = x
- 被作为参数传递给函数（新的本地引用）：foobar(x)
- 成为容器对象的一个元素：myList = [123, x, 'xyz']

减少引用计数的情况：

- 一个本地引用离开了作用范围，如 foobar 函数结束时，参数 x 就离开了的作用范围（scope）。（又翻译为作用域，需要仔细思考什么事作用域，一个变量的作用域是什么？结合代码块的概念思考，可以自己写一些简单的程序来测试。）
- 对象的别名被 del 语句显式销毁（explicitly）: del x
- 对象的别名被赋值给其他对象：x = 123
- 对象被从一个窗口对象中移除：myList.remove(x) # help(list)
- 窗口对象本身被 del 语句销毁（或离开作用域）：del myList

思考下面的代码会输出什么：

```
x = 123
a = [1, 2, x]
print(a)
x = 456
print(a)
b = a
a[1] = 'abc'
print(b)
c = a
```

del 语句（statement）：删除一个对象的引用

垃圾回收

[1] C 语言中的指针：自己控制 malloc 和 free。C++ 中为 new 和 delete

7 深拷贝（构建新的对象）与浅拷贝（拷贝引用）

指针 = 引用 = 别名

对象是真实存在于内存中。变量只是一个名字（别名、引用、指针），指向对象（可以认为变量不占用任何内存）。

自己设计一些例子，多尝试。

变量可以指向任何对象，运行时根据变量指向的对象，确定变量的类型（动态类型）。

赋值：浅拷贝

切片操作：生成一个新的 list 对象，但是内部元素仍然浅拷贝（只拷贝引用）

list 构造函数（工厂方法）：生成一个新的 list 对象，但是内部元素仍然浅拷贝（只拷贝引用）

copy.copy：浅拷贝，只拷贝父对象，不会拷贝对象的内部的子对象。

copy.deepcopy，深拷贝，拷贝对象及其子对象

http://blog.csdn.net/vicken520/article/details/8227524

http://python.jobbole.com/82294/

http://www.01happy.com/python-shallow-copy-and-deep-copy/

http://www.cnblogs.com/BeginMan/p/3197649.html