Assignment 7c

Noah Hinojos

February 21, 2024

In Python, mutation and rebinding both alter a variable's definition. The difference is that mutation does not construct a new object when redefining. Instead, mutation alters the internal state of the variable's value itself. Rebinding, on the other hand, redefines the variable by creating an entirely new value. Now, rebinding may or may not utilize the prior variable's value but it does define using a separately constructed value. In the analogy of painting a picture, Mutation can be thought of adding a tree to a landscape or by changing the color of all the mountains. Conversely, rebinding can be thought of as painting these changes by starting from a brand new canvas and not by altering the old one.

Here's an example psuedocode that may trip someone up that does not know the difference. In this program I *intend* to reverse the order of the elements in a list, but can you spot the issue?

```
some_list = [1, 2, 3, 4, 5]
for i in range(len(some_list) // 2):
    some_list[i] = some_list[len(some_list) - i - 1]
```

Now if you were to print the final value of some_list you would not get the reversed form: [5, 4, 3, 2, 1]. Instead, you would get: [5, 4, 3, 4, 5]. This is because the program mutates the original list when attempting to reverse it, and hence the program uses a newly mutated version for each iteration.

A solution to this is to use rebinding:

```
some_list = [1, 2, 3, 4, 5]
new_list = []
for i in range(len(some_list)):
    new_list.append(some_list[len(some_list) - i - 1])
some_list = new_list
```

This does in fact reverse the list in place by creating a new_list object to track changes. Then, it redefines some_list to be the new_list object.