Python是动态类型语言，相对而言C++、Java、C#等是静态类型语言。

Python中使用变量时，不需要声明其类型。

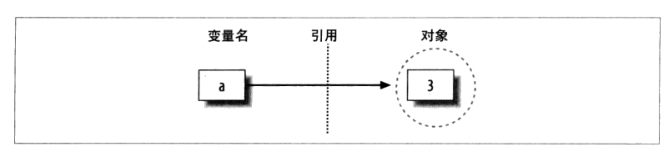
Python的底层是通过C语言来实现的。

Python中的“变量”：变量只是“标签”，它是对象的引用，变量无类型，而对象有类型。

a = 3

从概念上讲，Python在执行上述语句时，将会执行3个不同的步骤：

1. 创建一个对象来代表值3.
2. 创建一个变量a，如果它还没有创建的话。
3. 将变量a与新的对象3相关联。



如上图所示，变量a和对象3保存在内存中的不同部分。在执行a = 3之后，变量a成为对象3的一个引用。

在Python内部，变量实际上是对象内存空间的一个指针。

在Python中，从变量到对象的连接称为引用，以内存中的指针的形式实现，类似C语言中的指针。

a = 3

a = “spam”

a = 1.23

执行上述Python语句时，先将变量a与类型为整型的对象3关联，然后将变量a与类型为字符串的对象spam关联，最后将变量a与类型为浮点型的对象1.23相关联。从这里可以看出，类型是与对象相关联的。

Python中的垃圾回收：

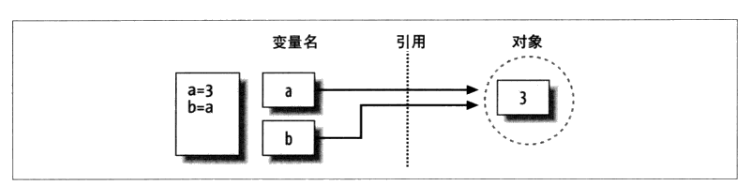
当变量被赋值一个新的对象时，其之前引用的对象有可能会被回收掉（如果没有其他变量引用它），这点类似C#中的对象的垃圾回收。

共享引用：

例：object\_type\_test/object\_type\_test1.py

a = 3  
b = a  
print('a = ' + str(a))  
print('b = ' + str(b))

执行b = a时，会将b引用到a所引用的对象3上。



如上图所示，变量a和b同时引用到对象3上。

输出为：

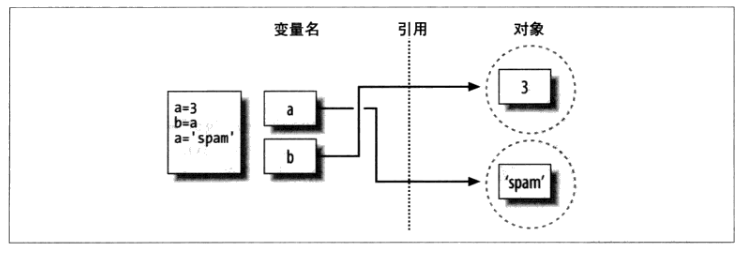
a = 3

b = 3

继续执行：

a = 'spam'

a被赋予一个新的对象spam。



如上图所示，a引用到spam对象上，b仍然引用到对象3上。

输出为：

a = spam

b = 3

c = 3  
d = c  
c = c + 2  
print('c = ' + str(c))  
print('d = ' + str(d))

输出为：

c = 5

d = 3

从上述例程中可以看出，两个变量引用到同一个对象后，如果修改了其中一个变量所引用的对象，另一个变量并不会受到影响。原因是上述变量最初引用的对象是不可变对象。

Python中的不可变对象：一旦创建后不可以改变。

数字、字符串、tuple等均为不可变对象。

Python中对于不可变对象的引用类似于C语言中的指针：当两个指针指向同一块内存地址后，如果改变了其中一个指针的指向，不会影响到另一个指针。

Python中指向可变对象和不可变对象的变量都类似于C语言中的指针，如果有多个变量指向同一对象，单纯改变变量本身，对其他变量指向的对象没有影响。其中指向不可变对象的变量类似指向常量的指针，指针本身可以改变，但其指向的值不会改变。可变对象的变量可以通过变量修改其指向的对象，而且会影响其它指向同一对象的变量。

Python中的可变对象：

列表、元组、集合、字典等

Python中对可变对象的引用与不可变对象有所不同：

例：程序object\_type\_test/object\_type\_test1.py

L1 = [1, 2, 3]  
L2 = L1  
  
L1 = 24  
print("L1:" + str(L1))  
print("L2:" + str(L2))  
print()  
  
L1 = [4, 5, 6]  
print("L1:" + str(L1))  
print("L2:" + str(L2))

输出为：

L1:24

L2:[1, 2, 3]

L1:[4, 5, 6]

L2:[1, 2, 3]

从上述输出可以看出，Python中对可变对象的引用与C语言中的指针也类似，如果改变其中一个变量的引用对象，对另一个变量没有影响。

例：程序object\_type\_test/object\_type\_test1.py

L3 = [1, 2, 3]  
L4 = L3  
L3[0] = 99  
print("L3:" + str(L3))  
print("L4:" + str(L4))

print(id(L3))  
print(id(L4))

输出为：

L3:[99, 2, 3]

L4:[99, 2, 3]

2712827028168

2712827028168

从上述输出结果可以看出，对于可变对象而言，创建之后可以修改，如果通过其中一个变量修改了可变对象的值（这里并未改变变量的引用，而只是改变了其所引用对象的一个元素），类似于C语言中通过指针修改了其所指向的变量（C中的变量）的值，这样另一个变量也会受到影响。

如果希望消除这种影响，可以考虑不直接将变量赋值给另一个变量，而是通过拷贝的方式来赋值。

例：程序object\_type\_test/object\_type\_test1.py

L5 = [1, 2, 3]  
  
# 通过拷贝后，  
# L5和L6引用的对象不同  
L6 = L5[:]  
L5[0] = 99  
print("L5:" + str(L5))  
print("L6:" + str(L6))  
print(id(L5))  
print(id(L6))

输出为：

L5:[99, 2, 3]

L6:[1, 2, 3]

2712827960200

2712827986824

可以看出通过拷贝的方式后，L5和L6所引用的对象不同，其id号并不相同。