12.13. Passing Mutable Objects

As you have seen, when a function (or method) is invoked and a parameter value is provided, a new stack frame is created, and the parameter name is bound to the parameter value. What happens when the value that is provided is a mutable object, like a list or dictionary? Is the parameter name bound to a *copy* of the original object, or does it become an alias for exactly that object? In python, the answer is that it becomes an alias for the original object. This answer matters when the code block inside the function definition causes some change to be made to the object (e.g., adding a key-value pair to a dictionary or appending to a list).

This sheds a little different light on the idea of parameters being *local*. They *are* local in the sense that if you have a parameter x inside a function and there is a global variable x, any reference to x inside the function gets you the value of local variable x, not the global one. If you set `x = 3`, it changes the value of the local variable x, but when the function finishes executing, that local x disappears, and so does the value 3.

If, on the other hand, the local variable x points to a list `[1, 3, 7]`, setting `x[2] = 0` makes x still point to the same list, but changes the list's contents to `[1, 3, 0]`. The local variable x is discarded when the function completes execution, but the mutation to the list lives on if there is some other variable outside the function that also is an alias for the same list.

Consider the following example.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
1  def double(y):
2      y = 2 * y
3
4  def changeit(lst):
5      lst[0] = "Michigan"
6      lst[1] = "Wolverines"
7
8  y = 5
9  double(y)
10 print(y)
11
12 mylst = ['our', 'students', 'are', 'awesome']
13 changeit(mylst)
14 print(mylst)
15
```

```
5
['Michigan', 'Wolverines', 'are', 'awesome']
```

Activity: 1 -- ActiveCode (ac11_12_1)

Try running it. Similar to examples we have seen before, running `double` does not change the global y. But running `changeit` does change `mylst`. The explanation is above, about the sharing of mutable objects. Try stepping through it in codelens to see the difference.

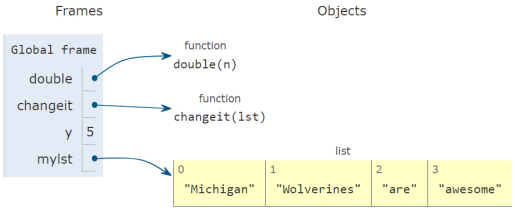
Python 3.3

```
1  def double(n):
2      n = 2 * n
3
4  def changeit(lst):
5      lst[0] = "Michigan"
6      lst[1] = "Wolverines"
7
8  y = 5
9  double(y)
10 print(y)
11
12 mylst = ['106', 'students', 'are', 'awesome']
13 changeit(mylst)
14 print(mylst)
```

<< First | < Back | Program terminated | Forward > | Last >>

line that has just executed
next line to execute

Visualized using Online Python Tutor by Philip Guo



Program output:

```
5
['Michigan', 'Wolverines', 'are', 'awesome']
```

Activity: 2 -- CodeLens: (clens11_12_1)

You have attempted 3 of 2 activities on this page

| Back to top