Programmering og Problemløsning

3.2+3: Verbose og letvægtssyntaks, funktioner, virkefelter, dokumentation, betingelser, mutérbare værdier (variable), løkker, tupler

Virkefelter (scope)

Navne (i yderste virkefelt) kan ikke overskrives

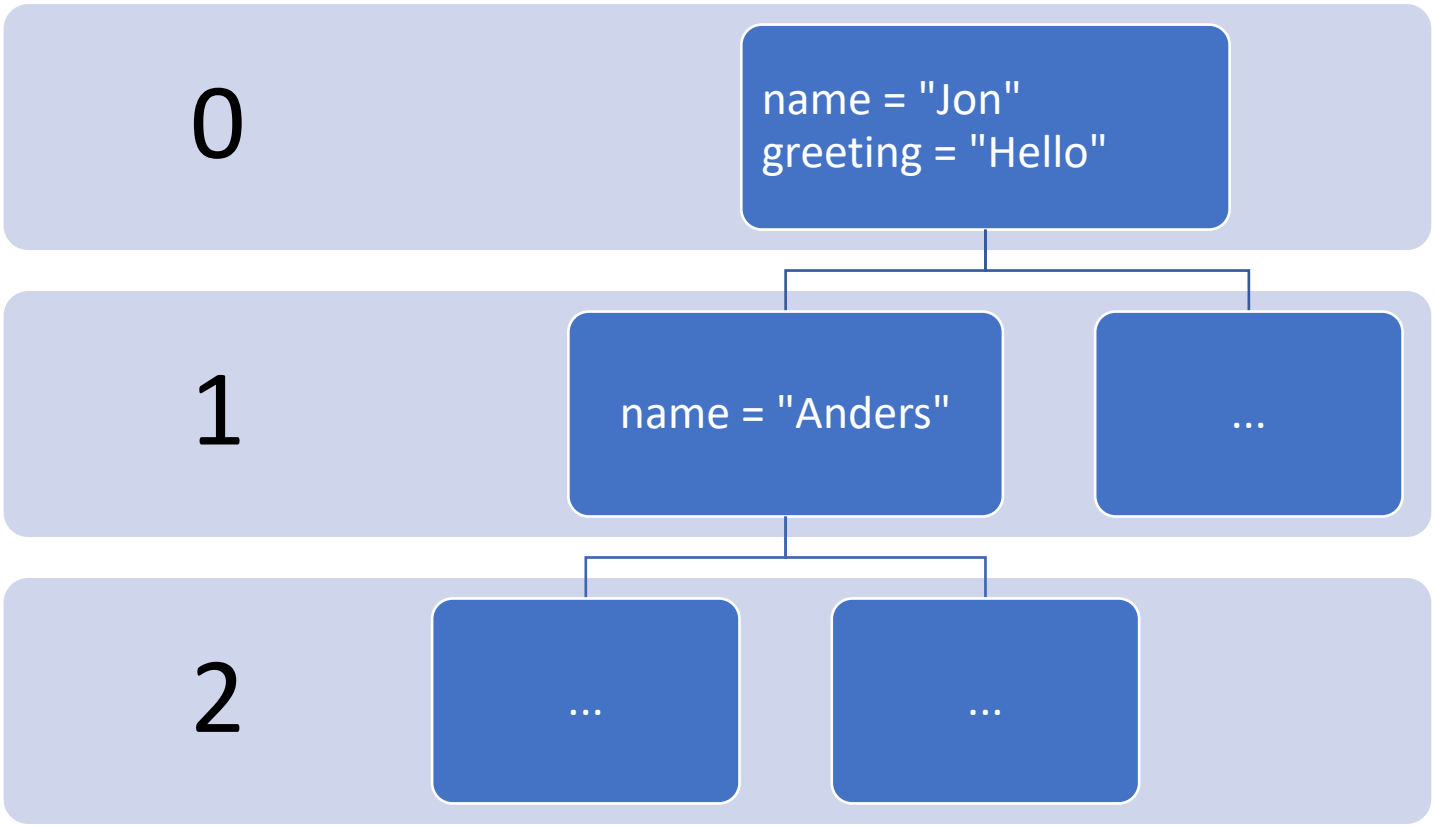
```
let name = "World"
let name = "Jon"
do printfn "Hello %s" name
```

Virkefelter via parenteser

```
let greeting = "Hello"
let name = "Jon"
do printfn "%s %s" greeting name
(
    let name = "Anders"
    do printfn "%s %s" greeting name
)
do printfn "%s %s" greeting name
```

0

1



Syntaks og virkefelter

Letvægtssyntaks

```
let name = "World"
do printfn "Hello %A" name
```

Valgfrit ’do’

```
let name = "World"
printfn "Hello %A" name
```

verbose syntaks

```
let name = "World" in do printfn "Hello %A" name
```

Funktioner

Leksikografisk virkefelt

Organisering = nemmere at forstå og vedligeholde

```
let greetings (name : string) : string =
    "Hello " + name
```

Indryk angiver funktionskroppen

```
let str = greetings "Jon"
printfn "%s" str
printfn "%s" (greetings "World")
```

```
> let greetings (name : string) : string =
-   "Hello " + name;;
val greetings : name:string -> string
```

```
let greetings name =
  "Hello " + name
```

```
let greetings name = "Hello " + name
```

```
let greetings name : string = "Hello " + name
```

```
let greetings (name : string) = "Hello " + name
```

Løs en andengradsligning (baglæns!)

```
let discriminant a b c =
  b ** 2.0 - 4.0 * a * c

let solution a b c sgn =
  let d = discriminant a b c
  (-b + sgn * sqrt d) / (2.0 * a)

let a = 1.0
let b = 0.0
let c = -1.0
let xp = (solution a b c +1.0)
printfn "0 = %fx^2 + %fx + %f => x_+ = %f" a b c xp
```

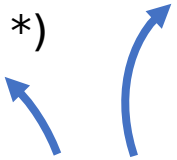
$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Dokumentation - simpel

Summary felt efter dokumentationsstandarden

```
/// The discriminant of a quadratic equation with parameters a, b, and c
let discriminant a b c = b ** 2.0 - 4.0 * a * c // Note that F# will automatically typecast to float
(* This function needs to be tested! *)
```

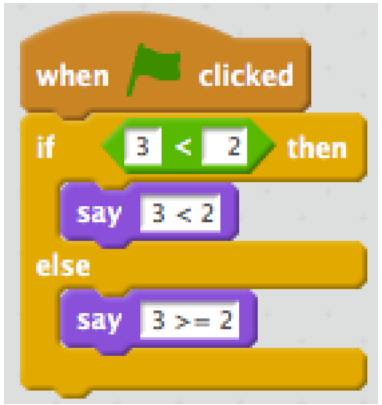


Almindelig kommentarer udenfor dokumentationsstandarden

Dokumentation - grundig

```
/// <summary>Find x when 0 = ax^2+bx+c.</summary>
/// <remarks>Negative discriminants are not checked.</remarks>
/// <example>
///   The following code:
///   <code>
///     let a = 1.0
///     let b = 0.0
///     let c = -1.0
///     let xp = (solution a b c +1.0)
///     printfn "0 = %.1fx^2 + %.1fx + %.1f => x_+ = %.1f" a b c xp
///   </code>
///   prints <c>0 = 1.0x^2 + 0.0x + -1.0 => x_+ = 0.7</c> to the console.
/// </example>
/// <param name="a">Quadratic coefficient.</param>
/// <param name="b">Linear coefficient.</param>
/// <param name="c">Constant coefficient.</param>
/// <param name="sgn">+1 or -1 determines the solution.</param>
/// <returns>The solution to x.</returns>
let solution a b c sgn =
  let d = discriminant a b c
  (-b + sgn * sqrt d) / (2.0 * a)
```

Betingelser



If-then-else

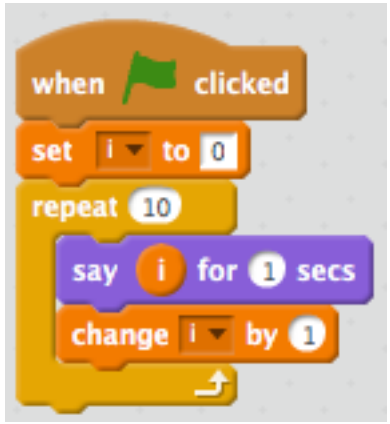
```
> if 3 < 2 then
-   printfn "3 < 2"
- else
-   printfn "3 >= 2";;
3 >= 2
val it : unit = ()

> let str =
-   if 3 < 2 then
-     "3 < 2"
-   else
-     "3 >= 2";;
val str : string = "3 >= 2"
```

Kæde af betingelser

```
> let str =
-   if 3 < 2 then
-     "3 < 2"
-   elif 3 = 2 then
-     "3 = 2"
-   else
-     "3 > 2";;
val str : string = "3 > 2"
```

Muterbare værdier og løkker



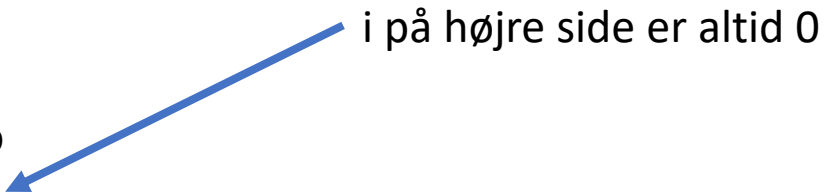

```
let mutable x = 5
printfn "%d" x
x <- -3
printfn "%d" x
```

```
for i = 1 to 10 do
  printf "%d " i
printfn ""
```

```
let mutable i = 1
while i <= 10 do
  printf "%d " i
  i <- i + 1
printf "\n"
```

Hvad gør programmet?

```
let i = 0
while i < 3 do
  let i = i + 1
  printfn "%d" i
```



Tupler

```
$fsharpi
...
> let a = (1, 1.0);;
val a : int * float = (1, 1.0)

> printfn "%A %A" (fst a) (snd a);;
1 1.0
val it : unit = ()

> let b = 1, "en", '\049'
val b : int * string * char = (1, "en", '1')
```

Produkttype

Funktioner til at indicerer i par

Parentes unødvendig men anbefalelses

```
> let (b1, b2, b3) = b;;
val b3 : char = '1'
val b2 : string = "en"
val b1 : int = 1

> let mutable c = (1,2)
- c <- (2,3)
- printfn "%A" c;;
(2, 3)
val mutable c : int * int = (2, 3)
val it : unit = ()
```

Venstre side af en binding kan have navngivne tuple-elementer

Hele typen - ikke enkelt - elementer kan være mutérbare