在写代码的时候，免不了要使用变量。但程序中的一个变量并不一定是在哪里都可以被使用，根据情况不同，会有不同的“有效范围”。看这样一段代码：

def func(x):

print 'X in the beginning of func(x): ', x

x = 2

print 'X in the end of func(x): ', x

x = 50

func(x)

print 'X after calling func(x): ', x

输出：

X in the beginning of func(x): 50

X in the end of func(x): 2

X after calling func(x): 50

变量 x 在函数内部被重新赋值。但在调用了函数之后，x 的值仍然是50。为什么？

这就得说一下变量的“作用域”：

当函数内部定义了一个变量，无论是作为函数的形参，或是另外定义的变量，它都只在这个函数的内部起作用。函数外即使有和它名称相同的变量，也没有什么关联。这个函数体就是这个变量的作用域。像这样在函数内部定义的变量被称为“局部变量”。

要注意的是，作用域是从变量被定义的位置开始。像这样的写法是有问题的：

def func():

print y

y = 2

print y

报错：

UnboundLocalError: local variable 'y' referenced before assignment

因为在 y = 2 之前，y 并不存在，调用 y 的值就会出错。

回到开始那个例子：

在函数 func 外部，定义的变量 x，赋值为 50，作为参数传给了函数 func。而在函数 func 内部，变量 x 是形参，它的作用域是整个函数体内部。它与外面的那个 x 没有关系。只不过它的初始值是由外面那个 x 传递过来的。

所以，虽然函数体内部的 x 被重新赋值为 2，也不会影响外面那个 x 的值。

不过有时候，我们希望能够在函数内部去改变一些变量的值，并且这些变量在函数外部同样被使用到。怎么办？

一种方法是，用 return 把改变后的变量值作为函数返回值传递出来，赋值给对应的变量。比如开始的那个例子，可以在函数结尾加上

return x

然后把调用改为

x = func(x)

还有一种方法，就是使用“全局变量”。

在 Python 的函数定义中，可以给变量名前加上 global 关键字，这样其作用域就不再局限在函数块中，而是全局的作用域。

通过 global 改写开始的例子：

def func():

global x

print 'X in the beginning of func(x): ', x

x = 2

print 'X in the end of func(x): ', x

x = 50

func()

print 'X after calling func(x): ', x

输出：

X in the beginning of func(x): 50

X in the end of func(x): 2

X after calling func(x): 2

函数 func 不再提供参数调用。而是通过 global x 告诉程序：这个 x 是一个全局变量。于是函数中的 x 和外部的 x 就成为了同一个变量。这一次，当 x 在函数 func 内部被重新赋值后，外部的 x 也随之改变。

前面讲的局部变量和全局变量是 Python 中函数作用域最基本的情况。实际上，还有一些略复杂的情况，比如：

def func():

print 'X in the beginning of func(x): ', x

# x = 2

print 'X in the end of func(x): ', x

x = 50

func()

print 'X after calling func(x): ', x

输出：

X in the beginning of func(x): 50

X in the end of func(x): 50

X after calling func(x): 50

程序可以正常运行。虽然没有指明 global，函数内部还是使用到了外部定义的变量。然而一旦加上

x = 2

这句，程序就会报错。因为这时候，x 成为一个局部变量，它的作用域从定义处开始，到函数体末尾结束。

建议在写代码的过程中，显式地通过 global 来使用全局变量，避免在函数中直接使用外部变量。