Higher-order functions in practice

futurelearn.com/courses/functional-programming-erlang/1/steps/161066

This activity gives you a chance to work with the higher order functions 'map', 'filter' and 'reduce' (or `foldr`) and to define some HoFs for yourself.

Using higher-order functions

Define the functions `doubleAll`, `evens`, and `product` using the higher-order functions `lists:map`, `lists:filter` and `lists:foldr`.

```
doubleAll([]) -> [];
doubleAll([X|Xs]) ->
    [ 2*X | doubleAll(Xs) ].

evens([]) -> [];
evens([X|Xs]) when X rem 2 == 0 ->
    [X | evens(Xs) ];
evens([_|Xs]) ->
    evens(Xs).

product([]) -> 1;
product([X|Xs]) -> X * product(Xs).
```

Zipping

a) Define a function `zip/2` that "zips together" pairs of elements from two lists like this:

```
zip([1,3,5,7], [2,4]) = [ {1,2}, {3,4} ]
```

where you can see that the elements from the longer list are lost.

b) Define a function `zip_with/3` that "zips together" pairs of elements from two lists using the function in the first argument, like this:

```
zip_with(fun(X,Y) -> X+Y end, [1,3,5,7], [2,4]) = [ 3, 7 ]
```

c) Re-define the function `zip_with/3` using `zip` and `lists:map`.

d) Re-define `zip/2` using `zip_with/3`.

We'll go over the zip question in the next step, but do please use the comments on this step to discuss your solutions to all the questions here.