Another example

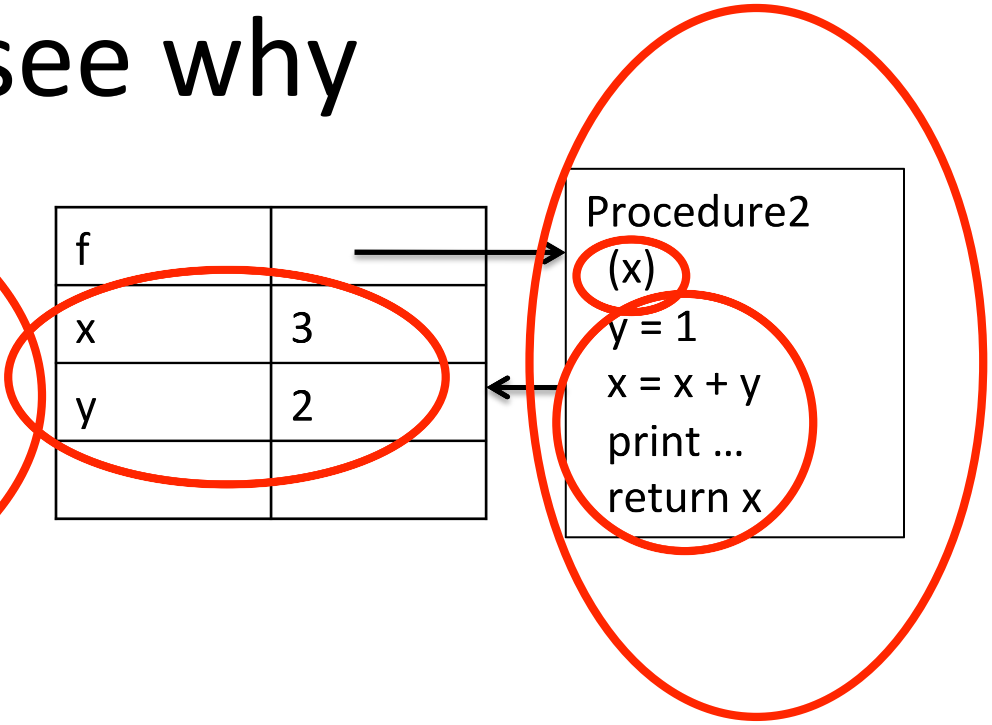
```
def f(x):
    y = 1
    x = x + y
    print('x = ' + str(x))
    return x

x = 3
y = 2
z = f(x)
print('z = ' + str(z))
print('x = ' + str(x))
print('y = ' + str(y))
```

- Causes the following to appear in the Python shell

```
x = 4
z = 4
x = 3
y = 2
```

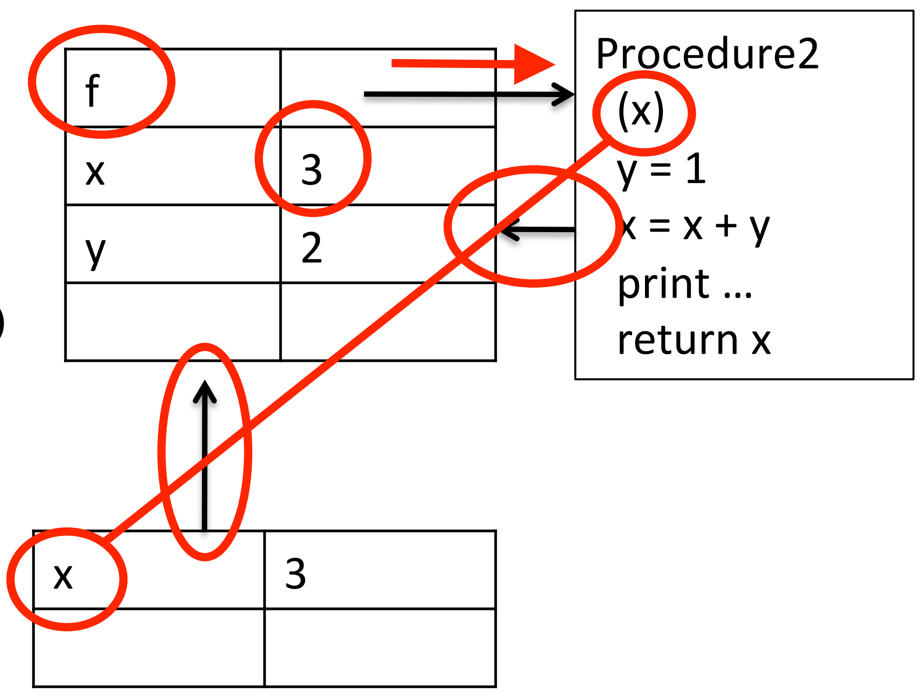
Let’s see why

```
def f(x):
    y = 1
    x = x + y
    print(‘x = ‘ = str(x))
    return x

x = 3
y = 2
z = f(x)
print(‘z = ‘ + str(z))
print(‘x = ‘ + str(x))
print(‘y = ‘ + str(y))
```



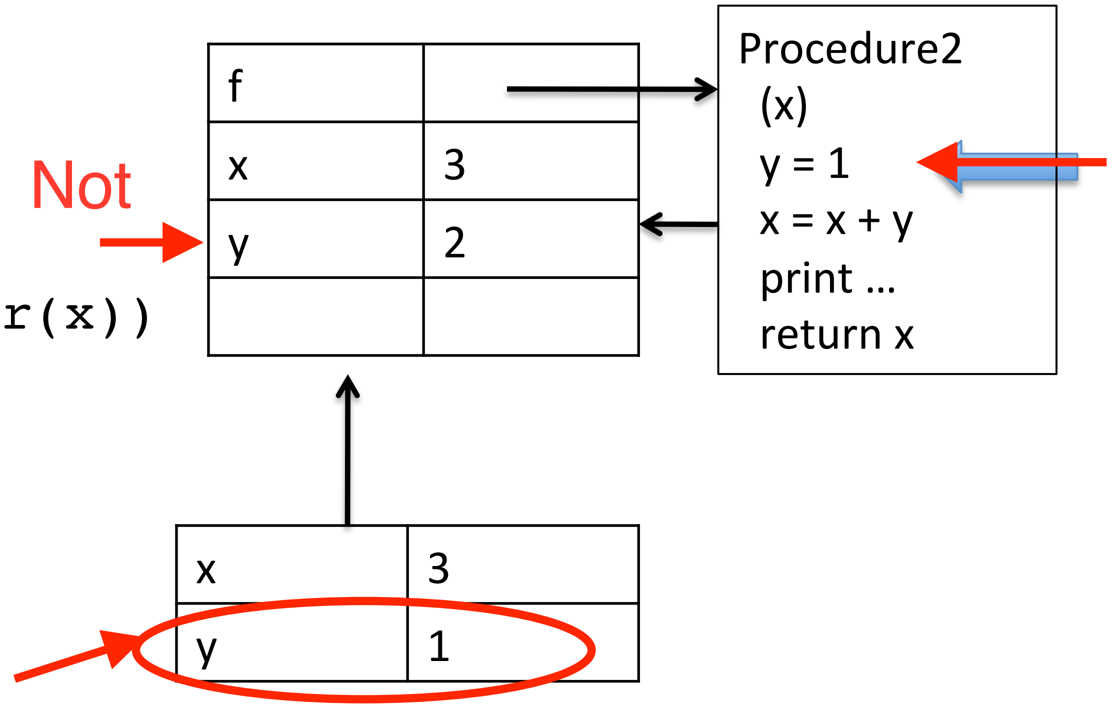
Let’s see why

```
def f(x):
    y = 1
    x = x + y
    print(‘x = ‘ = str(x))
    return x

x = 3
y = 2
z = f(x)
print(‘z = ‘ + str(z))
print(‘x = ‘ + str(x))
print(‘y = ‘ + str(y))
```



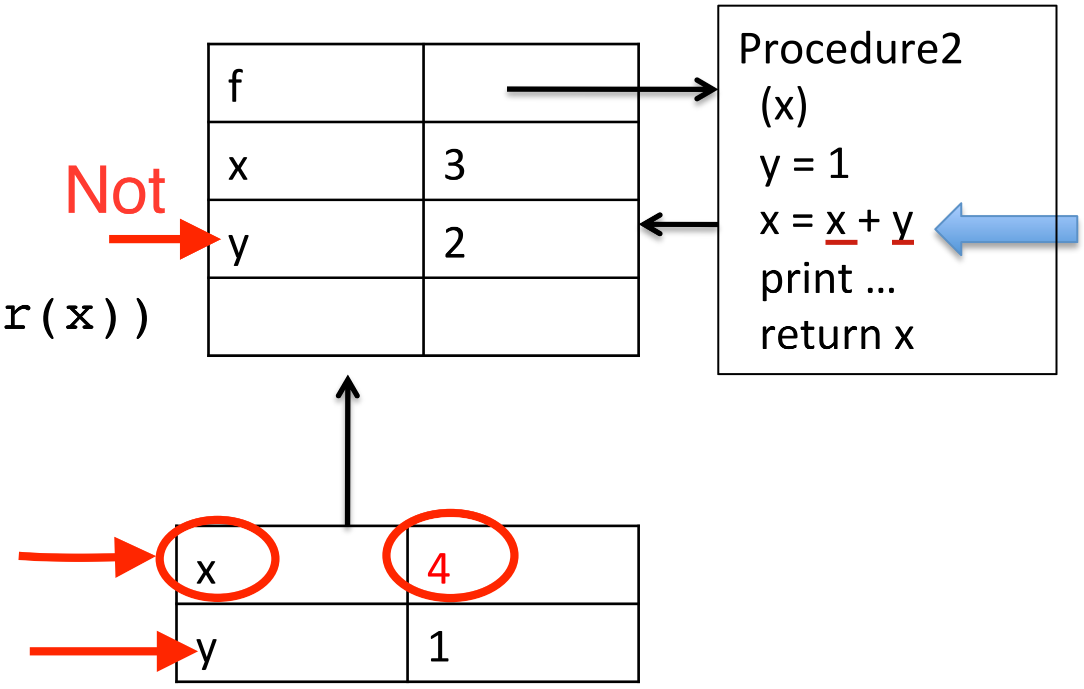
Let's see why

```
def f(x):
    y = 1
    x = x + y
    print('x = ' = str(x))
    return x

x = 3
y = 2
z = f(x)
print('z = ' + str(z))
print('x = ' + str(x))
print('y = ' + str(y))
```



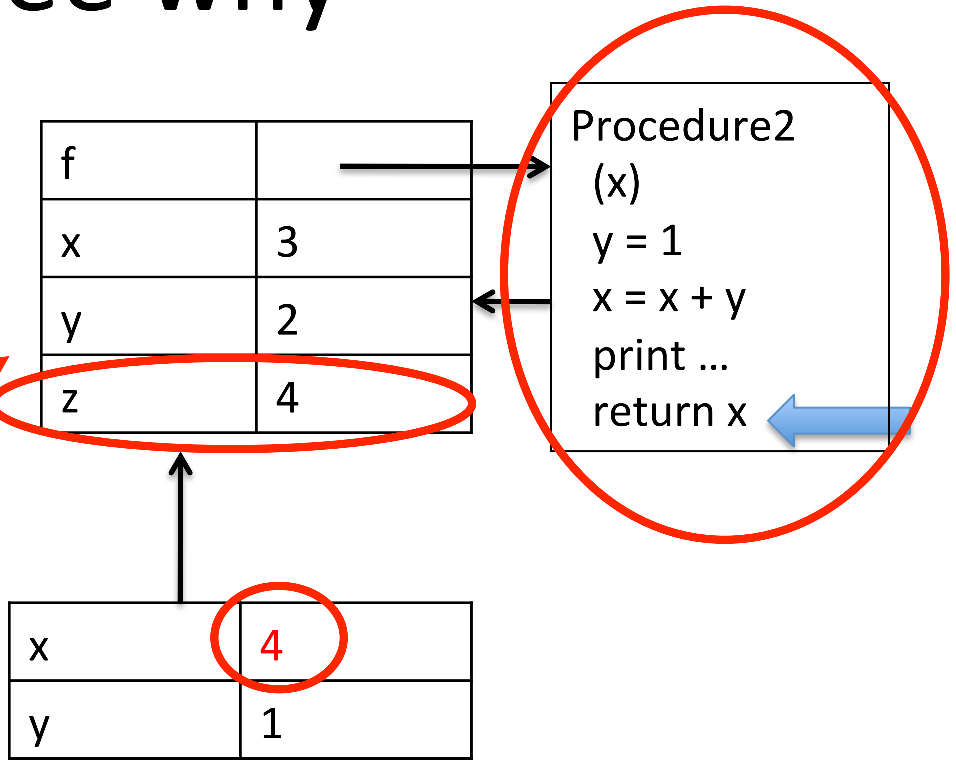
Let’s see why

```
def f(x):
    y = 1
    x = x + y
    print(‘x = ‘ = str(x))
    return x

x = 3
y = 2
z = f(x)
print(‘z = ‘ + str(z))
print(‘x = ‘ + str(x))
print(‘y = ‘ + str(y))
```



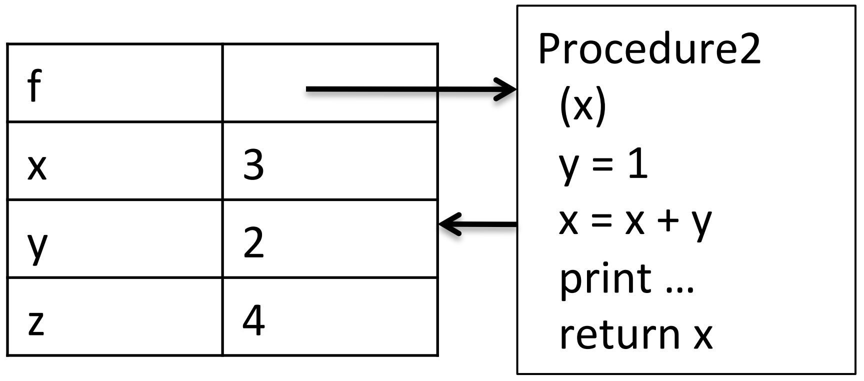
Let’s see why

```
def f(x):
    y = 1
    x = x + y
    print(‘x = ‘ = str(x))
    return x

x = 3
y = 2
z = f(x)
print(‘z = ‘ + str(z))
print(‘x = ‘ + str(x))
print(‘y = ‘ + str(y))
```



Let’s see why

```
def f(x):
    y = 1
    x = x + y
    print(‘x = ‘ = str(x))
    return x

x = 3
y = 2
z = f(x)
print(‘z = ‘ + str(z))
print(‘x = ‘ + str(x))
print(‘y = ‘ + str(y))
```



Now control reverts to the global environment, where the values of x, y and z are visible

Some observations

- Each function call creates a new environment, which scopes bindings of formal parameters and values, and of local variables (those created with assignments within body)
- Scoping often called static or lexical because scope within which variable has value is defined by extent of code boundaries