Exercise 4: Recursion Practice

Due date: *Monday* October 30 before 10pm. (Note: we've extended the deadline by a day to give you a bit more time)

Starter code

- ex4.py
- ex4_test.py

Note: the hypothesis tests this week have more complicated inputs, which we've provided in the form of helper functions. You don't need to understand how these helper functions work, but we encourage you to use them to write your own hypothesis tests.

Task 1: More recursion on nested lists

We have seen in lecture some examples of recursive functions on nested lists. For this task, you will implement two more nested list recursive functions, one which non-mutating, and another that is mutating.

Note that these both involve taking our basic nested list recursive template and modifying it to take care of the additional complexity in both problems. In particular, as you loop through the items in a nested list (when it's a list, not an int), you'll want to **check whether each item is an integer or a list**, and do slightly different things in each case.

For `add_one`, which *mutates* nested lists, you'll want to modify our basic pattern for recursing through nested lists. In Python, iterating through a list of numbers with `for x in lst` doesn't allow you to change what numbers are stored in the list:

```
>>> lst = [1, 2, 3]
>>> for x in lst:
...     x = x * 10
>>> lst
[1, 2, 3]  # NOT [10, 20, 30]
```

So instead, you'll want to loop through list indices rather than list items:

```
if isinstance(obj, int):
    ...  # base case
else:
    ...
    for i in range(len(obj)):
        ...  # do something recursively (with obj[i])
```

Keeping track of indicies allows you to actually modify the elements of the input list!

Task 2: Family trees

In this task, you'll consider a different type of recursive structure: a *family tree*.

Consider the following situation. A person has a name and some (or no) children, each of whom is another person with a name and some (or no) children. This is modelled in the starter code using a `Person` class with two attributes, `name` and `children`. A person with no children would have an empty list stored in their `children` attribute.

A **descendant** of a person is either a child of that person, or a child of a child of that person, etc. So clearly, a person with no children has no descendants.

Your task is to complete the body of the method `count_descendants`, which returns the number of descendants of a person.

The million dollar recursive question: *How is the number of descendants of a person related to the number of descendants of their children?*

For course-related questions, please contact **csc14817f@cs.toronto.edu**.