例如

```
list1 = [1,2,3,4,5,6]
list2 = list1.copy()
print(list2)

#运行程序
[1, 2, 3, 4, 5, 6]
```

需要注意，使用赋值语句也可以实现上述功能。

例如

```
list1 = [1,2,3,4,5,6]
list2 = list1
print(list2)
```

此程序使用赋值语句实现列表的复制，如果对 list1 进行操作，那么 list2 也会随之变化，原因在于 Python 语言中这种列表复制操作根本上只是为列表增加了一个别名，list1 和 list2 指向的地址是相同的。

例如

```
list1 = [1,2,3,4,5,6]
list2 = list1
list1.pop(3)          #删除索引值为 3 的元素
print(list2)

#运行程序
[1, 2, 3, 5, 6]
```

5.1.6 列表的操作符

列表也支持一些操作符运算，例如“+”“+=”“*”和“*=”，运算规则与数字类型不同。数字类型是不可变数据类型，所以不管用什么操作符进行运算，其结果所指向的内存地址都会改变。列表类型是可变数据类型，当使用“+=”和“*=”操作符运算时，其结果指向的内存地址不会改变；而当使用“+”和“*”操作符运算时，其结果指向的内存地址会改变。

id()函数可以返回变量所指向的内存地址，接下来将用 id()函数举例说明数字类型和列表类型经过操作符运算过后内存地址的变化情况。

例如

```
>>>x=3
>>>id(x)
8791256490704          #地址数值不恒定，读者得到的数值与此数值不同是正常现象，下同
>>>x+=2
>>>id(x)
8791256490768          #因为整数类型是不可变的数据类型，所以改变了数值，它所指向的内存地址也
                       会跟着改变，下同
>>>x=x+2
>>>id(x)
8791256490832
```

下面是列表使用操作符时，内存地址的变化情况。