Stream of Primes

Lucas Larsson

Spring Term 2022

Introduction

This report is anticipated to be a short one, it was not a lot of work needed per say, rather much more variables to consider, i had to think a lot before starting with the code. The idea of a function taking in another function is fascinating enough for me, but a function that takes in another function as a parameter that takes in another function and i'm lost.

Implementation

As per usual I started by following the instructions, first function implemented was kind of easy.

```
def z(n) do
    fn -> {n, z(n + 1)} end
end
```

The function above `z/1` takes in an integer and return a tuple with the same integer and a anonymous function, that cam be used later for generating the sequence of the numbers.

```
iex(1000)> z = Primes.z(3)
#Function<3.32823058/0 in Primes.z/1>
iex(1001)> {_, z} = z.()
{3, #Function<3.32823058/0 in Primes.z/1>}
```

Calling the function `z.()` again would result in the the number 4 and another function, no more prints out are included because it is basically the same as in the instructions.

Next step was the filter function, same here just followed the instructions, the print outs were poetically helpful to clarify ambiguities, and to know that i did actually do it correctly.

.

```
def filter(fun, f) do
    {current, new_fun} = fun.()
    IO.inspect("filter" )
    IO.inspect(f)
    case rem(current, f) do
    0 ->
        filter(new_fun, f)
    _ ->
        {current, fn -> filter(new_fun, f) end}
end
```

As shown above the filter function `filter/2` takes in two parameters one is a function the other is a number, it returns a tuple of the same function and a number integer that is not dividable by `f`.

Third function is `sieve/2`, it takes in also a function and a integer, the function is a the number generator, and the number is a prime and returns a tuple with the next prime number and a function.

```
def sieve(n, p) do
    IO.inspect(:sieve )
    IO.inspect(n)
    {next_number, fun} = filter(n,p)
    {next_number, fn -> sieve(fun, next_number) end}
end
```

Last step was to join all the above mentioned functions in one function that is called `Prime/0`, and looks like see down below.

```
def primes() do
    IO.inspect(:primes)
    fn ->
        {2, fn -> sieve( z(3) end, 2) end}
    end
end
```

After the above mentioned step the function was ready to use, but it cant be used through the `Enumerable` Module, so i created another def-module and basically used the code from the instructions, this was not difficult especially since i got help from the professor in an exercise session.

Reflection

Not much to say as a reflection for this assignment i learned a lot from it, an i feel like i learned something important and relevant, i basically created a part of a library that can be used by others using a standard Module `Enumerable`, this is my favorite assignment so far, useful and difficult in the since that you need to think about the code you write but without writing a lot of it, i basically wrote less than 100 rows, which i think is perfect.