ASSIGNMENT 3: HIGHER ORDER FUNCTIONS

1. **Implement the for loop construct in Erlang as a function. DO NOT use any other variables other than the parameters `I`, `Pred`, `Update` and `Body`.**

   ```
   for(I, Pred, Update, Body) ->
   ```

   Example usage:

   A for loop to print 0 to 9:

   ```
   for(0, fun(X) -> X < 10 end, fun(X) -> X+1 end, fun(X) ->
       io:format("~i~n", [X]) end).
   ```

   Notice that Pred, Update and Body are all unary functions.

2. It is sometimes useful to know *where in a list* a certain element occurs, if it occurs at all. **Program the function `index-in-list-by-predicate` which searches for a given element.** The comparison between the given element and the elements in the list is controlled by a comparison parameter to index-in-list-by-predicate. The function should return the list position of the match (first element is number 0), or the atom `not_found` if no match is found.

   Example usage:

   ```
   index-in-list-by-predicate([20,10,30,40], 10, f(X,Y) ->
                               X =:= Y end).
   % should return 1

   index-in-list-by-predicate([20,10,30,40], 50, f(X,Y) ->
                               X =:= Y end).
   % should return not_found

   index-in-list-by-predicate([20,10,3,40], 20, f(X,Y) ->
                               X-Y =:= 17 end).
   % should return 2
   ```

3. The mathematical quantifiers *for all* and *there exists* are well-known. **Implement the following three Boolean functions (should return either true or false):**
   The function `for-all(List, Pred)` is supposed to check if all elements in the list `List` satisfy the predicate `Pred`.

The function `there-exists(List, Pred)` is supposed to check if one or more elements in the list `List` satisfy the predicate `Pred`.

Finally, the function `there-exists-1(List, Pred)` is supposed to check if exactly on element in the list `List` satisfies `Pred`.

4. The functions `foldr`, and `foldl` are fold equivalents, except that in `foldr`, we start processing from end of the list (hence fold right, or `foldr`), and in `foldl`, we start processing from beginning of the list (hence fold left, or foldl).

   Implement `foldr`, `foldl`. Then, implement `filterr` using `foldr`, and `filterl` using `foldl`.

5. The function `remdups` removes adjacent duplicates from a list. For example,
   `remdups ([1, 2, 2, 3, 3, 3, 1, 1]).`
   % should return [1, 2, 3, 1].

   Define `remdups` using `foldr` . Give another definition using `foldl`.

Unrestricted