CS-UY 1114: Lab 8

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. 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: *List Fun!*

Write the outputs of the follwing code snippets by hand.

```
print([3 * 5] + [4 ,7])
```

```
num_lst = [1 , 57 , 15]
num_lst[2] = 25
print(num_lst * 3)
```

```
str_lst = ["good" , "better"]
str_lst.append("best")
print(str_lst)
str_lst.pop()
str_lst += str_lst
print(str_lst[::2])
```

```
lst = []
for i in range (3):
    lst.append(i ** 2)
print(lst)
```

```
my_lst = []
for i in range (3 ,6):
    for k in range (2):
        my_lst.append(i + k)
print(my_lst)
```

Problem 2: *You Are The Impostor*

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 `lab8_q4.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: *Data Analysis*

Part A: *Data Collection*

Let's start by collecting data from the user. In the file data_analysis.py write a function get_data() that will:

1. Allow the user to input numbers until they input "q" or "Q" which stops input.
2. Will return all numbers entered by the user in a list.

Below is an example of a function call to get_data()

```
def main():
    get_data()

main()
```

The following should be the output and the function should return the list of numbers.

```
Please enter a number or Q to Quit: 8
Please enter a number or Q to Quit: 7
Please enter a number or Q to Quit: 2
Please enter a number or Q to Quit: 3
Please enter a number or Q to Quit: 9
Please enter a number or Q to Quit: 1
Please enter a number or Q to Quit: Q
```

A few things to keep in mind:

- If the user only inputs a q return an empty list
- You may assume the user input will be numerical or the letter "q" or "Q"
- Consider how to handle continuous user input until a quit code is entered

Part B: *Average of Data*

Next we will be writing a function to calculate the average of the data. In the file data_analysis.py write a function calculate_average(data) that will:

1. Calculate and return the average of the data.

```
average = sum of terms / num of terms
```

Below is an example of a function call to calculate_average(data)

```python
def main():
    calculate_average([2, 3, 9, 2.3, 5, 8])

main()
```

The above call should return the following. Do not worry about small floating point differences.

```
4.883333333333334
```

A few things to keep in mind:

- If the data list is empty return None
- Consider how to iterate over the list of value while tracking the sum of terms
- Using the sum() function is not allowed for this problem

Part C: *Median of Data*

Next we will be writing a function to calculate the average of the data. In the file data_analysis.py write a function calculate_median(data) that will:

1. Calculate and return the median of the data.

Below is an example of a function call to calculate_median(data)

```python
def main():
    calculate_median([8, 7, 2, 3, 9, 1])

main()
```

The above call should return the following. Do not worry about small floating point differences.

```
5.0
```

A few things to keep in mind:

- If the data list is empty return None
- Consider how to handle lists with an even or odd length

Part D: *Maximum of Data*

Next we will be writing a function to calculate the maximum of the data. In the file data_analysis.py write a function calculate_maximum(data) that will:

1. Determine and return the maximum value of the data provided.

Below is an example of a function call to calculate_maximum(data)

```python
def main():
    calculate_maximum([8, 7, 2, 3, 9, 1])

main()
```

The above call should return the following.

```
9
```

A few things to keep in mind:

- If the data list is empty return None

Part E: *Minimum of Data*

Next we will be writing a function to calculate the minimum of the data. In the file data_analysis.py write a function calculate_minimum(data) that will:

1. Determine and return the minimum value of the data provided.

Below is an example of a function call to calculate_minimum(data)

```
def main():
    calculate_minimum([8, 7, 2, 3, 9, 1])

main()
```

The above call should return the following.

```
1
```

A few things to keep in mind:

- If the data list is empty return None

Problem 4: Now you're doing the CS student shuffle.

We'll start with another question from pedagogical programming's great canon: the list shuffle.

1. The first, `shuffle_create`, will accept one list parameter and return a **new list** with the same elements from the list that was passed in, but shuffled in any random order.
2. The second, `shuffle_in_place`, will accept one list parameter and shuffle its elements in-place (that is, it does not create a new list) in any random order.

See the sample behavior below:

```
def main():
    list_one = ["Jean Valjean", "Javert", "Fantine", "Cosette", "Marius
Pontmercy", "Eponine", "Enjolras"]
    print("ORIGINAL LIST_ONE: {}".format(list_one))

    # First function execution
    print("LIST CREATED BY SHUFFLE_CREATE:
{}\n".format(shuffle_create(list_one)))

    list_two = ["A", 0, 0, 5, 1, 3, 2]
    print("ORIGINAL LIST_TWO: {}".format(list_two))

    # Second function execution
    shuffle_in_place(list_two)
    print("LIST_TWO AFTER SHUFFLE_IN_PLACE: {}".format(list_two))

main()
```

A possible output (since the behavior is pseudo-random):

```
ORIGINAL LIST_ONE: ['Jean Valjean', 'Javert', 'Fantine', 'Cosette',
'Marius Pontmercy', 'Eponine', 'Enjolras']
LIST CREATED BY SHUFFLE_CREATE: ['Enjolras', 'Fantine', 'Jean Valjean',
'Eponine', 'Cosette', 'Javert', 'Marius Pontmercy']

ORIGINAL LIST_TWO: ['A', 0, 0, 5, 1, 3, 2]
LIST_TWO AFTER SHUFFLE_IN_PLACE: [0, 1, 2, 'A', 5, 3, 0]
```

Some explanation as to what qualifies as a "shuffle" is probably in order. For **`shuffle_create`**, it may be worth considering the use of **`random`** and list functions, such as **`randrange`**/**`randint`**, **`pop`**, and **`append`**. That is, taking elements randomly from your original list, and adding them to your new list, which of course starts empty.

For shuffling in place, it's worth remembering what in-place actually *means*. We're **not** creating a new list, but rather editing the original list. In other words, we're replacing the element at index ***x*** with the element at index ***y***. Think about how to achieve this until none of the elements from the original list are in the same place as they were before.

Restrictions

- You may **not** use the **`shuffle`** method from the **`random`** module
- You may **not** use the **`sample`** method

Problem 5: *Financial Literacy Cheat Sheet*

As means of improving your financial skills, you've been keeping track of your overall profits and losses per week. Now, as the next step in your path to becoming more financially responsible, you'd like to organize your data and get some overall info about it.

Your program, given a list of integers, will organize the data so subsequent weeks of profits are grouped together and subsequent weeks of losses are grouped together. This means your program will be creating a list of lists where each element list represents a continuous period of profit or loss. Complete this task in the function `organize_into_profits_losses(lst)` where `lst` is a list of integers. `organize_into_profits_losses(lst)` will return a list of lists.

Then, your program will **output** some general information about your spending habits the last few weeks including:

- How many times you've had a streak of only profits
- How many times you've had a streak of only losses
- Total balance at the end
- A conclusion on your financial habits

If the balance is negative or 0, the conclusion should encourage to spend less. If the balance is positive, the conclusion should encourage to maintain your current financial behavior. Complete this task in the function `def spending_statistics(lst_lsts)`. This function will *not* return anything.

The following is an example of a possible output:

```
Here are your spending habits over the last few weeks: [[1, 4], [-2], [3],
[-3, -5], [3]]
You have had 3 periods of subsequent profit.
You have had 2 periods of subsequent losses.
Total balance: 1
You're doing great! Keep it up!
```

You can use the following main definition to test your code:

```python
def main():
    weeks_lst = [1, 4, -2, 3, -3, -5, 3]
    organized_weeks_lst = organize_into_profits_losses(weeks_lst)
    print("Here are your spending habits over the last few weeks:",
organized_weeks_lst)
    spending_statistics(organized_weeks_lst)
```

Constraints: You can assume you have at least *one* week in your list.