CSCI046 Homework 5: Scope and Memory Management

DUE: Thursday, 27 February beginning of class

Name: Ethan Kurz

Collaboration policy: You are allowed to use any resources you would like to complete this assignment, and you are encouraged to work in teams. Remember, learning the material is your responsibility, so collaborate in a way that will help you learn.

I highly recommend that you first try to evaluate the python code without a computer, and only use a computer to check your work. If you don't understand a problem, then use `https://pythontutor.com` to go step by step through the code.

Note: By default, all variables are **global** and accessible anywhere inside or outside a function. A variable becomes **local** if it defined within a function and the `global` keyword is not used.

Problem 1. Write the output of the following python code.

```
1 xs = [1,2,3]
2 def foo():
3     print('xs=',xs)
4 foo()
```

xs= [1,2,3]

Problem 2. Write the output of the following python code.

```
1 xs = [1,2,3]
2 def foo():
3     xs = 'a'
4     print('xs=',xs)
5 foo()
```

xs = a

Note: Modifying a global variable does not make the variable local. The only way to make a variable local is if it appears to the left of an equals sign *by itself* (i.e. appearing to the left of an equal sign does not matter if there is a slice). Calling a function that modifies a variable does not make a variable local.

Problem 3. Write the output of the following python code.

```
1 xs = [1,2,3]
2 def foo():
3     xs = 'a'
4 foo()
5 print('xs=',xs)
```

xs = [1, 2, 3]

Problem 4. Write the output of the following python code.

```
1 xs = [1,2,3]
2 def foo():
3     xs[-1] = 'a'
4 foo()
5 print('xs=',xs)
```

xs = [1, 2, 'a']

Problem 5. Write the output of the following python code.

```
1 xs = [1,2,3]
2 def foo():
3     xs.pop()
4 foo()
5 print('xs=',xs)
```

xs = [1, 2]

Problem 6. Write the output of the following python code.

```
1 xs = [1,2,3]
2 def foo():
3     xs.append('a')
4 foo()
5 print('xs=',xs)
```

xs = [1,2,3,'a']

Problem 7. Write the output of the following python code.

```
1  xs = [1,2,3]
2  def foo():
3      xs.pop()
4  def bar():
5      global xs
6      xs = [4,5,6]
7      xs.append('a')
8  foo()
9  bar()
10 bar()
11 foo()
12 foo()
13 print('xs=',xs)
```

xs = [4,5]

Problem 8. Write the output of the following python code.

```
1 xs = [1,2,3]
2 def foo():
3     xs = [4,5,6]
4     xs.append('a')
5 foo()
6 print('xs=',xs)
```

xs = [1, 2, 3]

Problem 9. Write the output of the following python code.

```
1 xs = [1,2,3]
2 def foo():
3     xs = [4,5,6]
4     xs.append('a')
5     print('xs=',xs)
6 foo()
```

xs = [4, 5, 6, 'a']

Problem 10. Write the output of the following python code.

```
1 xs = [1,2,3]
2 def foo():
3     global xs
4     xs = [4,5,6]
5     xs.append('a')
6 foo()
7 print('xs=',xs)
```

xs = [4, 5, 6, 'a']

Problem 11. ~~Write the output of the following python code.~~ The following code causes an error in python. Enter the code into the python interpreter; make sure you understand the error message and why the error is occurring. You do not have to write anything for this problem.

```
1 xs = [1,2,3]
2 def foo():
3     xs.append('a')
4     xs = [4,5,6]
5 foo()
6 print('xs=',xs)
```

Problem 12. Write the output of the following python code.

```
1 xs = [1,2,3]
2 def foo():
3     global xs
4     xs.append('a')
5     xs = [4,5,6]
6 foo()
7 print('xs=',xs)
```

xs = [4,5,6]

Note: Assignment makes two variable names refer to the same object. Changing the contents of one variable actually changes the contents of the object, and therefore changes the contents of both variables. In order to create new, distinct, objects, you must copy the variable.

Problem 13. Write the output of the following python code.

```
1 xs = [1,2,3]
2 ys = xs
3 ys.append('a')
4 print('xs=',xs)
5 print('ys=',ys)
```

xs=[1,2,3,'a']
ys=[1,2,3,'a']

Problem 14. Write the output of the following python code.

```
1 import copy
2 xs = [1,2,3]
3 ys = copy.copy(xs)
4 ys.append('a')
5 print('xs=',xs)
6 print('ys=',ys)
```

xs=[1,2,3]
ys=[1,2,3,'a']

Note: When lists contain non-container objects like integers, `copy` and `deepcopy` behave exactly the same way. When lists contain containers, then copy and deep copy behave differently.

Problem 15. Write the output of the following python code.

```
1 import copy
2 xs = [[1,2,3],[4,5,6]]
3 ys = copy.copy(xs)
4 ys[0][0] = 'a'
5 xs[1][1] = 'b'
6 print('xs=',xs)
7 print('ys=',ys)
```

xs = [[a, 2, 3], [4, 'b', 6]]

ys = [[a, 2, 3], [4, 'b', 6]]

Problem 16. What is the output of the previous code if we change `copy.copy` to `copy.deepcopy`?

xs = [[1, 2, 3], [4, 'b', 6]]

ys = [['a', 2, 3], [4, 5, 6]]

Problem 17. Write the output of the following python code.

```
1 import copy
2 xs = [[1,2,3],[4,5,6]]
3 ys = copy.copy(xs)
4 ys.append('a')
5 ys[0].append('b')
6 print('xs=',xs)
7 print('ys=',ys)
```

xs = [[1, 2, 3, 'b'], [4, 5, 6]]

ys = [[1, 2, 3, 'b'], [4, 5, 6], 'a']

Problem 18. What is the output of the previous code if we change `copy.copy` to `copy.deepcopy`?

~~xs would have~~

xs = [[1, 2, 3], [4, 5, 6]]

ys = [[1, 2, 3, 'b'], [4, 5, 6], 'a']

Note: Variables that are parameters to a function are always local variables. But, if a default parameter is used, these objects are only created once and the same copy is used in all subsequent calls that require a default parameter.

Problem 19. Write the output of the following python code.

```
1  xs = [1,2,3]
2  def foo(xs=[]):
3      print('xs=',xs)
4      xs.append(len(xs)+1)
5  foo()
6  foo()
7  foo()
8  foo([])
9  foo()
10 foo()
```

xs = [ ]
xs = [1]
xs = [1, 2]
xs = [ ]
xs = [1, 2, 3]
xs = [1, 2, 3, 4]

Note: Unfortunately, default parameters behave differently for container objects and non-container objects.

Problem 20. Write the output of the following python code.

```
1  n = 7
2  def foo(n=0):
3      print('n=',n)
4      n+=1
5  foo()
6  foo()
7  foo()
8  foo(1)
9  foo()
10 foo()
```

n = 0
n = 0
n = 0
n = 1
n = 0
n = 0

Note: The following problems illustrate different ways that you might try to reverse a list in python. At first glance, they all look the same. Due to memory management issues, however, they are not all correct. Understanding memory management is important when tracking down these subtle bugs.

Problem 21. Write the output of the following python code.

```
1  def reverse_list(xs):
2      ys = xs
3      for i in range(len(ys)):
4          ys[i] = xs[-i-1]
5      return ys
6
7  xs = [1,2,3]
8  ys = reverse_list(xs)
9  print('xs=',xs)
10 print('ys=',ys)
```

xs = [3,2,3]

ys = [3,2,3]

Problem 22. Write the output of the following python code.

```
1  import copy
2  def reverse_list(xs):
3      ys = copy.copy(xs)
4      for i in range(len(ys)):
5          xs[i] = ys[-i-1]
6      return ys
7
8  xs = [1,2,3]
9  ys = reverse_list(xs)
10 print('xs=',xs)
11 print('ys=',ys)
```

xs = [3,2,1]

ys = [1,2,3]

Problem 23. Write the output of the following python code.

```
1  import copy
2  def reverse_list(xs):
3      ys = copy.copy(xs)
4      for i in range(len(ys)):
5          ys[i] = xs[-i-1]
6      return ys
7
8  xs = [1,2,3]
9  ys = reverse_list(xs)
10 print('xs=',xs)
11 print('ys=',ys)
```

xs = [1, 2, 3]

ys = [3, 2, 1]

Problem 24. Write the output of the following python code.

```
1  def reverse_list():
2      ys = xs
3      for i in range(len(ys)):
4          ys[i] = xs[-i-1]
5      return ys
6
7  xs = [1,2,3]
8  ys = reverse_list()
9  print('xs=',xs)
10 print('ys=',ys)
```

xs = [3, 2, 3]

ys = [3, 2, 3]

Note: Because reversing a list is a common task, python provides two inbuilt methods to accomplish it. The `reverse` function does not create a copy of a list, but instead changes the list in place. The `reversed` function makes a copy of the list, reverses the copy, and leaves the original list unmodified. Both functions are widely used in python code.

Problem 25. Write the output of the following python code.

```
1 xs = [1,2,3]
2 ys = xs.reverse()
3 print('xs=',xs)
4 print('ys=',ys)
```

xs = [3, 2, 1]

Problem 26. Write the output of the following python code.

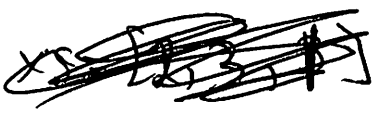
```
1 xs = [1,2,3]
2 ys = list(reversed(xs))
3 print('xs=',xs)
4 print('ys=',ys)
```

xs = [1, 2, 3]
ys = [3, 2, 1]

Note: Sorting is another common task, and python contains two functions for sorting lists: `sort` and `sorted`. It is a python convention that any function in the past tense returns a copy of the list without modifying the list, and any function written in the imperative modifies the list in place.

Problem 27. Write the output of the following python code.

```
1 xs = [2,3,1]
2 ys = xs.sort()
3 print('xs=',xs)
4 print('ys=',ys)
```



xs = [1, 2, 3]

Problem 28. Write the output of the following python code.

```
1 xs = [2,3,1]
2 ys = list(sorted(xs))
3 print('xs=',xs)
4 print('ys=',ys)
```

xs = [2, 3, 1]

ys = [1, 2, 3]

Note: Internally, the `reversed` and `sorted` functions perform shallow copies instead of deep copies.

Problem 29. Write the output of the following python code.

```
1 xs = [[1,2,3],[4,5,6]]
2 ys = list(reversed(xs))
3 ys.append('a')
4 ys[0].append('b')
5 print('xs=',xs)
6 print('ys=',ys)
```

xs=[[1,2,3],[4,5,6,'b']]

ys=[[4,5,6,'b'],[1,2,3],'a']

Problem 30. Write the output of the following python code.

```
1 import copy
2 xs = [[1,2,3],[4,5,6]]
3 ys = list(reversed(copy.deepcopy(xs)))
4 ys.append('a')
5 ys[0].append('b')
6 print('xs=',xs)
7 print('ys=',ys)
```

xs=[[1,2,3],[4,5,6]]

ys=[[4,5,6,'b'],[1,2,3],'a']

Note: It is fairly common to define functions within functions in python. These are called "local functions" because they are only accessible from within the outer function (just like local variables). These problems are designed to give you practice understanding how scope works in these local functions.

Problem 31. Write the output of the following python code.

```
1  x = 'a'
2  xs = [1,2,3]
3  def foo():
4      x = 'b'
5      xs = [1,2,3]
6      def bar():
7          x = 'c'
8          xs[0] = 'd'
9      bar()
10     print('x=',x)
11     print('xs=',xs)
12 foo()
```

x = b
xs = ['d',2,3]

Problem 32. Write the output of the following python code.

```
1  x = 'a'
2  xs = [1,2,3]
3  def foo():
4      global xs
5      x = 'b'
6      xs = [1,2,3]
7      def bar():
8          global x
9          x = 'c'
10         xs[0] = 'd'
11     bar()
12     print('x=',x)
13     print('xs=',xs)
14 foo()
```

y = b
xs = ['d',2,3]

Note: The next set of problems use recursion. Remember that everytime a function is called, a new frame is created that has new local variables.

Problem 33. Write the output of the following python code.

```
1 xs = []
2 def foo(i):
3     print('i=',i,'xs=',xs)
4     if i>3:
5         return
6     xs.append(i)
7     foo(i+1)
8 foo(0)
```

i = 0 xs = []
i = 1 xs = [0]
i = 2 xs = [0,1]
i = 3 xs = [0,1,2]
i = 4 xs = [0,1,2,3]

Problem 34. Write the output of the following python code.

```
1 x = 0
2 def foo(y):
3     global x
4     x += 1
5     print('x=',x,'y=',y)
6     if y==0:
7         return
8     foo(y-1)
9     print('x=',x,'y=',y)
10 foo(3)
```

x = 1 y = 3
x = 2 y = 2
x = 3 y = 1
x = 4 y = 0
x = 4 y = 1
x = 4 y = 2
x = 4 y = 3

Note: These final problems combine recusion, memory management, and scope issues on the practical problem of reversing a list. Notice that you must pay close attention to these issues or you can accidentally corrupt your input data.

Problem 35. Write the output of the following python code.

```
1  def reverse_recursive(xs):
2      def go(xs,ys):
3          if len(xs)==0:
4              return ys
5          ys.append(xs.pop())
6          return go(xs,ys)
7      return go(xs,[])
8
9  xs = [1,2,3]
10 ys = reverse_recursive(xs)
11 print('xs=',xs)
12 print('ys=',ys)
```

xs=[ ]

ys=[3,2,1]

Problem 36. Write the output of the following python code.

```
1  def reverse_recursive(xs):
2      import copy
3      xs = copy.copy(xs)
4      def go(xs,ys):
5          if len(xs)==0:
6              return ys
7          ys.append(xs.pop())
8          return go(xs,ys)
9      return go(xs,[])
10
11 xs = [1,2,3]
12 ys = reverse_recursive(xs)
13 print('xs=',xs)
14 print('ys=',ys)
```

xs=[1,2,3]

ys=[3,2,1]

Note: Combining recursion with default parameters is potentially dangerous.

Problem 37. Write the output of the following python code.

```
1  def reverse_recursive(xs,ys=[]):
2      import copy
3      xs = copy.copy(xs)
4      if len(xs)==0:
5          return ys
6      ys.append(xs.pop())
7      return reverse_recursive(xs,ys)
8
9  xs = [1,2,3]
10 ys = reverse_recursive(xs)
11 print('xs=',xs)
12 print('ys=',ys)
13
14 xs = [4,5,6]
15 ys = reverse_recursive(xs)
16 print('xs=',xs)
17 print('ys=',ys)
```

xs=[1,2,3]

ys=[3,2,1]

xs=[4,5,6]

ys=[3,2,1,6,5,4]

Problem 38. Write the output of the following python code.

```
1  def reverse_recursive(xs,ys=None):
2      import copy
3      xs = copy.copy(xs)
4      if ys is None:
5          ys = []
6      if len(xs)==0:
7          return ys
8      ys.append(xs.pop())
9      return reverse_recursive(xs,ys)
10
11 xs = [1,2,3]
12 ys = reverse_recursive(xs)
13 print('xs=',xs)
14 print('ys=',ys)
15
16 xs = [4,5,6]
17 ys = reverse_recursive(xs)
18 print('xs=',xs)
19 print('ys=',ys)
```

xs=[1,2,3]

ys=[3,2,1]

xs=[4,5,6]

ys=[6,5,4]

Note: There are many ways in python to make shallow copies of containers besides using `copy.copy`.

Problem 39. Write the output of the following python code.

```
1 xs = [1,2,3]
2 ys = xs[:]
3 ys.append('a')
4 print('xs=',xs)
5 print('ys=',ys)
```

xs=[1,2,3]

ys=[1,2,3,'a']

Problem 40. Write the output of the following python code.

```
1 xs = [1,2,3]
2 ys = list(xs)
3 ys.append('a')
4 print('xs=',xs)
5 print('ys=',ys)
```

xs=[1,2,3]

ys=[1,2,3,'a']

Note: Modifying a container that you are looping over is dangerous; strange behavior can occur depending on how you modify the container. We call these errors "silent errors" because they do not generate error messages.

Silent errors are one of the most insidious sources of programming bugs because they are so difficult to detect. Unfortunately, python is notorious for these silent errors due to its weak type system. Other languages (like C++, Java, and Haskell) have much stronger type systems that prevent certain classes of silent errors from ever occuring.

If you want to modify a container inside a for loop, you should probably instead loop over a copy of the container to prevent unintended behavior.

Problem 41. Write the output of the following python code.

```
1 xs = [1,2,3,4,5]
2 for x in xs:
3     print('x=',x)
4     xs.pop()
```

x=1
x=2
x=3

Problem 42. Write the output of the following python code.

```
1 xs = [1,2,3,4,5]
2 for x in xs:
3     print('x=',x)
4     del xs[0]
```

x=1
x=3
x=5

Problem 43. Write the output of the following python code.

```
1 xs = [1,2,3,4,5]
2 for x in list(xs):
3     print('x=',x)
4     xs.pop()
```

x = 1
x = 2
x = 3
x = 4
x = 5

Problem 44. ~~Write the output of the following python code.~~ Not all containers allow modification in the middle of a loop; the following code tries to modify a **deque** and generates an error. These error messages are good, and follow the so-called "fail loudly" principle. The inconsistency that some containers fail loudly and some fail silently is one of python's major flaws (IMNSHO). You do not have to write anything for this problem (but you should type it into the python interpreter).

```
1 from collections import deque
2 xs = deque([1,2,3,4,5])
3 for x in xs:
4     print('x=',x)
5     del xs[0]
```