

Introduktion til Programmering og Problemløsning (PoP)

List Module

Jon Sparring
Department of Computer Science
2021/10/07

UNIVERSITY OF COPENHAGEN



Modulet List

Modulet `List` indeholder en lang række operationer på lister.

```
// List creation
```

```
List.init: m:int -> f:(int -> 'T) -> 'T list
```

```
// Argumentet er index
```

```
let lst = List.init 10 (fun i -> i)
```

```
val it: int list = [0; 1; 2; 3; 4; 5; 6; 7; 8; 9]
```

```
// Ikke triviell funktion
```

```
let odd i = (i%2 = 1)
```

```
let lst = List.init 10 odd
```

```
val lst: bool list =
```

```
[false; true; false; true; false; true; false; true; false; true]
```

```
// Anonyme funktioner
```

```
let lst = List.init 10 (fun i -> (i%2 = 1))
```

```
val it: int list = [0; 1; 2; 3; 4; 5; 6; 7; 8; 9]
```

Modulet List: init, hvad sker der

```
// List creation
```

```
List.init: m:int -> f:(int -> 'T) -> 'T list
```

```
// A similar function
```

```
let rec init ((i,m): int*int) (f: int -> 'T) (lst: 'T list) : 'T list =  
  if i >= m then []  
  else (f i)::(init (i+1,m) f lst)
```

```
let lst = init (0,4) (fun i -> i*i) []
```

```
val init: int * int -> f: (int -> 'T) -> lst: 'T list -> 'T list
```

```
val lst: int list = [0; 1; 4; 9]
```

Modulet List: Map

```
// list transformation
```

```
List.map: f:('T -> 'U) -> lst:'T list -> 'U list
```

```
// [1...10] -> [log 1.0; log 2.0; ...; log 10.0]
```

```
let lst = List.map (fun i -> log (float i)) [1..10]
```

```
val lst: float list =
```

```
[0.0; 0.6931471806; 1.098612289; 1.386294361; 1.609437912; 1.791759469;  
 1.945910149; 2.079441542; 2.197224577; 2.302585093]
```

```
// ['a'..'z'] -> [ascii 'a'.. ascii 'z']
```

```
let lst = List.map (fun i -> int i) ['a'..'z']
```

```
val lst: int list =
```

```
[97; 98; 99; 100; 101; 102; 103; 104; 105; 106; 107; 108; 109; 110; 111; 112;  
 113; 114; 115; 116; 117; 118; 119; 120; 121; 122]
```

```
// (fun i -> f i) = f
```

```
let lst = List.map int ['a'..'z']
```

```
val lst: int list =
```

```
[97; 98; 99; 100; 101; 102; 103; 104; 105; 106; 107; 108; 109; 110; 111; 112;  
 113; 114; 115; 116; 117; 118; 119; 120; 121; 122]
```

Modulet List: map, hvad sker der

```
// list transformation
```

```
List.map: f:('T -> 'U) -> lst:'T list -> 'U list
```

```
// A similar function
```

```
let rec map (f: 'T -> 'U) (lst: 'T list) : 'U list =
```

```
  match lst with
```

```
    [] -> []
```

```
    | elm::rst -> f elm :: map f rst
```

```
let lst = map (fun i -> i*i) [0..3]
```

```
val map: f: ('T -> 'U) -> lst: 'T list -> 'U list
```

```
val lst: int list = [0; 1; 4; 9]
```

Modulet List: Fold

```
// list transformation
```

```
List.fold: f:('S -> 'T -> 'S) -> acc:'S -> lst:'T list -> 'S
```

$$f(\dots(f(f \text{acc}_0 \text{lst}[0]) \text{lst}[1]) \dots) \text{lst}[n-1]$$

$$\text{acc}_1$$

```
// sum
```

```
List.fold (fun acc elm -> acc + elm) 0 [1..3]
```

```
val it: int = 6
```

$$((0 + 1) + 2) + 3 = 6$$

```
// max
```

```
List.fold (fun acc elm -> max acc elm) 0 [1..3]
```

```
val it: int = 3
```

```
// string concatenate
```

```
List.fold (fun acc elm -> acc + string elm) "" ['a'; 'e'; 'i'; 'o'; 'u'; 'y']
```

```
val it: string = "aeiouy"
```

```
// list concatenate
```

```
List.fold (fun acc elm -> acc @ elm) [] [[1..3]; [3..-1..1]]
```

```
val it: int list = [1; 2; 3; 3; 2; 1]
```

Modulet List: fold, hvad sker der

```
// list transformation
```

```
List.fold: f: ('S -> 'T -> 'S) -> acc: 'S -> lst: 'T list -> 'S
```

```
// A similar function
```

```
let rec fold (f: 'S -> 'T -> 'S) (acc: 'S) (lst: 'T list) : 'S =  
  match lst with  
  | [] -> acc  
  | elm::rst -> fold f (f acc elm) rst
```

```
let lst = fold (fun acc i -> (i*i)::acc) [] [3..-1..0]
```

```
val fold: f: ('S -> 'T -> 'S) -> acc: 'S -> lst: 'T list -> 'S
```

```
val lst: int list = [0; 1; 4; 9]
```

Resumé

I denne video har du hørt om:

- Modulet **List** og **List.map** og **List.fold**