nonlocal Scope

Python 3 added a new keyword called **nonlocal**. The nonlocal keyword adds a scope override to the inner scope. You can read all about it in PEP 3104. This is best illustrated with a couple of code examples. One of the most common examples is to create function that can increment:

```
def counter():
    num = 0
    def incrementer():
        num += 1
        return num
    return incrementer
```

If you try running this code, you will receive an **UnboundLocalError** because the **num** variable is referenced before it is assigned in the innermost function. Let's add nonlocal to the mix:

```
def counter():
    num = 0
    def incrementer():
        nonlocal num
        num += 1
        return num
    return incrementer
c = counter()

print (c)
#<function counter.<locals>.incrementer at 0x7f78918f3378>

print (c())
#1

print (c())
#2

print (c())
#3
```

Now our incrementing function works as we expected it to. As a side note, this type of function is known as a **closure**. A closure is basically a block of code that “closes” nonlocal variables. The idea behind a closure is that you can reference variables that are defined outside of your function.

Basically nonlocal will allow you to assign to variables in an outer scope, but not a global scope. So you can’t use nonlocal in our **counter** function because then it would try to assign to a global scope. Give it a try and you will quickly get a **SyntaxError**. Instead you must use nonlocal in a nested function.

Wrapping Up

In this chapter, we learned that we can change how a variable is referenced using Python global and nonlocal keywords. We learned where you can use and why you might want to. We also learned about local scope. Have fun and keep experimenting!