Dev 8
Functional Programming

Start: 15:10

Vooraf

Voor vragen:

1) Steek je hand op
2) Stel ze in de chat

Wie zijn wij?

Ricardo Stam

0913788@hr.nl

Marcel Bostelaar

0917554@hr.nl

Het plan

Extra lessen i.c.m de dev videos.

We gaan ervan uit dat je de dev videos al bekeken hebt.

github.com/CSARotterdam/Dev_8_Extra_lessen

De planning

Week 1 – Lambda calculus & F# basics

Week 2 – F#; Data structuren & pattern matching

Week 3 – F#; Functie compositie & pipe operators

Voor vandaag

- Recap vorige week
- Opdracht van vorige week

- Functie compositie
  - Pipes
  - Ketens

Recap

Recap – Week 2

Generics

```
let returnTweedeWaarde (arg1 : 'a) (arg2 : 'b) : 'b = arg2
```

```
let returnEersteWaarde (arg1 : 'type1) (arg2 : 'type2) : 'type1 = arg1
```

Recap – Week 2

Tuples

```
let (eentuple : string*int) = "hallo",44

let (eentuple2 : string*int*bool) = "hallo",44,true
```

```
let voorbeeldTuple = 21, "Fred"

let leeftijd, naam = voorbeeldTuple

let grooteTuple = 21, "Fred", "de Jonge", "Rotterdam"

let leeftijd2, naam2, achternaam, woonplaats = grooteTuple
```

Recap – Week 2

Lists

```
let legeLijst = []

let voorbeeldlijst = ["hallo" ; "dit zijn elementen" ; "Ze moeten allemaal hetzelfde type hebben"]

let itemToevoegen = "nieuw item" :: voorbeeldlijst

let kanNietAanEinde = voorbeeldlijst :: "nieuw item"
```

Recap – Week 2

Records

```
type voorbeeldRecord = {fieldName1: string; fieldName2: int}

let instanceVanVoorbeeld = {fieldName1 = "dit is de string"; fieldName2 = 42}

type genericRecord<'a> = {veld1: 'a; veld2: int}

let (instance1Generic : genericRecord<string>) = {veld1 = "met string"; veld2 = 42}
let (instance2Generic : genericRecord<int list>) = {veld1 = [1;2;3;4;] ; veld2 = 42}
```

Recap – Week 2

Records

```
type voorbeeldRecord = {fieldName1: string; fieldName2: int}
let instance = {fieldName1 = "dit is de string"; fieldName2 = 42}

let veld1 = instance.fieldName1
let veld2 = instance.fieldName2
```

Recap – Week 2

Records

```
type voorbeeldRecord = {fieldName1: string; fieldName2: int}

let instanceVanVoorbeeld = {fieldName1 = "dit is de string"; fieldName2 = 42}

let verander1veld = {instanceVanVoorbeeld with fieldName2 = 420}
```

Recap – Week 2

Discriminated unions

```
type discriminatedUnionVoorbeeld =
    | EersteMogelijkheid of int
    | TweedeMogelijkheid of string
    | DerdeMogelijkheid of string*int
    | VierdeMogelijkheid of int
    | VijfdeMogelijkheid of int //Meerdere mogelijkheden mogen hetzelfde type hebben
    | MogelijkheidZonderWaarde  //Mogelijkheden zonder waarde kunnen ook
```

Recap – Week 2

Pattern matching

```
type voorbeeldUnion =
    | EersteMogelijkheid of int
    | TweedeMogelijkheid of string
    | DerdeMogelijkheid of string*int

let matchHet x : string =
    match x with
    | EersteMogelijkheid intwaarde -> intwaarde.ToString()
    | TweedeMogelijkheid stringwaarde -> stringwaarde
    | DerdeMogelijkheid (stringwaarde, intwaarde) -> stringwaarde + ", " + intwaarde.ToString()

let matchHet2 x : string =
    match x with
    | EersteMogelijkheid 420 -> "Deze regel matcht alleen cases van eerstewaarde met de waarde 420"
    | EersteMogelijkheid intwaarde -> intwaarde.ToString()
    | TweedeMogelijkheid stringwaarde -> stringwaarde
    | DerdeMogelijkheid (stringwaarde, 42) -> stringwaarde + ", hier matchde ik specifiek 42"
    | DerdeMogelijkheid (stringwaarde, intwaarde) -> stringwaarde + ", " + intwaarde.ToString()
```

Recap – Week 2

Pattern matching

```
let lijstenMatchen (lijst : 'a list) =
    match lijst with
    | [] -> "Lege lijst"
    | [x] -> "lijst met 1 item erin: " + x.ToString()
    | [x; y] -> "lijst met 2 items erin: " + x.ToString() + ", " + y.ToString()
    | head :: tail -> "een lijst met ten minste 1 waarde en een staart met 0 of meer items"

let minimaleLijstVerwerking (lijst : 'a list) =
    match lijst with
    | [] -> "Lege lijst"
    | head :: tail -> "1 Waarde " + head.ToString() + " en een staart van 0 of meer items"
```

Recap – Week 2

Option

```
type Option<'a> =
    | Some of 'a
    | None

let Niet420 (getal : int) : Option<int> =
    match getal with
    | 420 -> None
    | x -> Some x
```

Opdracht

Opdracht – Week 2

1. Maak records: cirkel met radius, rechthoek met lengte en breedte
2. Maak een discriminated union "vorm" met rechthoek en cirkel er in
3. Maak een functie die de oppervlakte van een rechthoek berekent
4. Maak een functie die de oppervlakte van een cirkel berekent
5. Maak een functie die de oppervlakte van een vorm berekent
6. Maak een functie die een lijst met vormen accepteert en alleen de cirkels teruggeeft en doe hetzelfde voor rechthoeken

Opdracht – Week 2

- Maak records: cirkel met radius, rechthoek met lengte en breedte
- Maak een discriminated union "vorm" met rechthoek en cirkel er in

```
type cirkel = {radius: float}

type rechthoek = {breedte: float; lengte: float}

type vorm =
    | Cirkelvorm of cirkel
    | Rechthoekvorm of rechthoek
```

Opdracht – Week 2

- Maak een functie die de oppervlakte van een rechthoek berekent
- Maak een functie die de oppervlakte van een cirkel berekent
- Maak een functie die de oppervlakte van een vorm berekent

```
let oppervlakteRechthoek (rechthoek : rechthoek) = rechthoek.breedte * rechthoek.lengte

let oppervlakteCirkel (cirkel : cirkel) = 3.14 * (cirkel.radius ** 2.0)

let oppervlakteVorm (vorm : vorm) =
    match vorm with
    | Cirkelvorm z -> oppervlakteCirkel z
    | Rechthoekvorm x -> oppervlakteRechthoek x
```

Opdracht – Week 2

- Maak een functie die een lijst met vormen accepteerd en alleen de cirkels teruggeeft en doe het zelfde voor rechthoeken

```
let rec mapList (func : 'a -> 'b) (list : 'a list) : 'b list =
    match list with
    | [] -> []
    | head :: tail -> func head :: mapList func tail

let rec optionOnlyExistingValue (list : 'a option list) : 'a list =
    match list with
    | [] -> []
    | head :: tail ->
        match head with
        | Some x -> x :: optionOnlyExistingValue tail
        | None -> optionOnlyExistingValue tail

let FilterLijst (filterFunctie : 'a -> 'b option) (list : 'a list) : 'b list =
    optionOnlyExistingValue (mapList filterFunctie list)
```

Opdracht – Week 2

- Maak een functie die een lijst met vormen accepteerd en alleen de cirkels teruggeeft en doe het zelfde voor rechthoeken

```
let alleenCirkels vorm =
    match vorm with
    | Cirkelvorm x -> Some x
    | _ -> None

let alleenRechthoeken vorm =
    match vorm with
    | Rechthoekvorm x -> Some x
    | _ -> None

let filterCirkels = FilterLijst alleenCirkels
let filterRechthoeken = FilterLijst alleenRechthoeken
```

Functie compositie

Functie compositie

Functie compositie is het combineren van meerdere functies om een grotere abstracte functie te maken.

```
let funA = fun a -> b
let funB = fun b -> c
let FunAdanFunB = fun a -> c
```

```
let plus1 input = input + 1

let plus2 input = plus1 (plus1 input)

let plus3 input = plus1 (plus2 input)
```

Functie compositie

Functie compositie kan gedaan worden door middel van operators

Alle operators zijn left associative, je leest ze van links naar rechts, als er geen haakjes om heen staan.

Pipe operator

Pipe operator

Pipe operators |> en <| sturen de uitkomst/waarde van hun platte kant door naar de pijlkant.

Input |> Target

```
let pipe input =
    keer40 input |> plus5
```

Target <| Input

```
let pipe input =
    plus5 <| keer40 input
```

Pipe operator

|> <|

Pipe = plus5(keer40(input))

Pipe2 = keer40(plus5(input))

```
let pipe input = keer40 input |> plus5
//let pipe input = plus5 <| keer40 input
let pipe2 input = keer40 <| plus5 input
//let pipe2 input = plus5 input |> keer40
```

LET OP: De uitkomst is een waarde, geen functie

Pipe operator

Compositie met pipes kan beide kanten op werken.

De pipe operator is left associative.

```
let pipeOperation1 input = keer40 input |> plus5 |> plus5
                         //=> (keer40 input|> plus5) |> plus5

let pipeOperation2 input = keer40 <| plus5 input |> plus5
                         //=> (keer40 <| plus5 input) |> plus5
```

Pipe operator

Dit is de volgorde van uitvoering

```
let pipe input = keer40 <| plus5 input |> plus5
let pipe input = (keer40 <| plus5 input) |> plus5
```

```
let pipe input = keer40 <| plus5 input |> plus5
//input = 3
keer40 <| plus5 3 |> plus5
keer40 <| 8 |> plus5
keer40 8 |> plus5
320 |> plus5
plus5 320
325
```

Ketens

Ketens

>> en << operators

"Knopen" twee functies aan elkaar.

Maakt een nieuwe functie die
eerst de ene, dan de andere
uitvoert met het resultaat van de ene

```
let keer40 input = input * 40

let plus5 input = input + 5

let keten = keer40 >> plus5

let keten2 = keer40 << plus5
```

Ketens

>> en << operators

Doen hetzelfde in omgekeerde volgorde

Keten = (input * 40) + 5

Keten2 = (input + 5) * 40

```
let keer40 input = input * 40

let plus5 input = input + 5

let keten = keer40 >> plus5

let keten2 = keer40 << plus5
```

LET OP: Uitkomst is hier een functie, geen waarde

Ketens

Let op, je kan niet alles achter elkaar plakken
In dit voorbeeld geeft links een returnwaarde "int -> int"
Maar rechts verwacht een "int", niet een functie "int ->int"

```
let keer40 input = input * 40
```

```
let som a b = a + b

let keten = keer40 >> som

let keten = som >> keer40
```

```
let chainOperator links rechts =
    fun x -> rechts (links x)

let chainOperator (links : int -> int -> int) (rechts: int -> int) =
    fun x -> rechts (links x)
```

Samengevat

Samenvatting

- Pipe operators sturen waardes van hun platte kant naar hun pijlkant
- Chain operators verbinden twee functies aan elkaar, geen inputbenoeming nodig
- Beide zijn left-associative (voer uit van links naar rechts)
- Deze operators kan je gebruiken in functie compositie voor betere leesbaarheid

Vragen?

Dank voor jullie aandacht en tijd!