EECS 345 Written 2

Patrick Landis (pal25)

April 9, 2018

Problem 1

a.)

list = [1, 2, 3, 4, 5]

The list isn't modified because all variables being changed in the function swap are local to that function.

b.)

list = [1, 2, 2, 4, 5]

Since list[save] is evaluated when the function is called, anytime b is updated we're updating the value of list[2]. Since the references of a and b are swapped the value of list[2] is updated to 2.

c.)

list = [1, 2, 2, 4, 5]

list[save] is evaluated when the function is called. Then whatever the value of b at the end of the function will be copied back into list[2].

d.)

list = [1, 2, 3, 3, 5]

All a references change to save and all b references change to list[save]. These values are evaluated when they are executed.

Problem 2

a.)

list = [1, 2, 3, 4, 5]

The list isn't modified because all variables being changed in the function swap are local to that function.

b.)

list = [1, 2, 6, 4, 5]

Since list[save] is evaluated when the function is called, anytime val2 is updated we're updating the value of list[2].

c.)

list = [1, 2, 6, 4, 5]

list[save] is evaluated when the function is called. Then whatever the value of val2 at the end of the function will be copied back into list[2].

d.)

list = Array Out of Bounds Exception (at least in Java)

All val1 references change to save and all val2 references change to list[save]. These values are evaluated when they are executed. The problem is that save = save + list[save] = 5. Later list[save] is used which is not do-able.

Problem 3

a.)

All of the variables, A, B, C, D are structually equivalent since they are all an array of 10 ints.

b.)

Variables A, and B are strict name equivalents since they are both of type T. C is of type C and D is of type int[1...10].

c.)

Variables A, B, and C are loose name equivalents since S is of type T. D is still a different type.

Problem 4

Reference Counting: We can immediately free tombstones which might be a benefit depending. With reference counting we already only need to keep track of the counts as well as the pointers for the tombstones. Since tombstones are used memory locations freeing immediately saves room overall.

Mark and Sweep: Arguably there are benefits to mark and sweep as well. With mark and sweep we don't need to keep a reference count and depending on how big memory allocations are GC might not take too long to run mark and sweep. Also since the tombstones themselves are such a small amount of memory it might not be necassary to free them immedlately.

Overall they are about equal and both have their pros and cons.

Problem 5

We can wrap a function in a function closure. From a python's example on wikipedia:

```
def counter():
    x = 0
    def increment(y):
        nonlocal x
        x += y
        print(x)
    return intrement

count = counter()
count(1) #prints 1
count(1) #prints 2
count(3) #prints 5
count(1) #prints 6
```

Each time the function is evaluated it returns a different result.