1 Critique the Code!

Take a look at these attempts to write recursive functions. Most of them have something wrong. What is wrong about them (if anything)? Check with a neighbor to see if you came to the same conclusions. Try to fix them if you can.

Question 1:

\[
\text{fact } n = n \times \text{fact } (n-1) \\
\text{fact } 0 = 1
\]

Question 2:

\[
\text{removeNegatives } (x:xs) \mid x < 0 = \text{result} \\
\mid \text{otherwise} = x : \text{result} \\
\text{where result} = \text{removeNegatives } xs
\]

Question 3:

\[
\text{reverse } [] = [] \\
\text{reverse } (x:xs) = (\text{reverse } xs) ++ [x]
\]

Question 4:

\[
\text{decList } (x:xs) = x - 1 : \text{decList } (x:xs) \\
\text{decList } [] = []
\]

2 Critique the Tail Code

Same thing, but this time these are attempts at making tail recursive code. If it’s not tail recursive, fix it so that it is.

Question 5:

\[
\text{sumList } [] a = 0 \\
\text{sumList } (x:xs) a = \text{sumList } xs \$ a + x
\]

Question 6:

\[
\text{incList } [] a = \text{reverse } a \\
\text{incList } (x:xs) a = \text{incList } xs (x + 1 : a)
\]
Question 7:

\[
\text{prodList } xx = \text{aux } xx \ 0 \\
\text{where } \text{aux } [] \ a = a \\
\text{aux } (x:xs) \ a = \text{aux } xs \ (x * a)
\]

3 Tailify the Code!

Convert these functions to tail recursion. Note, some may already be in tail form.

Question 8:

\[
\text{maxList } [x] = x \\
\text{maxList } (x:xs) = \text{max } x \ (\text{maxList } xs)
\]

Question 9:

\[
\text{fact } 0 = 1 \\
\text{fact } n = n * \text{fact } (n-1)
\]

Question 10:

\[
\text{all } p [] = \text{True} \\
\text{all } p (x:xs) \ | \ p x = \text{all } p xs \\
\ | \ \text{otherwise } = \text{False}
\]

Question 11:

\[
\text{fib } 1 = 1 \\
\text{fib } 2 = 1 \\
\text{fib } n = \text{fib } (n-1) + \text{fib } (n-2)
\]

Hint: you will need two accumulator variables, and the result will run in \(O(n)\) time.
Well Founded Induction

Malcom solve his problems with a chainsaw...
and he never has the same problem twice. — Arrogant Worms, Malcom

Hercules has a job to do. He has to slay the Hydra. The Hydra has nine heads. These are not just any heads; they are “level-9” heads. If one of them is cut off, eight level-8 heads grow to replace it. If you chop one of these, seven level-7 heads show up. This continues as you would imagine, until you get to a level-1 head. If you chop that one off, nothing else grows to take its place.

The question is this: how many head-choppings does Hercules have to perform to kill the Hydra?1

There are closed-form solutions to this, but this is a lecture about recursion, so use recursion to solve this.

We will use a list to represent the hydra’s heads.

The initial hydra head count will be represented by [9, 0, 0, 0, 0, 0, 0, 0, 0]. It shows nine heads of level nine, an no heads of the lower levels.

Write a function chop that will take a representation of the Hydra, chop of the highest level head it can get, and return the resulting hydra. Note that chop should run in $O(n)$ time. You can always, always, and forever make helper functions. Unless, of course, we tell you not to.

Sample run:

```
( chop [9,0,0,0,0,0,0,0,0], chop [0,0,2,0,0,0,0,0,0])
```

yields

```
([8,8,0,0,0,0,0,0,0], [0,0,1,6,0,0,0,0,0])
```

4 Are these too easy?

In that case, try writing a recursion in There and Back Again format. Here’s the problem statement, from Olivier Danvy.

“Computing a symbolic convolution: Given two lists $[x_1, x_2, ..., x_{n-1}, x_n]$ and $[y_1, y_2, ..., y_{n-1}, y_n]$, where $n$ is not known in advance, write a function that constructs $[(x_1, y_n), (x_2, y_{n-1}), ..., (x_{n-1}, y_2), (x_n, y_1)]$ in $n$ recursive calls and with no auxiliary list.”

---

1If you find this to be too violent, you can pretend that there’s this big puppy with nine heads....