Lists and Higher-Order Functions

Basics

Exercice 1 (iter)
Write the function `iter` which applies a function to each element of the list.

```
val iter : ('a -> unit) -> 'a list -> unit = <fun>

# iter (print_int) [1; 2; 3; 4];;
1234- : unit = ()
```

Exercice 2 (map)
Write the function `map` which applies a function to each element of the list and return a list of these results

```
val map : ('a -> 'b) -> 'a list -> 'b list = <fun>

# map (function x -> x * x) [1; 2; 3; 4];;
- : int list = [1; 4; 9; 16]
```

Exercice 3 (iteri)
Write the function `iter`, same as `iter`, but the function is applied to the index of the element as first argument (counting from 0), and the element itself as second argument.

```
val iteri : (int -> 'a -> unit) -> 'a list -> unit = <fun>

# iteri (function n -> function x -> if n mod 2 = 0 then print_int x) [1;2;3;4;5] ;;
135- : unit = ()
```

Exercice 4 (mapi)
Write the function `mapi`, same as `map`, but the function is applied to the index of the element as first argument (counting from 0), and the element itself as second argument.

```
val mapi : (int -> 'a -> 'b) -> 'a list -> 'b list = <fun>

# mapi (function n -> function x -> power(x,n)) [3;3;3;3;3];;
- : int list = [1; 3; 9; 27; 81]
```

Exercice 5 (for_all)
Write the function `for_all` which checks if all the elements of the list satisfy the predicate.

```
val for_all : ('a -> bool) -> 'a list -> bool = <fun>

# for_all (function x -> x = 0) [0; 0; 0; 0];;
- : bool = true
```

Exercice 6 (exists)
Write the function `exists` which checks if at least one element of the list satisfies the predicate.

```
val exists : ('a -> bool) -> 'a list -> bool = <fun>

# exists (function x -> x = 0) [3; 4; 0; 6];;
- : bool = true
```

Game of Life

1. Rewrite the `draw_cell` function from the previous practical, with new arguments : the cell (an integer), its coordinates (x,y) on the board, its size and a function which give the proper color according to the cell's state.

   ```
   val draw_cell : int -> int * int -> int -> (int -> Graphics.color) -> unit = <fun>
   ```

2. Rewrite the `draw_board` function from the previous practical using only the `iteri` and `draw_cell` functions. The `draw_board` function can not be itself recursive.

   ```
   val draw_board : int list list -> int -> (int -> Graphics.color) -> unit = <fun>
   ```

3. Write the `remaining` function which determines if there is at least one cell respecting the rule provided in the argument. The `remaining` function can not be itself recursive.

   ```
   val remaining : ('a -> bool) -> 'a list list -> bool = <fun>

   # remaining (function x -> x > 0) board ;;
   - : bool = true
   ```

4. Write the `map_board` function which will apply the provided function to each cell of the board if it has a value greater than 0.The `map_board` function can not be itself recursive.

   ```
   val map_board : 'a list list -> ('a -> 'b) -> 'b list list = <fun>
   ```

5. Write the `mapi_board` function which will apply the provided game's rules to each cell of the board. The `mapi_board` function can not be itself recursive. The function will take the following arguments :
   – the board
   – the rule on how to get the neighborhood of the current cell, as a list
   – the rule on how to count "actives" cells in a list
   – the rule on how the cell will behave according to the neighborhood "active" value

   ```
   val mapi_board : 'a list list -> (int -> int -> 'a list list -> 'b) ->
      ('b -> 'c) -> ('c -> 'd) -> ('d list list) = <fun>
   ```

6. Write the `game_of_life` function which takes the following arguments : the board's size, the cell's size. It will initialize the game's board properly, and will repeatedly draw the board, applies the rules until no cell remains.

   ```
   val game_of_life : int -> int -> unit = <fun>
   ```

Add-ons

Write the function `real_life` which takes all the rules and functions of the game (from display to check cell's life) in arguments, to make the game 100% customizable.

```
val real_life : '? -> unit = <fun>
```

Modify the game's rules to avoid killing the cells directly, but make them loose or gain life points according to the environment. It might be interesting to see a display reflecting this.