Recursive Functions

Dhruba Das

Spring Term 2024

Introduction

This assignment is a continuation of the previous assignment, Operation-on-Lists. In this assignment, we have two problems: Implementing higher order functions *map*, *reduce* and *filter*, AND re-implementing the ten functions which we implemented in the previous assignment, but using our higher order functions.

Higher order functions

In this assignment, we encounter the **three** different patterns once again, which we use to write our *map*, *reduce* and filter functions. To re-iterate, the first pattern defines functions which take a list and returns a new list containing values which were taken from the original list and/or modified. This is done so by the function *map* function, which takes a list, performs a function on that list, and returns a new list containing the resulting values:

```
def map([], _func) do [] end
  def map([h|t], func) do
    [func.(h)| map(t, func)]
  end
```

The second pattern defines functions which take a list, and an integer which is used to modify the values in the original list, and return a new list containing the modified values. This is done by the *filter* function. The *filter* function takes a list, and a function which returns true or false; then, the elements in the list which match the given criteria of the function, are returned in a new list:

```
def filter([], _func) do [] end
  def filter([h|t], func) do
    case func.(h) do
      true -> [h|filter(t, func)]
      false -> filter(t, func)
    end
  end
```

The third pattern defines a function which takes a list and returns an accumulated value obtained from all the values in the list. This is done by the *reduce* function, which takes a list, an accumulator *acc*, and a function, and then returns an accumulated value obtained by performing the given function on all the values in the list:

```
def reducel([], acc, _func) do acc end
  def reducel([h|t], acc, func) do
    reducel(t, func.(h, acc), func)
  end
```

The code shown above is *reducel*, which is the tail recursive variant of the *reduce* function.

Re-implementation

I will not be going through all 10 functions, but shown below are example-functions for each pattern:

```
def inc(list, val) do
    map(list, fn(x) -> x + val end)
  end
  ...
  def even(list) do
    filter(list, fn(x) -> rem(x, 2) == 0 end)
  end
  ...
  def lengt(list) do
    reducel(list, 0, fn(_, b) -> 1 + b end)
  end
```

Pipe operator

The instructions as to how to use the pipe operator are given to us, so I'll not be showing the original *square* function; instead shown below is the function *squarepipe*, which is the same function but is defined using pipe operators instead. The *squarepipe* function takes a list of integers and returns the sum of the square of all values less than *n*.

```
def square_pipe(list, n) do
    list |>
      filter(fn(x) -> x < n end) |>
      map(fn(x) -> x * x end) |>
      reducel(0, fn(x, y) -> x + y end)
  end
```

An example

Shown below is an a working example of the *squarepipe* function:

```
[High]
iex(9)> list
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
...
iex(21)> High.square_pipe(list, 3)
5
```