CSC148 Lab#5, winter 2018

learning goals

In this lab you will practice tracing and implementing recursive functions where the input is a non-list or a possibly-nested list, as presented in lecture. You may want to review lecture materials.

You should work on these on your own before Thursday, and you are encouraged to then go to your lab where you can get guidance and feedback from your TA . There will be a online quiz during the second hour, in BA3175.

reading recursion

Work through the recursion exercises to practice tracing and understanding a recursive function. Remember to work from the simplest to more complex arguments and **do not** trace recursive calls that you have already traced something similar for, just replace them by the value they produce.

Notice the structure of the code. There are as many as three **if**..., **elif**..., **else**... blocks. We've used the **if** and **elif** for arguments that don't decompose into parts that can be dealt with recursively, so they directly return a value without any recursive sub-call. We use the **else** is for the general case: combines and returns the results of several recursive sub-calls. Of course it's possible to restructure these blocks.

writing recursion

Open file ex5.py and save it under a new sub-directory called **lab5**. This file provides you headers and docstrings for the functions you will implement, as well as a helper function we think you'll find useful:

gather_lists: There will be some cases where you have a list whose elements are sub-lists, and what you'd really like is to concatenate them into a single list. That's what **gather_lists** does.

Now implement the following functions.

implement list_all

Read over the header and docstring for this function, but **don't** write any implementation until you fill in the steps below:

1. One of the examples in our docstring is simple enough not to require recursion. Write out an **if** condition that checks for such cases, and then write the body of the **if** block that returns the correct thing. Include an **else**... for when the argument is **not** so easy to deal with.

2. Now suppose the function works correctly and does what the docstring claims: returns a list of the non-list values from the list or non-list it is given. If you call the function on each element in a list and it returns such a list, how will you combine the resulting list of lists into the correct output?

Fill in the implementation and see whether it works.

implement max_length

Read over the docstring but again, don't write the implementation before working through the steps below.

1. In the docstring there is an example where the argument can't be broken into smaller, similar parts. Write an **if** condition that detects such cases, and then returns the right thing. Add an **else** to deal with lists that can be broken into parts that can be dealt with recursively.

2. Now assume the function works properly when it is called on the elements of a list: for a list, it returns the maximum length of the list and all its sub-lists, or for a non-list it returns 0. If you call the function and get a correct result for each element of a list, how can you correctly combine them so the function gives the right result for the entire list? (Don't forget the length of the entire list figures into your result).

Fill in the implementation, and see whether it works.

implement list_over

Read over the docstrings, then complete the steps below:

1. There are some arguments that cannot be decomposed into parts that can be solved by this same function. Write **if** and **elif** conditions to detect these cases, and then return the correct values. Put an **else** in place for the remaining cases, where the argument can be decomposed into recursive sub-cases.

2. Assume that the function now satisfies the docstring for all the elements of a list: if the element is string of length over n, it produces a list containing that string; if the element is another string, it produces an empty list, and if it is a list of strings, it produces a list of those over length n. If the function produced the correct result for each element of a list, how would you combine all those results into the correct overall result?

Implement the function and try it out on the given examples and any others you think of.

quiz and exercise

Finish implementing all functions in ex5.py. All the functions you implement must be recursive. The quiz will likely resemble one of your solutions.

To increase your confidence in your code, we provide some unit tests.

- test_count_above_basic.py
- test_count_all_basic.py
- test_count_even_basic.py
- test_count_odd_basic.py
- test_count_even_basic.py
- test_count_odd_basic.py

Some of these tests use hypothesis. You can find out more reading the hypothesis quickstart guide