40 Zadań Hardcore Practice 🔥

📋 FOLD NA LISTACH - 10 zadań

(Użyj fold_left LUB fold_right - Twój wybór!)

1. Reverse z akumulatorem

ocaml

```
reverse_fold : 'a list -> 'a list
(* [1;2;3] -> [3;2;1] *)
```

2. Map przez fold

ocaml

```
map_fold : ('a -> 'b) -> 'a list -> 'b list
(* map_fold (fun x -> x*2) [1;2;3] -> [2;4;6] *)
```

3. Filter przez fold

ocaml

```
filter_fold : ('a -> bool) -> 'a list -> 'a list
(* filter_fold (fun x -> x > 2) [1;2;3;4] -> [3;4] *)
```

4. Flatten przez fold

ocaml

```
flatten_fold : 'a list list -> 'a list
(* [[1;2]; [3]; [4;5]] -> [1;2;3;4;5] *)
```

5. Group consecutive

ocaml

```
group_consecutive : 'a list -> 'a list list
(* [1;1;2;2;2;3] -> [[1;1]; [2;2;2]; [3]] *)
```

6. All pairs

ocaml

```
all_pairs : 'a list -> ('a * 'a) list
(* [1;2;3] -> [(1,2); (1,3); (2,3)] - wszystkie pary bez powtórzeń *)
```

7. Split at predicate

ocaml

```
split_at : ('a -> bool) -> 'a list -> 'a list * 'a list
(* split_at (fun x -> x > 3) [1;2;4;5;2] -> ([1;2], [4;5;2]) *)
```

8. Intersperse

ocaml

```
intersperse : 'a -> 'a list -> 'a list
(* intersperse 0 [1;2;3] -> [1;0;2;0;3] *)
```

9. Zip with

ocaml

```
zip_with : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list
(* zip_with (+) [1;2;3] [4;5;6] -> [5;7;9] *)
```

10. Scan (prefix sums)

ocaml

```
scan : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a list
(* scan (+) 0 [1;2;3;4] -> [0;1;3;6;10] *)
```

🌳 FOLD NA DRZEWIE - 10 zadań

ocaml

```
type 'a tree = Leaf | Node of 'a tree * 'a * 'a tree
```

1. Tree sum

ocaml

```
tree_sum : int tree -> int
```

2. Tree map

ocaml

```
tree_map : ('a -> 'b) -> 'a tree -> 'b tree
```

3. **Tree filter** (usuwa węzły niespełniające predykat)

ocaml

```
tree_filter : ('a -> bool) -> 'a tree -> 'a tree
```

4. **Tree paths** (wszystkie ścieżki od root do leaf)

ocaml

```
tree_paths : 'a tree -> 'a list list
(* Node(Leaf, 1, Node(Leaf, 2, Leaf)) -> [[1;2]] *)
```

5. **Tree level lists** (elementy na każdym poziomie)

ocaml

```
tree_levels : 'a tree -> 'a list list
(* poziom 0: [root], poziom 1: [dzieci root], etc. *)
```

6. **Tree zigzag** (poziomy naprzemiennie L→R i R→L)

ocaml

```
tree_zigzag : 'a tree -> 'a list
```

7. **Tree max path sum**

ocaml

```
max_path_sum : int tree -> int
(* maksymalna suma na ścieżce root→leaf *)
```

8. **Tree to dot** (reprezentacja graficzna)

ocaml

```
tree_to_dot : 'a tree -> string
(* generuje format DOT dla graphviz *)
```

9. **Tree subtrees** (wszystkie poddrzewa)

ocaml

```
all_subtrees : 'a tree -> 'a tree list
```

10. Tree is balanced

ocaml

```
is_balanced : 'a tree -> bool
(* różnica wysokości poddrzew ≤ 1 w każdym węźle *)
```

🎯 TYPOWANIE - 10 zadań (średnie/trudne)

Podaj najogólniejszy typ lub "BŁĄD TYPÓW" + uzasadnienie

1.

ocaml

```
fun f g h x -> f (g x) (h x)
```

2.

ocaml

```
fun f x -> f (f x) (f x x)
```

3.

ocaml

```
let rec fix f = f (fix f)
```

4.

ocaml

```
fun f -> fun x -> fun y -> f y x
```

5.

ocaml

```
fun p f g x -> if p x then f x else g x
```

6.

```ocaml
fun f g -> fun x -> fun y -> f (g x y) (g y x)
```

7.

```ocaml
let rec map f xs =
  match xs with
  | [] -> []
  | x :: xs' -> f x :: map f xs'
```

8.

```ocaml
fun f -> (fun x -> f (x x)) (fun x -> f (x x))
```

9.

```ocaml
fun cmp lst ->
  let rec sorted = function
    | [] | [_] -> true
    | x :: y :: xs -> cmp x y && sorted (y :: xs)
  in sorted lst
```

10.

```ocaml
fun fold f acc xs ->
  let rec loop acc = function
    | [] -> acc
    | x :: xs -> loop (f acc x) xs
  in loop acc xs
```

🔍 ZMIENNE WOLNE/ZWIĄZANE - 10 zadań

Dla każdego wyrażenia wskaż wszystkie wystąpienia zmiennych i sklasyfikuj je jako wolne/związane

1.

ocaml

```ocaml
let x = y + 1 in x * z
```

2.

ocaml

```ocaml
fun f -> let x = f y in x + f x
```

3.

ocaml

```ocaml
let f = fun x -> fun y -> x + y + z in f a b
```

4.

ocaml

```ocaml
let rec fact n = if n = 0 then 1 else n * fact (n-1)
```

5.

ocaml

```ocaml
let x = 5 in
fun x -> let y = x + z in fun z -> x + y + z
```

6.

ocaml

```ocaml
(fun x -> x + y) (let y = 3 in y + 1)
```

7.

ocaml

```ocaml
let f = fun g -> fun x -> g (g x) in
let h = fun y -> y + 1 in
f h z
```

8.

```ocaml
let x = 1 and y = x + 2 in x + y
```

9.

```ocaml
fun f ->
  let rec g x = f x + g (x-1) in
  g 5
```

10.

```ocaml
let outer = fun x ->
  let inner = fun x -> x + y in
  fun y -> inner x + y
in outer a b
```

🚀 RULES OF ENGAGEMENT:

- **Nie podawaj odpowiedzi** - tylko zadania!
- **Robię w swojim tempie** - nie spiesz się
- **Pytaj się** jak coś niejasne
- **Testuj w OCaml** - sprawdzaj czy kompiluje się

WHEN READY - wyślij swoje rozwiązania i zobaczymy co potrzebujesz dopracować! 🔥

UNLEASH THE CODING BEAST! 💪⚡