
OCAML
PROGRAMMING

TAKE #1 (EASY)

Find out whether a list is a palindrome.

INPUT-OUTPUT (EXPECTYED) BEHAVIOUR

```
# is_palindrome ["x"; "a"; "m"; "a"; "x"];; - : bool = true
# not (is_palindrome ["a"; "b"]);; - : bool = true
```

TAKE #1 (EASY): SOLUTION

Find out whether a list is a palindrome.

INPUT-OUTPUT (EXPECTYED) BEHAVIOUR

 # is_palindrome ["x"; "a"; "m"; "a"; "x"];; - : bool = true
 # not (is_palindrome ["a"; "b"]);; - : bool = true

```
# 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;;
val is_palindrome : 'a list -> bool = <fun>
```

TAKE #1

```
# is_palindrome ["x"; "a"; "m"; "a"; "x"];;
- : bool = true
# not (is_palindrome ["a"; "b"]);;
- : bool = true
# is_palindrome [1;4;4;1];;
- : bool = true
#
```

TAKE #1

ALTERNATIVE SOLUTION (BETTER)

```
# let is_palindrome list =
(* Use the built-in List.rev *)
list = List.rev list;;

val is_palindrome : 'a list -> bool = <fun>
```

TAKE #2 (EASY)

Remove the n'th element from a list.

```
# remove_at 1 ["a"; "b"; "c"; "d"];;
- : string list = ["a"; "c"; "d"]
```

TAKE #2 SOLUTION

```
# let rec remove_at n = function
    | [] -> []
    | h :: t -> if n = 0 then t else h :: remove_at (n - 1) t;;

val remove_at : int -> 'a list -> 'a list = <fun>

# remove_at 1 ["a"; "b"; "c"; "d"];;
- : string list = ["a"; "c"; "d"]


# remove_at 2 [1; 2; 3; 4];;
- : int list = [1; 2; 4]
```

TAKE #3 (INTERMEDIATE)

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"]]
```

TAKE 3 SOLUTION

```
#  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>
# 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"]]
# pack [1;1];;
- : int list list = [[1; 1]]
# pack [1;2;1];;
- : int list list = [[1]; [2]; [1]]
#
```

TAKE #4

We suppose that a list contains elements that are lists themselves. The objective is to sort the elements of this list according to their length. E.g. short lists first, longer lists later, or vice versa.