- DEV 8 -
Functioneel programmeren

•••

Quick & Dirty Recap

Wat gaan we bespreken

- Lambda calculus
  - Expressies
  - Shadowing
  - Symbolen
  - Opdrachten

Wat gaan we bespreken

- F#
  - Code
    - Voorbeelden
  - Opdrachten

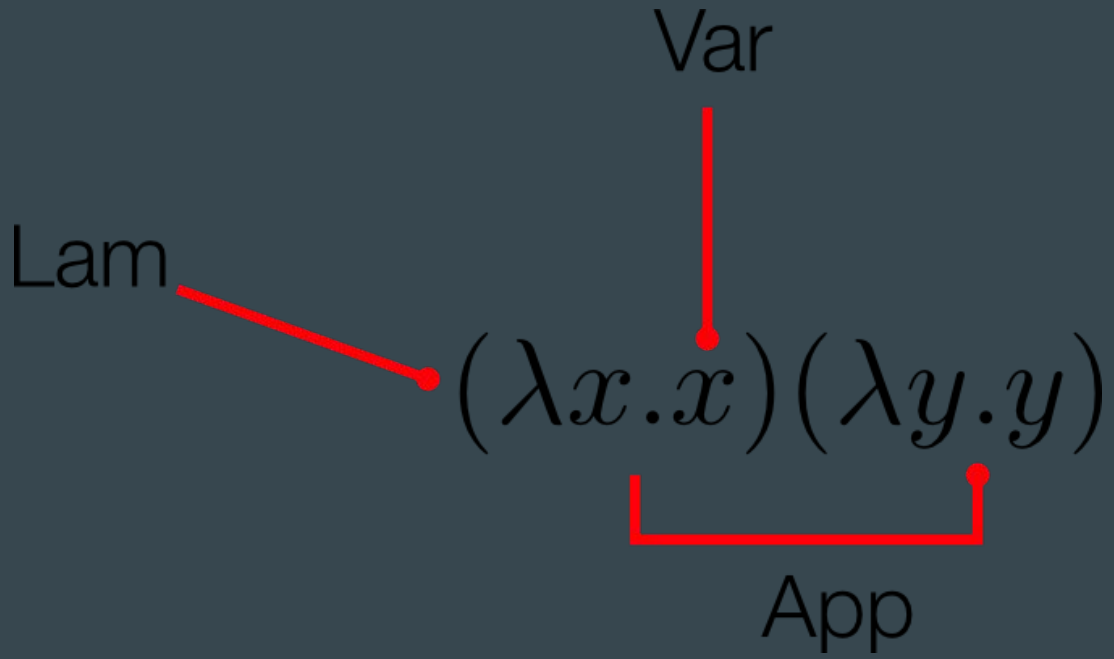
$\lambda x\, y.y\ 1\ 0$

Lambda calculus

...

$\lambda x\, y.x$

Lambda calculus



Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda - Expressies

Lambda calculus

Lambda calculus is een “eenvoudige” taal.

Alle elementen in de taal zijn een “Expressie” (E).

Lambda calculus

Lambda calculus is een “eenvoudige” taal.

Mogelijke expressies (E):

Variabele - E

Variabele - x

Lambda calculus

Lambda calculus is een “eenvoudige” taal.

Mogelijke expressies (E):

Variabele - E
Functie definitie - λ E . E || λ E → E

Variabele - x
Functie definitie - λ x . x

Lambda calculus

Lambda calculus is een “eenvoudige” taal.

Mogelijke expressies (E):

Variabele - E
Functie definitie - λ E . E || λ E → E
Functie applicatie - E E

Variabele - x
Functie definitie - λ x . x
Functie applicatie - (λ x . x) y

Lambda calculus

Variabele - E
Functie definitie - λ E . E
Functie applicatie - E E

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

Variabele - E
Functie definitie - λ E . E
Functie applicatie - E E

```
def true(x, y):
    return x
```

Definities:
(T)rue - λx y . x
(F)alse - λx y . y

Lambda calculus

Variabele - E
Functie definitie - λ E . E
Functie applicatie - E E

```
def false(x, y):
    return y
```

Definities:
(T)rue - λx y . x
(F)alse - λx y . y

Lambda calculus

Variabele - E
Functie definitie - λ E . E
Functie applicatie - E E

```
λxy.x 1 0

((λxy.x) 1) 0   //(EE)E
```

Definities:
(T)rue - λx y . x
(F)alse - λx y . y

Lambda - Beta reduction

Lambda calculus

Definities:
(T)rue - λx y . x
(F)alse - λx y . y

1. ((T) 1) 0
2. ((F) 1) 0

Lambda calculus

Definities:

(T)rue - λx y . x

(F)alse - λx y . y

(A)nd - λ x y. x y x

Lambda calculus

Definities:
(T)rue - λx y . x
(F)alse - λx y . y
(A)nd - λ x y. x y x

```
def AND(x, y):
    if x:
        return y
    return x
```

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

Definities:
(T)rue - λx y . x
(F)alse - λx y . y
(A)nd - λ x y. x y x

```
λxy.xyx F T

((λxy.xyx) F) T

(λy.FyF) T

(FTF) //((EE)E)

F
```

Lambda calculus

Definities:
(T)rue - λx y . x
(F)alse - λx y . y
(A)nd - λ x y. x y x
(O)r - λ x y. x x y

```
def OR(x, y):
    if x:
        return x
    return y
```

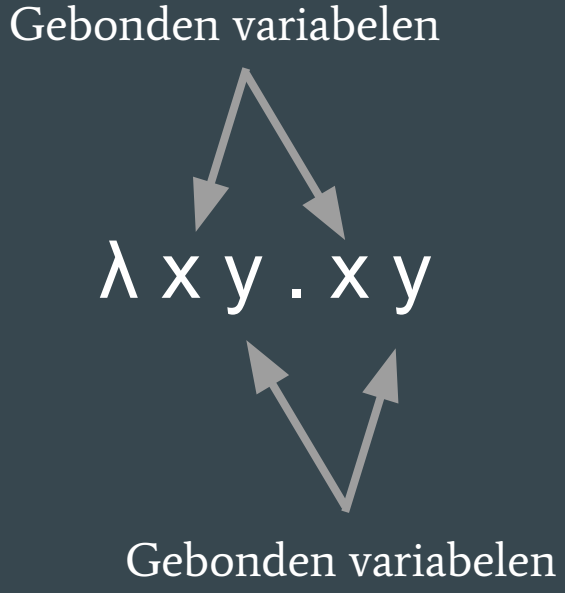
Lambda calculus

Definities:
(T)rue - λx y . x
(F)alse - λx y . y
(A)nd - λ x y. x y x
(O)r - λ x y. x x y

```
λxy.xxy F T

((λxy.xxy) F) T

(λy.FFy) T

(FFT) //((EE)E)

T
```

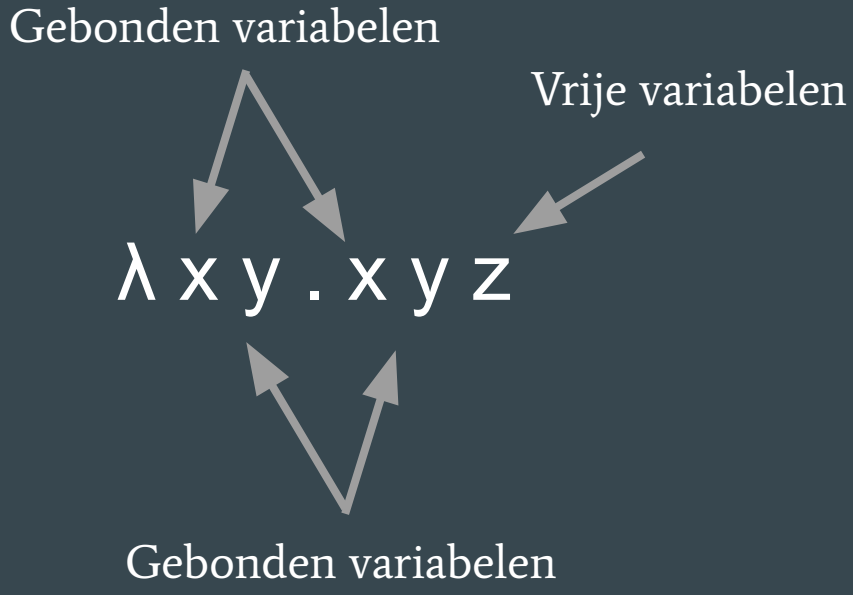
Lambda - Shadowing

Lambda calculus



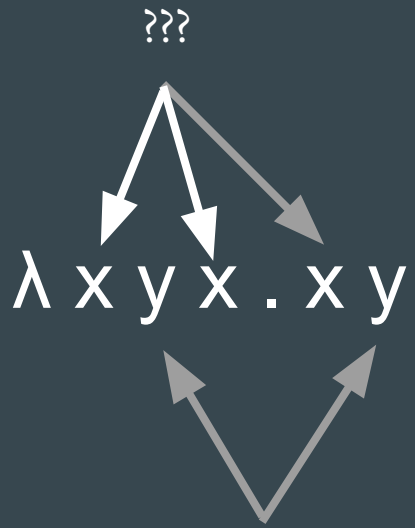
Introductie - Expressies - Beta redux - **Shadowing** - Symbolen

Lambda calculus



Introductie - Expressies - Beta redux - **Shadowing** - Symbolen

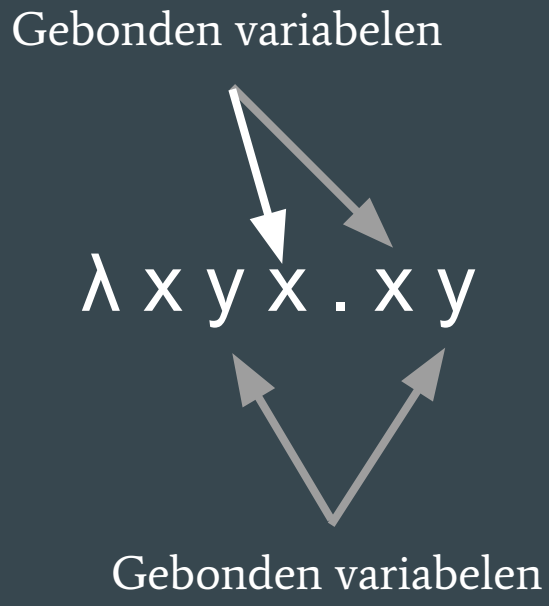
Lambda calculus



Gebonden variabelen

Introductie - Expressies - Beta redux - **Shadowing** - Symbolen

Lambda calculus



Introductie - Expressies - Beta redux - **Shadowing** - Symbolen

Lambda - symbolen

Lambda calculus

Wat is de volgende stap?

$$\overline{(\lambda x \to t)\ u \to_\beta t[x \mapsto u]}$$

$$\begin{cases} x = \ldots \\ t = \ldots \\ u = \ldots \\ t[x \mapsto u] = \ldots \end{cases}$$

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

((λy z. y) 1) 0

x =

t =

u =

t[x -> u] =

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

((λy z. y) 1) 0

x = y
t =
u =
t[x -> u] =

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

((λy z. y) 1) 0

x = y
t = (λz. y)
u =
t[x -> u] =

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

((λy z. y) 1) 0

x = y
t = (λz. y)
u = 1
t[x -> u] =

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

((λy z. y) 1) 0

x = y
t = (λz. y)
u = 1
t[x -> u] = (λz . 1) 0

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

((λy z. y) 1) 0

x = y
t = (λz. y)
u = 1
t[x -> u] = (λz . 1) 0

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

(((λx y. x y) (λz. z+1)) 5)

$$\frac{t \to_\beta t' \wedge u \to u' \wedge t'\ u' \to_\beta v}{t\ u \to_\beta v}$$

$$\begin{cases} t = \ldots \\ u = \ldots \\ t' = \ldots \\ u' = \ldots \\ v = \ldots \end{cases}$$

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

(((λx y. x y) (λz. z+1)) 5)

t =

u =

tl =

ul =

v =

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

(((λx y. x y) (λz. z+1)) 5)

t = (λxy.x y) (λz.z + 1)
u =
tl =
ul =
v =

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

(((λx y. x y) (λz. z+1)) 5)

t = (λxy.x y) (λz.z + 1)
u = 5
tl =
ul =
v =

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

(((λx y. x y) (λz. z+1)) 5)

t = (λxy.x y) (λz.z + 1)
u = 5
t1 = (λy.(λz.z + 1) y)
u1 =
v =

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

(((λx y. x y) (λz. z+1)) 5)

t = (λxy.x y) (λz.z + 1)
u = 5
t1 = (λy.(λz.z + 1) y)
u1 = 5
v =

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus

(((λx y. x y) (λz. z+1)) 5)

t = (λxy.x y) (λz.z + 1)
u = 5
t1 = (λy.(λz.z + 1) y)
u1 = 5
v = (λz.z + 1) 5 || 6

Introductie - Expressies - Beta redux - Shadowing - Symbolen

Lambda calculus: Opdrachten

...

Lambda calculus - Opdrachten

1. (((λxyz.zxy) "Hi ") "Name ") (λxy. x + y + "!")
2. (((λxyz.y(zx)) 6) (λy.y-4)) (λx.x+2)

Lambda calculus - Opdrachten

(((λxyz.z x y) "Hi") "Name") (λxy. x + y + "!")

1. (((λxyz.z x y) "Hi ") "Name") (λxy. x + y + "!")
2. ((λyz.z "Hi " y) "Name") (λxy. x + y + "!")
3. (λz.z "Hi " "Name") (λxy. x + y + "!")
4. (λxy. x + y + "!") "Hi " "Name"
5. (λy. "Hi " + y + "!") "Name"
6. ("Hi " + "Name" + "!")
7. "Hi Name!"

Lambda calculus - Opdrachten

(((λxyz.z x y) "Hi") "Name") (λxy. x + y + "!")

1. (((λxyz.z x y) "Hi ") "Name") (λxy. x + y + "!")
2. ((λyz.z "Hi " y) "Name") (λxy. x + y + "!")
3. (λz.z "Hi " "Name") (λxy. x + y + "!")
4. (λxy. x + y + "!") "Hi " "Name"
5. (λy. "Hi " + y + "!") "Name"
6. ("Hi " + "Name" + "!")
7. "Hi Name!"

Lambda calculus - Opdrachten

(((λxyz.y (z x)) 6) (λy.y-4)) (λx.x+2)

---

1. (((λxyz.y (z x)) 6) (λy.y-4)) (λx.x+2)
2. ((λyz.y (z 6)) (λy.y-4)) (λx.x+2)
3. (λz.(λy.y-4) (z 6)) (λx.x+2)
4. (λy.y-4) ((λx.x+2) 6)
5. (λy.y-4) (6+2)
6. 6+2-4
7. 8-4
8. 4

Lambda calculus - Opdrachten

(((λxyz.y (z x)) 6) (λy.y-4)) (λx.x+2)

---

1. (((λxyz.y (z x)) 6) (λy.y-4)) (λx.x+2)
2. ((λyz.y (z 6)) (λy.y-4)) (λx.x+2)
3. (λz.(λy.y-4) (z 6)) (λx.x+2)
4. (λy.y-4) ((λx.x+2) 6)
5. (λy.y-4) (6+2)
6. 6+2-4
7. 8-4
8. 4

F#

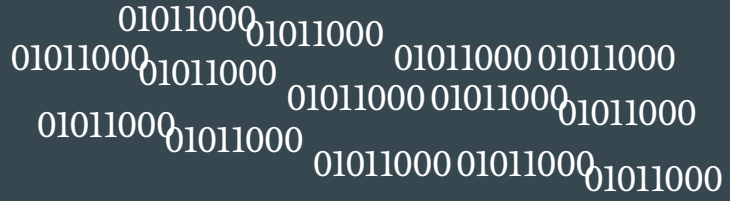
...

F# - Syntax

F#

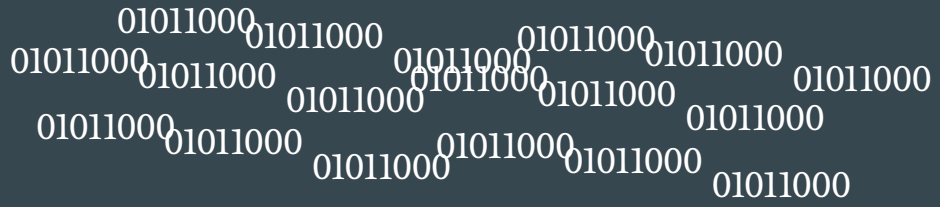
Slides en project

https://github.com/CSARotterdam/Development-8-recap-1819


01011000 01011000
01011000 01011000 01011000 01011000
01011000 01011000 01011000
01011000 01011000 01011000

Opdracht

...


01011000 01011000
01011000 01011000 01011000 01011000 01011000
01011000 01011000 01011000 01011000 01011000
01011000 01011000 01011000 01011000
01011000 01011000 01011000

F# - Opdrachten

Maak een power off functie. Het is alleen toegestaan om +,-,* en / operator te gebruiken. Functie definitie is als volgt:

n:int -> m:int -> int

F# - Opdrachten

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000

F# - Opdrachten

En nu?

F# - Opdrachten

Opdrachten van:

Reader 1, 3 en 4

Vragen?

...

Feedback?

...