OCAML PROGRAMMING

Remove sequential duplicates

```
# let rec destutter list =
    match list with
    | [] -> []
    | [hd] -> [hd]
    | hd :: hd' :: tl ->
      if hd = hd' then destutter (hd' :: tl)
      else hd :: destutter (hd' :: tl)
  ;;
val destutter : 'a list -> 'a list = <fun>
```

Remove sequential duplicates

```
# let rec destutter list =
    match list with
    | [] -> []
    | [hd] -> [hd]
    | hd :: hd' :: tl ->
      if hd = hd' then destutter (hd' :: tl)
      else hd :: destutter (hd' :: tl)
  ;;
val destutter : 'a list -> 'a list = <fun>
```

[hd] -> [hd] allocates a new list element

More Efficient Solution

```
# let rec destutter = function
    | [] as l -> l
    | [_] as l -> l
    | hd :: (hd' :: _ as tl) ->
      if hd = hd' then destutter tl
      else hd :: destutter tl
  ;;
val destutter : 'a list -> 'a list = <fun>
```

The as pattern allowws us to declare a name for the thing matched by a pattern or subpattern.

Even More Efficient

```
# let rec destutter = function
    | [] | [_] as l -> l
    | hd :: (hd' :: _ as tl) when hd = hd' -> destutter tl
    | hd :: tl -> hd :: destutter tl
  ;;
val destutter : 'a list -> 'a list = <fun>
```

Ex1

- Find out whether a list is a palindrome
- # is_palindrome ["x" ; "a" ; "m" ; "a" ; "x"];;
  - : bool = true
- # not (is_palindrome ["a" ; "b"]);;
  - : bool = true

Solution

```
# let rev list =
    let rec aux acc = function
      | [] -> acc
      | h::t -> aux (h::acc) t in
    aux [] list;;
val rev : 'a list -> 'a list = <fun>
```

```
   let is_palindrome list =
          list = rev list;;
```

Ex2

- Insert an element at a cartain position.
- # insert_at "alfa" 1 ["a";"b";"c";"d"];;
  - : string list = ["a"; "alfa"; "b"; "c"; "d"]
- # insert_at "alfa" 3 ["a";"b";"c";"d"];;
  - : string list = ["a"; "b"; "c"; "alfa"; "d"]
- # insert_at "alfa" 4 ["a";"b";"c";"d"];;
  - : string list = ["a"; "b"; "c"; "d"; "alfa"]

Solution

- let rec insert_at x n = function
  | [] -> [x]
  | h :: t as l -> if n = 0 then x :: l
  else h :: insert_at x (n-1) t;;

Ex3

- Create a list containing all integers within a given range.
  - If first argument is smaller than second, produce a list in decreasing order.

- # range 4 9;;
  - : int list = [4; 5; 6; 7; 8; 9]
- # range 9 4;;
  - : int list = [9; 8; 7; 6; 5; 4]

Sol 3

```
# let range a b =
    let rec aux a b =
      if a > b then [] else a :: aux (a+1) b  in
    if a > b then List.rev (aux b a) else aux a b;;
val range : int -> int -> int list = <fun>
```

Ex4

- Rotate a list N places to the left.

- # rotate ["a"; "b"; "c"; "d"; "e"; "f"; "g"; "h"] 3;;
  - : string list = ["d"; "e"; "f"; "g"; "h"; "a"; "b"; "c"]
- # rotate ["a"; "b"; "c"; "d"; "e"; "f"; "g"; "h"] (-2);;
  - : string list = ["g"; "h"; "a"; "b"; "c"; "d"; "e"; "f"]

Sol 4

```
# let split list n =
    let rec aux i acc = function
      | [] -> List.rev acc, []
      | h :: t as l -> if i = 0 then List.rev acc, l
                       else aux (i-1) (h :: acc) t  in
    aux n [] list

  let rotate list n =
    let len = List.length list in
    (* Compute a rotation value between 0 and
len-1 *)
    let n = if len = 0 then 0 else (n mod len + len)
mod len in
    if n = 0 then list
    else let a, b = split list n in b @ a;;
val split : 'a list -> int -> 'a list * 'a list = <fun>
val rotate : 'a list -> int -> 'a list = <fun>
```

Ex5

- Eliminate consecutive duplicates of list elements
- # compress
  ["a";"a";"a";"a";"b";"c";"c";"a";"a";"d";"e";"e";
  "e";"e"];;
  - : string list = ["a"; "b"; "c"; "a"; "d"; "e"]

Sol 5

```
# let rec compress = function
    | a :: (b :: _ as t) -> if a = b then compress t
                            else a :: compress t
    | smaller -> smaller;;
val compress : 'a list -> 'a list = <fun>
```

Ex 6

Pack consecutive duplicates of list elements
into sublists

```
# pack
["a";"a";"a";"a";"b";"c";"c";"a";"a";"d";"d";"e";"e";"e";"e"];;
- : string list list =
[["a"; "a"; "a"; "a"]; ["b"]; ["c"; "c"]; ["a"; "a"]; ["d"; "d"];
 ["e"; "e"; "e"; "e"]]
```

Sol 6

```
 let pack list =
   let rec aux current acc = function
     | [] -> []    (* Can only be reached if original
list is empty *)
     | [x] -> (x :: current) :: acc
     | a :: (b :: _ as t) ->
       if a = b then aux (a :: current) acc t
       else aux [] ((a :: current) :: acc) t  in
   List.rev (aux [] [] list);;
val pack : 'a list -> 'a list list = <fun>
```

Es 7

- Given two indices, i and k, the slice is the list containing the elements between the i'th and k'th element of the original list (both limits included). Start counting the elements with 0 (this is the way the List module numbers elements).

- # slice ["a";"b";"c";"d";"e";"f";"g";"h";"i";"j"] 2 6;;
  - : string list = ["c"; "d"; "e"; "f"; "g"]

Sol 7

```
# let slice list i k =
    let rec take n = function
      | [] -> []
      | h :: t -> if n = 0 then [] else h :: take (n-1) t
    in
    let rec drop n = function
      | [] -> []
      | h :: t as l -> if n = 0 then l else drop (n-1) t
    in
    take (k - i + 1) (drop i list);;
val slice : 'a list -> int -> int -> 'a list = <fun>
```