参数传递

在 python 中，类型属于对象，变量是没有类型的：

```
a=[1,2,3]

a="Runoob"
```

以上代码中，[1,2,3] 是 List 类型，"Runoob" 是 String 类型，而变量 a 是没有类型，她仅仅是一个对象的引用（一个指针），可以是 List 类型对象，也可以指向 String 类型对象。

可更改(mutable)与不可更改(immutable)对象

在 python 中，strings, tuples, 和 numbers 是不可更改的对象，而 list,dict 等则是可以修改的对象。

- **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。**
- **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。**

python 函数的参数传递：

- **不可变类型：类似 c++ 的值传递，如 整数、字符串、元组。如fun（a），传递的只是a的值，没有影响a对象本身。比如在 fun（a）内部修改 a 的值，只是修改另一个复制的对象，不会影响 a 本身。**
- **可变类型：类似 c++ 的引用传递，如 列表，字典。如 fun（la），则是将 la 真正的传过去，修改后fun外部的la也会受影响**

python 中一切都是对象，严格意义我们不能说值传递还是引用传递，我们应该说传不可变对象和传可变对象。

python 传不可变对象实例

```
#!/usr/bin/python3

def ChangeInt( a ):
    a = 10

b = 2
ChangeInt(b)
print( b ) # 结果是 2
```

实例中有 int 对象 2，指向它的变量是 b，在传递给 ChangeInt 函数时，按传值的方式复制了变量 b，a 和 b 都指向了同一个 Int 对象，在 a=10 时，则新生成一个 int 值对象 10，并让 a 指向它。

传可变对象实例

可变对象在函数里修改了参数，那么在调用这个函数的函数里，原始的参数也被改变了。例如：

```
#!/usr/bin/python3

# 可写函数说明
def changeme( mylist ):
   "修改传入的列表"
   mylist.append([1,2,3,4]);
   print ("函数内取值: ", mylist)
   return

# 调用changeme函数
mylist = [10,20,30];
changeme( mylist );
print ("函数外取值: ", mylist)
```

传入函数的和在末尾添加新内容的对象用的是同一个引用。故输出结果如下：

```
函数内取值:  [10, 20, 30, [1, 2, 3, 4]]
函数外取值:  [10, 20, 30, [1, 2, 3, 4]]
```