CS-UY 1114: Lab 9

Shallow/Deep Copy, Lists and Tuples

You must get checked out by your lab CA prior to leaving early. **If you leave without being checked out, you will receive 0 credits for the lab.**

Restrictions

The Python structures that you use in this lab should be restricted to those you have **learned in lecture so far (topics learned in Rephactor may only be used if they have also been taught in lecture)**. Please check with your teaching assistants in case you are unsure whether something is or is not allowed!

Create a new python file for each of the following problems.

Your files should be named *lab[num]_q[num].py* similar to homework naming conventions.

Problem 1: *Shallow/Deep Copy Warm-up*

Write down what is contained in each of the lists at the end of the code snippet and as well as the print statements

```
lst = [7, 8, 9]
lst2 = lst
lst.pop()
lst2.append(10)
```

```
lst = [4, 5, 6]
```

```
def func(lst):
    lst.pop()
    lst = [7, 8, 9]
    lst.append(7)
    print("Inside func lst =", lst)
print(“Outside func lst=”, lst)
```

```
import copy
lst = [1, 2, [2, 1]]
lst_copy = copy.copy(lst)
lst[0] = 10
lst_copy[2][0] = 30
```

```
import copy
lst = [1, 2, [2, 1]]
lst_copy = copy.deepcopy(lst)
lst[0] = 10
lst_copy[2][0] = 30
```

Problem 2: *There's An Imposter Among Us…*

Write a function, **`is_impostor()`**, that will accept two parameters. The first, `information`, will be a list of a list containing any number of elements of any type. The second parameter, `corrupter_function`, will be a function. Yes, you read that correctly. In Python, it is entirely possible to pass functions as parameters. You don't need to worry about how this works, or about what kind of function `corrupter_function` will be.

The actual corrupter function (let's call it `corrupter()`) itself accepts one list object

and returns ***either a shallow or a deep copy of the original list***. Since you don't have access to the contents of `corrupter()`, you won't know if a deep or a shallow copy has been returned. That's where you come in. You will have to find a way to determine whether the list returned by `corrupter()` is a deep or a shallow copy.

If you determine that `corrupter()` has produced a deep copy, return `True`. Otherwise, return `False`.

NOTE: While it may sound like a bit of an abstract problem, think of what it means to be a **deep** copy of a list, and what it means to be a **shallow** copy of an object. The program should actually be very short and simple.

Testing

Here's a possible implementation of a `main()` function:

```
import super_secret_module


def main():
   original_list = [1, 2, 3]

   print(is_impostor(original_list, super_secret_module.corrupter))
```

The output (`True` / `False`) will vary depending on whether `corrupter()` returned a deep or shallow copy. Of course, `super_secrect_module` doesn't exist but, if you would like to use a sample corrupter function for testing purposes, feel free to download ours from the lab announcement on Brightspace. (just keep in mind that this file needs to be in the same folder as `lab10_q2.py` in order for the `main()` shown above to work).

Passing functions as parameters

If you're wondering how to use functions passed into other functions, here's a quick example:

```
# Some mathematical operations

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    return a / b

# Our calculator function doesn't care what function "operation" is.
# It just passes "a" and "b" into the function assigned to the
# `operation` parameter.

def calculator(a, b, operation):
    return operation(a, b)

def main():
    # Notice that I didn't include () after multiply
    # Just the name of the function is used in the function call.

    product = calculator(4, 5, multiply)
    print(product)

main()
```

Output:

```
20
```

Problem 3: *Nested Lists / Tuple Warm-up*

We're not done warming up! Write these following code snippets out by hand

```
my_tuple = (1, 2, 3)
my_lst = []
for i in range(3):
    my_lst.append(my_tuple[i] ** 2)
print(my_lst)
```

```
lst = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for i in range(len(lst)):
   if i % 2 == 0:
       for j in range(len(lst[i])):
           lst[i][j] *= -1

print(lst)
```

```
tup_1 = (349, 498, 425, 871)
tup_2 = (24, 73, 85, 1)
final = []
for i in range(4):
   final.append(tup_1[i] + tup_2[i])

print(tuple(final))
```

Problem 4: *Cluster Investigation*

You are given a list of tuples. Each “cluster” (tuple) contains some amount of integers within them. Your task is to write a program that not only outputs the minimum and maximum cluster by sum, but also at what index that cluster was found in.

For simplicity's sake, assume that there will be at least two tuples in the list.

For example, given the list:

```
lst =[(1, 2, 3), (5, 0, 9), (8, 1, 3, 2), (1, 8), (1, 1, 1, 1)]
```

The output of the program would be:

```
Smallest cluster is cluster 4 With a value of 4
Largest cluster is cluster 1 With a value of 14
```

Hints:

- What variables can we define to keep track of the minimum and maximum sums of the clusters?
- What method can we use to get an index of a value within a list?

Problem 5: *The Classic 2-Digit-Trim*

You’ve been told by your boss to clean up and filter the incoming lists that the company is receiving. When asking what he means by this, he explains to you that this cleanup involves **removing all duplicates** from the tuples given and **to return the tuples that only contain numbers that have 2 or more digits**.

Write a program that does exactly as the boss mentioned.

A sample list looks like this:

```
lst =[(54, 2), (34, 55, 20, 34), (222, 23, 222), (12, 12, 45, 45), (78, 6, 6), (9, 66, 34, 181)]
```

After a successful cleaning and filtering, the final returned list to give to the boss should look like this:

```
[(34, 55, 20), (222, 23), (12, 45)]
```