Task 1 - Simple Functions

Casper Kristiansson

2022-01-08

Introduction

The first exercise of the course Programming II consisted of creating basic elixir functions, compiling, and testing them. The author will be discussing the basic layout of initiating a new elixir project and how to compile and test that project. There will also be examples of a couple of basic functions which the author will be discussing and explaining.

Getting started

The author firstly started by installing Elixir directly from their website. After installing the programming language, it is possible to start and run commands directly to Elixirs interactive shell.

```
> iex
> x = 5
> x #=> 5
> foo = fn(x) -> x + 1 end
> foo.(2) #=> 3
```

Figure 1: Running basic Elixir commands in shell

The next step is to create a new Elixir project. The recommend way to create projects and compile them is to build an Elixir application which comes with an executable Elixir script called "mix."

```
> mix new task1
> iex -S mix
> Task1.hello() #=> Hello
```

Figure 2: Creating a Elixir Project

Creating Basic Functions

The author started off by watching the lectures on "introduction" and by reading the introduction document. The functions that the author built is solutions for the exercises in the document and a couple of exercises that was made up. The author created a total of seventeen different functions and the most interesting ones will be discussed below. To get an understanding of the language the author decided to solve some exercises in different ways either using recursion or using the Elixir Enumerable protocol. One of the bigger exercises that the author solved was implementing the insertion sort algorithm using recursion.

```
def add(x, l) do
   if Enum.member?(l, x), do: l, else: [x | l]
end
def remove(x, l) do
   Enum.filter(l, fn y -> y != x end)
end
```

Figure 3: Add and remove method solved using Enumerable protocol

Figure 3 consists of two methods, add and remove, which was solved using the Enumerable protocol. The first function "add" purpose is to add an element to a list if and only if that element does not already exist in the list. Using the member algorithm from Enumerable will return either True or False if it exists in the list.

```
def sum_list(list) do
   if length(list) == 0 do
     0
   else
     [head | tail] = list
     head + sum_list(tail)
   end
 end
```

Figure 4: Sum of a list using recursion

In Figure 4 the author created a recursion function to calculate the sum of all elements in a list. The algorithm starts off by checking if the list is empty. If it is the recursion is done and the method can return 0. Otherwise, the method divides the list by the head and tail, then using recursion the head value is added with a function call. This means that the function will continue until the recursion list is empty.