CPS506 Lab 4 – (Tail) Recursion

Preamble

In this lab you'll practice writing recursive functions in Elixir. Each of these questions would be very easy in Java using arrays and for loops, but in functional languages recursion is the name of the game. Aside from a lecture or two in CCPS 109, you probably haven't practiced it much up to now.

Lab Description

Start by creating a module called Lab4 in an Elixir file called **lab4.ex**. Each of the functions below must be implemented in your Lab4 module. Unless otherwise specified, input lists may contain any combination of types, and be of arbitrary length. Assume nothing!

i) **`sumEven`** – Return the sum of all even integers in an input list.

ii) **`sumNum`** – Returns the sum of all *numeric* values in the list.

iii) **`tailFib`** – accepts an integer argument, **n**, and returns the nth Fibonacci number. Assume the first two Fibonacci numbers are 1 and 1. That is, tailFib(1) == 1, tailFib(2) == 1. There is no tailFib(0). If the function is called with argument less than or equal to 0, return the atom **`:error`**. Your tail-recursive implementation must avoid the double recursive call. No $O(2^n)$!

iv) **`reduce`** – Consider MyEnum.reduce from this week's lecture (you can find it in the week 4 slides). Lab4.reduce will build on this. You will add an implementation that handles the optional 3rd argument to initialize acc. For example – the following two examples, exactly as written, should work correctly:

```
Lab4.reduce([1, 2, 3], fn(x, acc) -> x+acc end)
Lab4.reduce([1, 2, 3], 10, fn(x, acc) -> x+acc end)
```

You may use helper functions if you wish, so long as the user can invoke your function as seen above.

Submission

This is a practice exercise only. There is nothing to submit.