基础语法文档(二)

Author: Limzh

经过第一天的体验，现在我们进入了学习数据类型、变量作用域和类的阶段。变量是你能操作的基本单元，他们有不同的类型。学习他们很好理解，但为什么我们要绕过函数，直接学习类的概念呢？这是因为Python万物都是对象！每一个你能想到的东西都是一个类的实例！`层次感`就是由`类`这个概念延伸出来的。准备好了？那我们现在开始吧！

学习之前，稍作复习，我们过去学习的内容是：`规范性`和`快捷性`，注释和缩进的语法。

今天我们要掌握的是：

- [x] `类`概念的整体把握并通过实例来体会`面向对象`（`层次感`）的优点！
- [ ] 函数（突然意识到，要引入类的概念就不得不引入函数QAQ）
- [x] 字符串、整数、浮点数的类型以及相应的操作方法，能够利用学到的方法解决一些简单问题。
- [x] 掌握`作用域`的概念，并结合`缩进`知识理解为什么我们要有`缩进`或者`花括号（C语言）`来限定作用域.

Section 1. 变量的引用机制+简单数据类型

1.`input()`与`print()`

介绍输入函数和输出函数最基本的用法:

- `dst = input(var)`: 当程序运行到该函数的时候，输出`var`的内容，`挂起(hang over)`等待键盘输入并获取该输入，以字符串的类型返回到一个变量中.
- `print(var)`: 输出`var`中的内容并另起一行.

字符串的基本方法

2. 字符串

顾名思义，字符串是一些字符连起来组成的字符值。它的语法很简单，使用双引号：`"  ...  "`或者单引号:`'  ...  '`来创建一个字符串值。比如：

```
x = "This is a string variable."
y = 'This is an another string variable.'
```

3. 整数和浮点数

在Python中，可以对`整数`执行运算符操作，包括加、减、乘、除操作。
比如:

```
x = 10 + 20
y = 10 * 20
z = 10 - 20
k = 10 / 20 # 带小数，结果是0.5
l = 10 // 20 # 不带小数，不进行四舍五入，直接抹去小数位,结果是 0
```

浮点数亦然.

```
x = 10.11
y = 20.12 # 像这样的形式就是浮点数（floating number）
```

4. 什么是变量!

这涉及一个相当重点的Python思想，与C语言完全不同。

首先，请看下面一段样例代码，并推测输出结果：

```
# Python version
x = [0,1,2]
y = x

x[0] = 1
print(y)
```

C ++版本：

```
// C++ version
int x[3] = {0,1,2};
int y[3] = {0,1,2};

x[0] = 1
printf("%d", y)
```

诶？在Python中，y的值是`[1, 1, 2]`！但是C++版本是`[0, 1, 2]`!这是为什么呢？？？

如果我们稍微改变一下代码，会有改变吗?

```
# Python version
x = [0, 1, 2]
y = [0, 1, 2]

x[0] = 1
print(y)
```

y的值还是`[0, 1, 2]`,并没有发生改变。

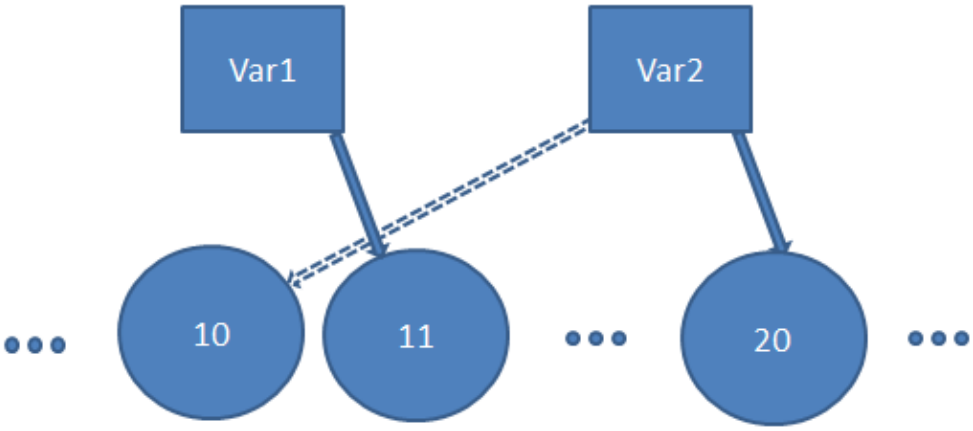
> *每一个语法的设计都不是冗余，背后隐藏着语言对内存的管理机制。* ----- Guido van Rossum

请接下来思考这一个问题，为什么C语言中的变量需要声明类型而Python却不需要呢？这会不会与这个现象有关呢?

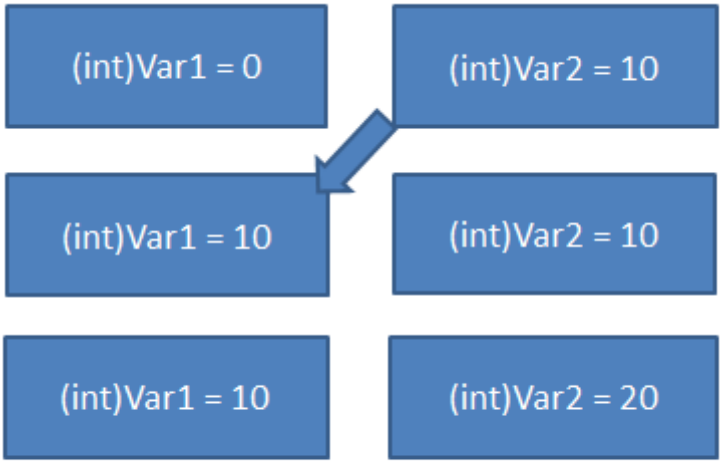
下面给出我对Python变量机制的理解和思考：

- `引用机制`

  Python中的变量实际上一个个实际数据类型`对象`的`引用`(标签)，可以理解为对一个实体的标签，而在不同变量之间的拷贝复制（如x = y），实际上是表示：x是y的所指向实体的引用。



Python变量机制



C 语言变量机制

- `变量缓存机制`

  在Python中，Python会有一个缓存对象的机制，以便重复使用。当我们创建多个等于1的引用时，实际上是让所有这些引用指向同一个对象，以达到节省资源的目的 。（这一点看不懂没关系，我演示给你看。）

因此，在编写Python的时候要明白Python变量的机制是和C++不同的。在C++中，我每一个变量名都代表着独立的一块内存，变量之间的赋值是对`值的拷贝`，因此创建变量的时候要声明类型；但在Python中，变量名仅仅代表着对所指对象的`标签引用`，变量之间的赋值根本上是一种`引用的传递关系`，这就导致了变量与被赋值变量之间建立了依赖关系。怀揣着这样对变量的理解，就可以少写一些Bug.

练习：

1. 请打开`day2/variables.py`，并将其代码复制到*代码可视化网站-Python版*运行，并观察其变量与实际数据之间的映射关系，理解`变量`一词在Python中的意味。并进一步的，打开`day2/variables.cpp`，将其代码复制到 *代码可视化网站-Cpp版* 运行，比较Cpp中的变量和Python中的变量的差异。

   请理解本section中最重要的一点：

   > Cpp中，变量是实际存在的内存块，其类型声明是为了分配相应的内存空间；而Python中，数据是实际存在的内存块但是变量只是一个访问的标签引用。

2. 由于Python变量引用机制的特殊性，Python编程中对变量赋值另一个变量时，常常出现变量相互依赖的情况。（比如section中的例子）。Python中，一个变量赋值给另一个变量（x = y），因其只是复制`引用关系`,而被称作`浅复制`，浅复制常常是许多Bug的来源。请考虑，如何对`列表变量`进行赋值使得两个变量不仅仅是浅复制关系，而是真正的赋值呢(`深复制`)呢?

Python包含以下方法:

序号	方法
1	list.append(obj) 在列表末尾添加新的对象
2	list.count(obj) 统计某个元素在列表中出现的次数
3	list.extend(seq) 在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
4	list.index(obj) 从列表中找出某个值第一个匹配项的索引位置
5	list.insert(index, obj) 将对象插入列表
6	list.pop([index=-1]) 移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
7	list.remove(obj) 移除列表中某个值的第一个匹配项
8	list.reverse() 反向列表中元素
9	list.sort(key=None, reverse=False) 对原列表进行排序
10	list.clear() 清空列表
11	list.copy() 复制列表

(Hint: 答案就在上图)

3. 思考题：仅仅复制引用关系被称作`浅复制`，在Python的列表等容器对象赋值的时候广泛地采取了浅复制的方法。如果对数据进行真正的赋值，那么我们称之为`深复制`。请回答以下两个问题：
    - 浅复制和深复制的优缺点?
    - Python的`变量引用机制`决定了其`垃圾回收机制`，即：对无用的内存数据进行释放的机制。如果你是程序设计者，根据`引用机制`，你将如何断定一块内存数据是否无用？（Hint: 你在第一题的可视化代码中曾观察过这种现象，可视化界面里，一块内存的消失意味着垃圾回收机制对其进行了释放。假如你已经有了部分猜想，请用可视化网站执行下列代码以验证）

```
x = [1,2,3]
y = x
z = y.copy()
z[0] = 10
x = 1
y = 1
```

4.我为你准备了一个计算器模板，打开`day2/calculator.py`,请按照注释所写要求，在不参照资料的情况下把它填补完整。

Section 2. 变量的命名空间与作用域

昨天我们学习了`缩进`的概念，缩进的本质是作用域的限定，作用域的本质是：***变量生效的空间范围***。变量还有生命周期，生命周期的本质是：***变量生效的时间范围***。 生命周期结束，变量的内存被释放。大致上，变量的作用域分为全局作用域和本地作用域两种。注意： 只有函数和类的才有低级的作用域。尽管循环和if语句也有缩进，但并没有自己的作用域。

如示例程序：

```
# Global scope
x = 10
def func1():
    # local scope
    y = 10
func1()
```

四点特性:

- 声明在全局作用域的变量的生命周期是整个程序，也即，除非触发`垃圾回收机制`，不然全局变量直到程序执行结束之后才会被释放。当程序进入低级作用域的时候，会额外开辟新的内存空间。本地作用域（低级作用域）的变量直到生命周期结束之后才会被释放。
- 高级别作用域（比如第一级，第二级）可以被低级别作用域（第n级）访问到，但高级别不可以访问低级别。这个很好理解，我们都认识习大大，但是习大大无法认识到我们。
- 访问变量的时候会首先从当前作用域开始查找变量名，如果查找不到就向上一级查找，直到找到对应变量名或者找不到返回报错。举一个例子，李明是是一位国家领导人，它处在全局作用域，离我们级别较远。咱俩聊天的时候，我突然提起李明，因为我们同级别的人没有叫这个名字的，所以你就会去想级别较远的人里面有没有叫这个名字的，发现没有，就又会去想更远级别的，直到想到最高级别。但是假如，恰好我们认识一个李明，是我的老师。那么我们谈起李明，在不加修饰和语境的情况下，都默认指的是该老师，因为他离我们级别较近。
- 可以使用`nonlocal`，`global`关键字来对变量的作用域范围加以限定。

为熟悉这四点特性，我们看几个例子

实例1：

```
# 全局作用域
x = 10
def func1():
    # 第一级作用域
    x = 11
    x = x - 2
# 调用函数
func1()
# x 是几呢?
print(x)
```

- x是10，没有改变。原因是因为`func1`是第一级作用域，里面声明的x和全局作用域的x是两个不一样的x。因此func1内部的x = 11 不会覆盖掉全局x. 同时， x = x - 2会先从当前作用域开始找x，所以，它找到的是第一级作用域的x，自然也不会对全局作用域产生影响。

实例2：

```
# 全局作用域
x = 10
def func1():
    # 第一级作用域
    y = x - 2
    # y是几呢?
    print(y)
# 调用函数
func1()
# x 是几呢?
print(x)
```

- x是10，原因没有变化。y是8，而不会报错原因是第二、三点特性。

实例3：

```
# 全局作用域
x = 10
def func1():
    # 第一级作用域
    y = x - 2
    x = 11
    # y是几呢?
    print(y)
# 调用函数
func1()
# x 是几呢?
print(x)
```

- y 会报错.这个可能有点费解，明明y = x - 2是在x = 11之前，那么它不应该还是8嘛？根据第三点特性，`y = x - 2`会先在本地作用域下寻找x，由于本地作用域确实存在x这个变量，因此它一定会引用本地的x（与执行的先后顺序无关、也与执不执行无关，我只知道本地作用域有x这个变量就可以根据第三特性来引用）。但是因为本地的x还没有被赋值（尚未执行），因此x值未知，会报错。

实例4:

```python
# 全局作用域
x = 10
def func1():
    # 第一级作用域
    global x
    y = x - 2
    x = 11
    # y是几呢?
    print(y)
# 调用函数
func1()
# x 是几呢?
print(x)
```

- y是8，x 是 11。当我第一级作用域下加了global 关键字，就意味着低级作用域中的对x操作全都对全局作用域的x生效！至于`nonlocal`，指的是该关键字限定的变量是非全局变量以外的作用域变量，依然依据从内层向外寻找的模式。

练习:

1. 将上面四份实例代码放到*代码可视化网站-Python版*运行，并观察其运行过程。
2. 打开文件`day2/scope.py`，在hint处添加合适的关键字(global/nonlocal)，使得程序可以输出一句我们很熟悉的密语。尝试用*代码可视化网站-Python版*运行，理解。

请在完成上面的小练习之后，点击图片.