Report for Enkla Uppgifter

Fredrik Janetzky

Spring Term 2022

Introduction

This is the first report in the course ID1019. The code is designed and written by the author of this report, Fredrik Janetzky. Only a few samples was picked out and explained, not all the solved tasks.

Recursive definitions

Product

The first task i want to present is one of the easier tasks. How to calculate the product of two numbers. The reason why i picked this task is that by solving it you realise that it can be written in many different ways. This was a good way of learning a bit of elixir. By looking on the code snippet below, this was one way way of solving it, by using different cases. We begin with setting a base case that if m == 0, the product should be zero. In the other case m can be any integer variable. In this specific case we use recursion and call the same function again but with m-1 instead. We keep on doing this until we get to the base case where m is zero.

```
def product_case(m, n) do
    case m do
        0 ->
          0
        _ ->
          (n + product_case(m-1,n))
    end
end
```

I also tried out to do the same type of solution but instead of solving it with cases I used conditions like.

```
def product_cond(m, n) do
  cond do
    m == 0 ->
```

```
      0
    true ->
      (n + product_cond(m-1,n))
  end
end
```

This solved the task in the same way but written differently.

Exponent

The next task i chose to report about is exponents. I both did the normal exponent-function but also the more effective and quicker function that look if the exponent is odd or even. This function was a bit more tricky then the last one. One problem I had that was that i first started off by using different cases, this resulted in some compiling errors. The solution to this was to use conditions instead. I first checked that if the exponent was 1 the result would be x. If the exponent was not 1, the next condition would check if the exponent was even. I did this by checking if the remainder was 0, if so i took x raised to n/2 multiplied by itself. Otherwise i would go to the else condition. Inside this condition it was working recursive. This was very learning both by figuring out the different conditions but also how the recursion was working.

```
def quick_exp(x, n) do
  cond do
    n == 1 -> x
    rem(n, 2) == 0 ->
       y = (exp(x, div(n, 2)))
       y * y
    true ->
       x * (exp(x, n - 1))
  end
end
```

List operations

Lists

The last function that i want to speak about is the $len - function$ that will count the number of elements in a list. This task was interesting because it was concerning lists. Like seen in the code snippet below I chose to build the function with two *clauses* instead. The reason why I did this was because i had some problems while defining the list with a *head* and *tail*. In a list with three elements, `[1,2,3]` the tail is then `[2,3]`. After a couple of recursive calls the list will be empty and no tail or head would exist and it occurred

problems if I defined a *head* and *tail* on an empty list. But by dividing them into two *clauses* and define the first as an empty list everything worked fine.

```
def len([]) do 0 end
def len([head | tail]) do  len(tail) + 1 end
```

The important things when you dividing a function into *clauses* is to have them in the correct order that you want them to execute in.