# 数据对象和其引用

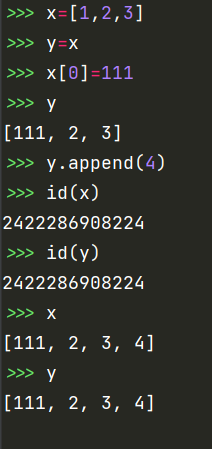
## Python中变量的本质——引用

### 变量和可变数据类型

如下例子中，x变量还指向了一个列表[1,2,3],然后有使用了一个变量y来指向x。

再通过y来操作数据对象，然后通过x和y查看数据对象的内容发现修改的内容是一样的，查看x和y的id得到的值是一样的，也就是x和y只是数据对象的引用，他们指向的数据实际上是同一份。所有通过变量x对数据对象的改变都可以反映到变量y上，反之亦然。

变量的赋值y=x这一操作实际上是复制了一分数据对象的引用。

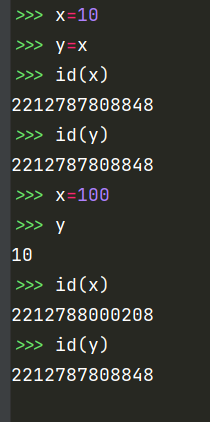


### 变量和不可变数据类型1

如下的例子中，x=10；y=x这两句，使用变量x指向了数据对象10，探后使用y=x这一句，让变量y也指向了10，而且此时查看x和y的id的值是一样的。

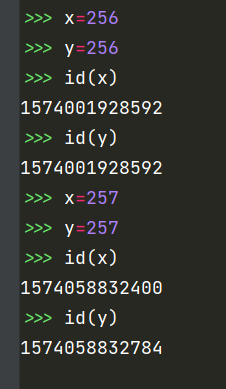
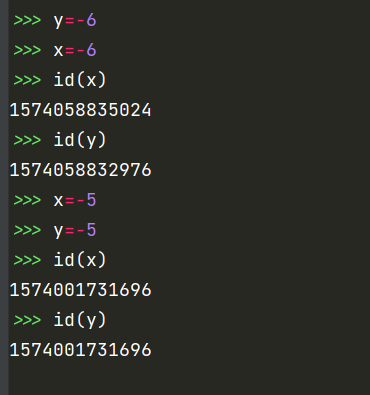
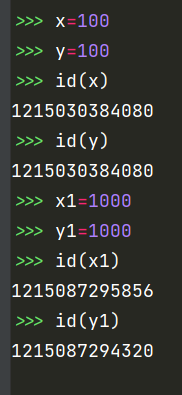
然后通过x=100给变量x重新赋值为100，再次查看y发现其值仍然是原来的10，而y通过赋值语句变为了100，此时再次查看x和y的值发现不一样了，也就是在重新给y赋值后，x和y不再是指向了同一个对象。

其实很好理解，y在一开始通过语句y=x使得y和x引用指向了同一个数据对象，但是在y=100后，原来的y引用断开了，指向了型的数据对象100，而原来的x还是保持原来的指向10，他们现在是指向了两个单独的对象了。



### 变量和不可变数据类型2——特殊的数据范围（-5）~256

查看下面的两组对比，**可以看到在(-5)~256这个范围内的整数，如果有两个变量x=n;y=n(n的值在-5~256范围内)，那么查看x和y的值是一样的，也就是在-5~256这个范围内，不论有多少的变量，如果他们被赋值为同一个该范围的整数值，那么这些变量其实都是引用的同一分数据对象**。

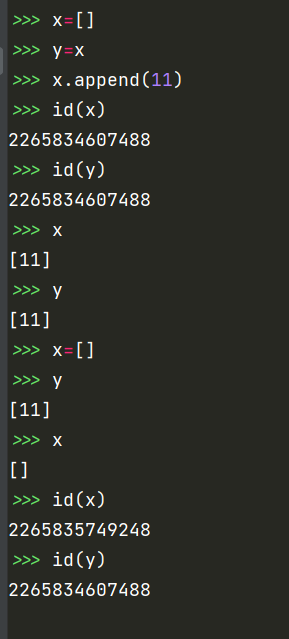
  

**在-5~256这个范围之外，x=n;y=n;这样的赋值语句（n在前后值相同），那么这些变量x,y其实每一份都是引用单独的数据对象，彼此都是独立的同一份**。

### 字典，列表的初始化定义和clear——数据对象的清空

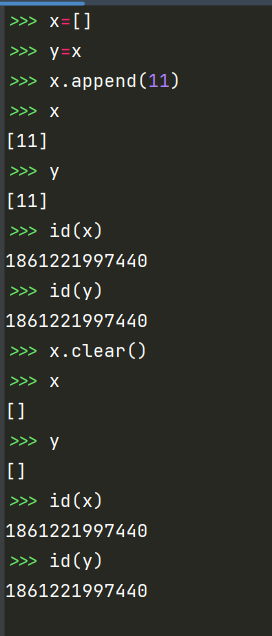
如下例子中，使用x=[]定义一个列表，然后使用y=x用y复制了x的一个引用。对列表进行append（11）值后，查看x和y的id发现两者是相同的，查看x和y的值也是相同的，都是append后的列表。

然后再次使用x=[]相当于清空原来的x列表，但是此时再次查看x和y的id发现，两者的id值不一样了，y仍然保留原来的列表，但是x经过再次的x=[]后，已经是一个新的对象了。



在使用字典也有这个现象。也就是对于x=[]或者x={}这种做法，是定义一个新的数据对象，某种程度上来说，确实具有清理原来的列表或者字段对象的效果，但是实际上是新定义了一个数据对象，并不是清空。

要真正的清空列表或者字段，需要使用列逼走或者元组的clear方法，如下：



使用clear方法才是真正的清空原来的数据对象内容。可以到x和y的id在clear以后还是相同的没有改变。

究其原因还是回归到python中数据对象和其引用的问题上来。在x=[]；y=x;这里是创建了一个列表数据对象来使用x这个变量来指向它，也就是x是定义的列表的引用，然后y=x就是又多了一个引用来指向定义的列表，此时列表有了两个引用x,y，所以可以通过x和y这两个变量来对列表数据对象操作，append或者insert等。因为x和y指向的是同一个数据对象，多以id(x）和 id(y)都是相同的。通过x和y对数据对象的操作都可以通过x和y变量本身相互获得和反映出来。但是在x=[]之后，相当于原来变量x指向的引用断开了原来的指向，新指向了一个新的列表数据对象，而原来的变量y还是保持原来的引用指向没有改变，多以此时通过y查看还是原来的数据内容，但是x指向了新的数据对象。