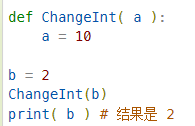
**一、输出、变量**

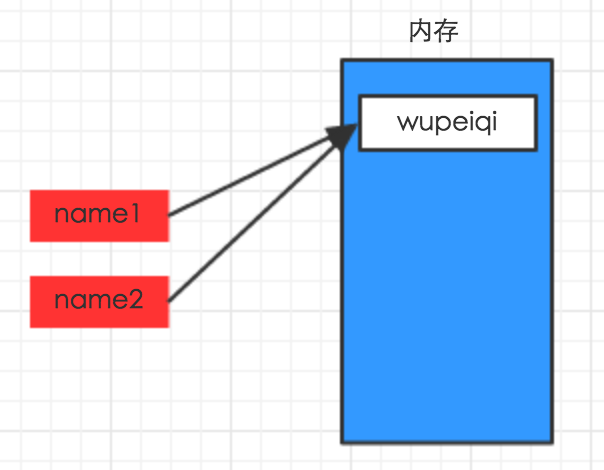
print("{0:\*^20,.3f}".format(12345.67890))

用\*所处位置表示的字符来补充不足位置，共20位，逗号表示显示千位分隔符，.3f表示精度为3位

python中所有的 变量都是指针，类型属于对象。

**不可变类型**：变量赋值 a=5 后再赋值 a=10，这里实际是新生成一个 int 值对象 10，再让 a 指向它，而 5 被丢弃，不是改变a的值，相当于新生成了a。当在函数中改变不可变类型的变量的值的时候，例如此时，有一个int类型的对象2，b指向2，再通过函数将b传至ChangeInt中时，实际上是将b的指针给了a，此时a指向2，而当再运行函数内部的a=10语句时，将a的指针指向10，再运行print(b)语句时，b其实没有改变。



此时再对name1进行修改，则改变的只是name1所对应的指针，不改变内存中的值。

**可变类型**：变量赋值 la=[1,2,3,4] 后再赋值 la[2]=5 则是将 list la 的第三个元素值更改，本身la没有动，只是其内部的一部分值被修改了。

也可以理解为list指令是申请空间（类似栈），如果我们定义一个list，例如：list1=[1,2,3]，

那么此时可以得到list1的地址，即id(list1)=4435060856 ，如果对list1进行数据的添加，例如list1.append(4)，则此时的list1为[1,2,3,4]，而地址id(list1)=4435060856，可以看出地址并无改变。

<https://www.cnblogs.com/Jacck/p/7790049.html> 可变与不可变详解

"{1} {0} {1}".format("hello", "world") # 设置指定位置

a, b = b, a + b 类似于这种算式，先计算等号右边的再赋值

**二、集合**

list是有序的，可以修改，通过下标获取值

tuple是有序的，不可以修改，通过下标获取值

字典是无序的，通过键获取值，查找效率高

str1.split() "abc 123"→['abc' , ‘123’] 用split将该行以空格为准分割成列表

字符比较时，不能直接用list中的元素进行比较，重复添加下标会导致报错，应定义后再进行比较

strA=list1[0]

strB=list1[1]

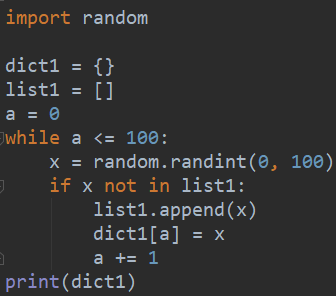
print (strA.upper()>strB.upper())

list2.reverse() 倒转list

元组（tuple）用小括号，列表（list）用中括号

元组中只包含一个元素时，需要在元素后面添加逗号，否则括号会被当作运算符使用

随机数字典 自己的方法



**三、迭代器**

迭代器是一个可以记住遍历的位置的对象。

常用：

try:

print(next(it))

except StopIteration:

sys.exit()

在调用生成器运行的过程中，每次遇到 yield 时函数会暂停并保存当前所有的运行信息，返回 yield 的值, 并在下一次执行 next() 方法时从当前位置继续运行。