Programmering og Problemløsning

5.1: Lister

Repetition af Nøglekoncepter

- Moduler og biblioteker
- Black- og white-box testing
- Højere-ordens funktioner
- Anonyme funktioner

- Closures
- Håndkøring

Navn -> (input, krop, virkefeltets værdier)

Leksikografisk

```
E0:
  testScope -> ((x), testScope-body, ())
  linje 6: testScope 2.0 -> ?   6.0
  stdout -> "6.0"
  return -> ()
E1: ((x -> 2.0), testScope-body, ())
  a -> 3.0
  f -> ((z), a * z, (a -> 3.0))
  a -> 4.0
  f 2.0 -> ?   6.0
  return -> 6.0

E2: ((z -> 2.0), a * z, (a -> 3.0))
  return -> 6.0
```

Dynamisk

```
E0:
  testScope -> ((x), testScope-body, ())
  linje 6: testScope 2.0 -> ?   8.0
  stdout -> "8.0"
  return -> ()
E1: ((x -> 2.0), testScope-body, ())
  a -> alpha
  f -> ((z), a * z, (a -> alpha))
  f 2.0 -> ?   8.0
  return -> 8.0

E2: ((z -> 2.0), a * z, (a -> alpha))
  return -> 8.0
```

Linje	Navn	Værdi
2	alpha	3.0
4	alpha	4.0

Ting på lister

Tupler:

```
> let a = (3, "tre");;
val a : int * string = (3, "tre")
```

Forskellige typer og størrelse fastlagt på definitionstidspunkt

Strenge:

```
> "hej med jer".[4..6];;
val it : string = "med"
```

Samme type (char), og operatorer til indicering og sammensætning

Lister:

> let a = ['h'; 'e'; 'j'];;
val a : char list = ['h'; 'e'; 'j']

Ligesom strenge, kan holde andre elementer af same type og er **ikke mutérbar**

- Den tome liste:
  > let a = [];;
  val a : 'a list

  Generisk type

- Indicering
  > ['h'; 'e'; 'j'].[1..];;
  val it : char list = ['e'; 'j']

  Slicing (som strenge)

- Samenligning

  > [2; 3; 5] > [2; 2; 6];;
  val it : bool = true

  'Alfabetisk' sammenligning

- Append (@)
  > ['h'; 'e'; 'j'] @ [' '; 'm'; 'e'; 'd'];;
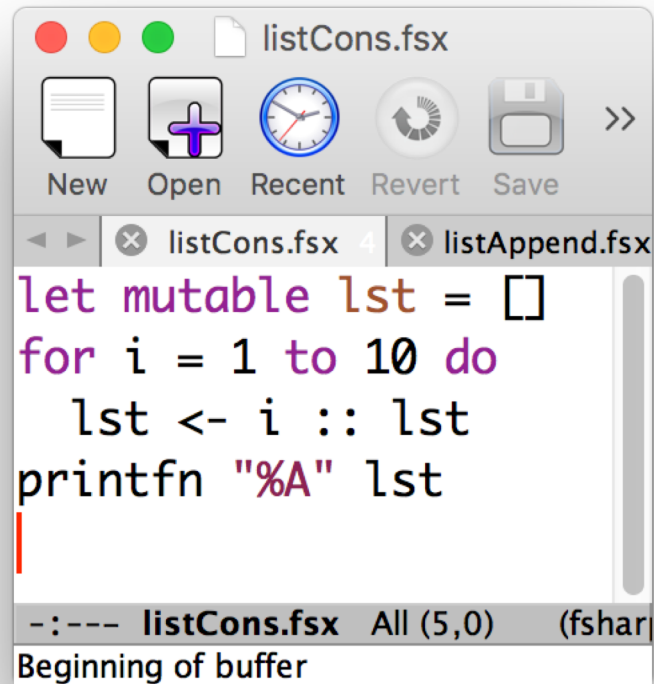 val it : char list = ['h'; 'e'; 'j'; ' '; 'm'; 'e'; 'd']

  Sammensætning af 2 lister

- Cons (::)
  > 'h' :: ['e'; 'j'];;
  val it : char list = ['h'; 'e'; 'j']

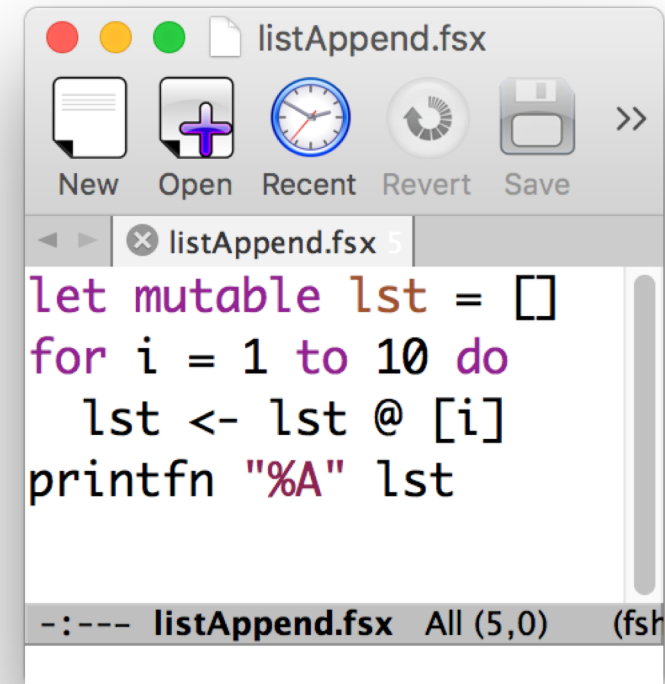 Sætte et element forest i en liste (prepend)

Hvilket program skriver "[1; 2; 3; 4; 5; 6; 7; 8; 9; 10]" på skærmen

listCons.fsx

```
let mutable lst = []
for i = 1 to 10 do
  lst <- i :: lst
printfn "%A" lst
```

listAppend.fsx

```
let mutable lst = []
for i = 1 to 10 do
  lst <- lst @ [i]
printfn "%A" lst
```

https://tinyurl.com/ycogq4ur

Køretid

listAppendLarge.fsx

```
let mutable lst = []
for i = 1 to 40000 do
  lst <- lst @ [i]
```
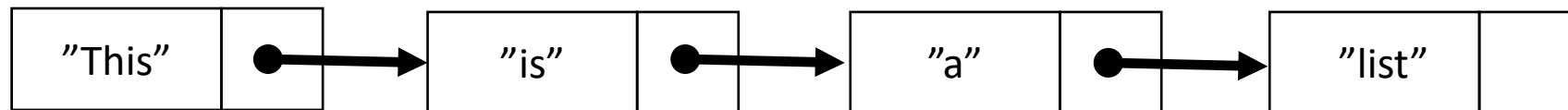
listAppendLarge.fsx && time mono listAppendLarge.exe

listConsLarge.fsx

```
let mutable lst = []
for i = 1 to 40000 do
  lst <- i :: lst
```

listConsLarge.fsx && time mono listConsLarge.exe

Lister er linkede lister:

Listeværdier og -funktioner

- Oprettelse af lister

  ```
  > let a = List.init 5 (fun i -> i)
  - let b = [0..4];;
  val a : int list = [0; 1; 2; 3; 4]
  val b : int list = [0; 1; 2; 3; 4]
  ```

- Hoved og hale

  ```
  > let a = [0..5]
  - printfn "%A %A" a.Head a.Tail;;
  0 [1; 2; 3; 4; 5]
  val a : int list = [0; 1; 2; 3; 4; 5]
  val it : unit = ()
  ```

- Gennemløb af lister

  ```
  > let a = List.init 5 (fun i -> pown 2 i)
  - for i = 0 to a.Length - 1 do
  -   printf "%d " a.[i]
  - printfn "";;
  1 2 4 8 16
  val a : int list = [1; 2; 4; 8; 16]
  val it : unit = ()
  ```

- Bedre gennemløb af lister

  ```
  > let a = List.init 5 (fun i -> pown 2 i)
  - List.iter (fun e -> printf "%d " e) a
  - printfn "";;
  1 2 4 8 16
  val a : int list = [1; 2; 4; 8; 16]
  val it : unit = ()
  ```

Map og Fold

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

  lst = [1;2;3] => [f 1; f 2; f 3]

- Map'e funktioner på lister

  > let a = [0..5]
  - List.map (fun e -> e * e) a;;
  val a : int list = [0; 1; 2; 3; 4; 5]
  val it : int list = [0; 1; 4; 9; 16; 25]

List.fold: f:('State -> 'T -> 'State) -> elm:'State -> lst:'T list -> 'State

  lst = [1;2;3] og elm = 0 => f (f (f 0 1) 2) 3

- Folde en liste sammen: sum

  > let a = [0..5]
  - let sum acc elm = acc + elm
  - List.fold sum 0 a;;
  val a : int list = [0; 1; 2; 3; 4; 5]
  val sum : acc:int -> elm:int -> int
  val it : int = 15

- Folde en liste sammen: con

  > let a = [0..5]
  - let app con elm = acc + (string elm)
  - List.fold con "" a;;
  val a : int list = [0; 1; 2; 3; 4; 5]
  val app : acc:string -> elm:int -> string
  val it : string = "012345"

- Vende en liste om: rev

  > let a = [0..5]
  - let rev acc elm = elm :: acc
  - List.fold rev [] a;;
  val a : int list = [0; 1; 2; 3; 4; 5]
  val rev : acc:'a list -> elm:'a -> 'a list
  val it : int list = [5; 4; 3; 2; 1; 0]

Arrays:

```
> let a = [|'h'; 'e'; 'j'|];;
val a : char [] = [|'h'; 'e'; 'j'|]
```

Ligesom lister, men hverken append eller cons operaterbare, og **mutérbar**

- Den tome array:
```
> let a = [||];;
val a : 'a []
```

Generisk type

- Indicering
```
> [|'h'; 'e'; 'j'|].[1..];;
val it : char [] = [|'e'; 'j'|]
```

Slicing (som strenge og lister)

- Samenligning
```
> [|2; 3; 5|] > [|2; 2; 6|];;
val it : bool = true
```

'Alfabetisk' sammenligning

- Mutérbar!
```
> let a = [|2; 3; 5|]
- a.[2] <- 4
- printfn "%A" a;;
[|2; 3; 4|]
val a : int [] = [|2; 3; 4|]
val it : unit = ()
```

Implicit mutable nøgleord

Køretid

listIndicering.fsx

```
let N = 100000
let lst = [1..N];;
let mutable max = -1
for i = 0 to lst.Length - 1 do
  let v = lst.[i];
  max <- if v > max then v else max
printfn "%A" max
```

fsharpc listIndicering.fsx && time mono listIndicering.exe

arrayIndicering.fsx

```
let N = 100000
let arr = [|1..N|];;
let mutable max = -1
for i = 0 to arr.Length - 1 do
  let v = arr.[i];
  max <- if v > max then v else max
printfn "%A" max
```

fsharpc arrayIndicering.fsx && time mono arrayIndicering.exe

Arrays er lageraritmetik: