The Joy of Functional Programming

NICOLE RAUCH
softwareentwicklung &
entwicklungscoaching

Functional, isn't that a totally esoteric subject?!

Functional, isn't that a totally esoteric subject?!

- ABN AMRO Amsterdam *Risk analysis in investment banking*
- AT&T *Network security: processing of internet abuse complaints*
- Bank of America Merril Lynch
  Backend data transformation and loading
- Barclays Capital Quantitative Analytics Group
  Mathematical modelling of equity derivatives
- Bluespec, Inc. *Modelling & verification of integrated circuits*
- Credit Suisse *Checking, manipulating and transforming spreadsheets*
- Deutsche Bank Equity Proprietary Trading, Directional Credit Trading *All its software infrastructure*
- Facebook *Internal tools*
- Factis Research, Freiburg *Mobile solutions (backend)*
- fortytools gmbh *web-based productivity tools - REST-backend*
- Functor AB, Stockholm *static analysis*

Functional, isn't that a totally esoteric subject?!

- Galois, Inc *Security, information assurance and cryptography*
- Google *Internal projects*
- IMVU, Inc. *Social entertainment*
- JanRain *Network and web software*
- MITRE *Analysis of kryptographic protocols*
- New York Times *Image processing for the New York Fashion Week*
- NVIDIA *In-house tools*
- Parallel Scientific *High-availability cluster management system*
- Sankel Software *CAD/CAM, gaming and computer animation*
- Silk, Amsterdam *Filter and visualize large amounts of information*
- Skedge Me *Online scheduling platform*
- Standard Chartered *Wholesale banking business*
- Starling Software, Tokio *Commercial automated options trading system*
- Suite Solutions *Management of large sets of technical documentation*

(Quelle: `http://www.haskell.org/haskellwiki/Haskell_in_industry`)

Well-known functional languages

Scheme

Erlang

Clojure

ML

F#

Miranda

OCaml

Haskell

Lisp

Scala

Well-known functional languages

Scheme

Erlang

Clojure

ML

F#

Miranda

OCaml

Haskell

(Java 8)

Lisp

Scala

(JavaScript)

Well-known functional languages

Scheme

Erlang

Clojure

ML

F#

Miranda

OCaml

Haskell

Lisp

Scala

(JavaScript)

Now, what is so special about functional programming?

Now, what is so special about functional programming?

Immutability

Now, what is so special about functional programming?

Immutability

Each variable can only be assigned to once

Now, what is so special about functional programming?

Immutability

Each variable can only be assigned to once

Functions are „first order citizens“

Now, what is so special about functional programming?

Immutability

Each variable can only be assigned to once

Functions are „first order citizens“

Functions can be treated in the same way as strings or numbers

Functions are values

Functions are values

JavaScript:

```
function times(x, y) { return x * y; }

var timesVar = times;

timesVar(3, 5) === 15;
```

Functions are values

JavaScript:

```
function times(x, y) { return x * y; }

var timesVar = times;

timesVar(3, 5) === 15;
```

Haskell:

```
times x y = x * y

timesVar = times

timesVar 3 5 == 15
```

Functions are function parameters

Functions are function parameters

JavaScript:

```
function times3(y) { return 3 * y; };

function apply(func, arg) { return func(arg); }

apply(times3, 5) === 15;
```

Functions are function parameters

JavaScript:

```
function times3(y) { return 3 * y; };

function apply(func, arg) { return func(arg); }

apply(times3, 5) === 15;
```

Haskell:

```
apply func arg = func arg

apply (\ x -> 3 * x) 5 == 15
```

Functions are return values

Functions are return values

JavaScript:

```
function times(x) { return function (y) { return x * y; }; }

times(3)(5) === 15;
```

Functions are return values

JavaScript:

```
function times(x) { return function (y) { return x * y; }; }

times(3)(5) === 15;
```

Haskell:

```
times x = (\y -> x * y)

times 3 5 == 15
```

Strange... ?!

JavaScript: Two different invocations

```
function times(x, y) { return x * y; }
times(3, 5) === 15;

function times(x) { return function (y) { return x * y; }; }
times(3)(5) === 15;
```

Haskell: Two identical invocations

```
times x y = x * y
times 3 5 == 15

times x = (\y -> x * y)
times 3 5 == 15
```

Currying! (also known as Schönfinkeling)

Currying! (also known as Schönfinkeling)

In real functional languages we write:

```
times x y = x * y
```

but actually the following happens:

```
times x = (\y -> x * y)
```

Currying! (also known as Schönfinkeling)

In real functional languages we write:

```
times x y = x * y
```

but actually the following happens:

```
times x = (\y -> x * y)
```

Because functions always take exactly one argument

Currying! (also known as Schönfinkeling)

In real functional languages we write:

```
times x y = x * y
```

but actually the following happens:

```
times x = (\y -> x * y)
```

Because functions always take exactly one argument

Useful for partial evaluation:

```
times x y = x * y

times3 = times 3

times3 5 == 15
```

And what if I don't want an argument?

- In real functional languages, functions always take exactly one argument!
- What if I don't want to pass anything?

And what if I don't want an argument?

- In real functional languages, functions always take exactly one argument!
- What if I don't want to pass anything?
- `Unit` to the rescue!
- `Unit` is a type with only one element
- In Haskell, this element is called `()`

Important library functions: filter

- filter or select

Important library functions: filter

- filter or select
- Takes a collection and a function
- Returns those elements of the collection for which the function yields `true`

Important library functions: filter

- filter or select
- Takes a collection and a function
- Returns those elements of the collection for which the function yields `true`

JavaScript: (using the lodash-library)

```
_.filter([1, 2, 3, 4], function (x) { return x % 2 === 0; }) === [2, 4]
```

Important library functions: filter

- filter or select
- Takes a collection and a function
- Returns those elements of the collection for which the function yields `true`

JavaScript: (using the lodash-library)

```
_.filter([1, 2, 3, 4], function (x) { return x % 2 === 0; }) === [2, 4]
```

Haskell:

```
filter (\x -> x `mod` 2 == 0) [1,2,3,4] == [2,4]
```

Important library functions: map

- map or collect

Important library functions: map

- map or collect
- Takes a collection and a function
- Yields a collection of the results of applying the function to the elements of the original collection

Important library functions: map

- map or collect
- Takes a collection and a function
- Yields a collection of the results of applying the function to the elements of the original collection

JavaScript:

```
_.map( [1, 2, 3, 4], function (x) { return x + 5; } ) === [6, 7, 8, 9]
```

Important library functions: map

- map or collect
- Takes a collection and a function
- Yields a collection of the results of applying the function to the elements of the original collection

JavaScript:

```
_.map( [1, 2, 3, 4], function (x) { return x + 5; } ) === [6, 7, 8, 9]
```

Haskell:

```
map (\x -> x + 5) [1,2,3,4] == [6,7,8,9]
```

Important library functions: fold

- fold or reduce or inject

Important library functions: fold

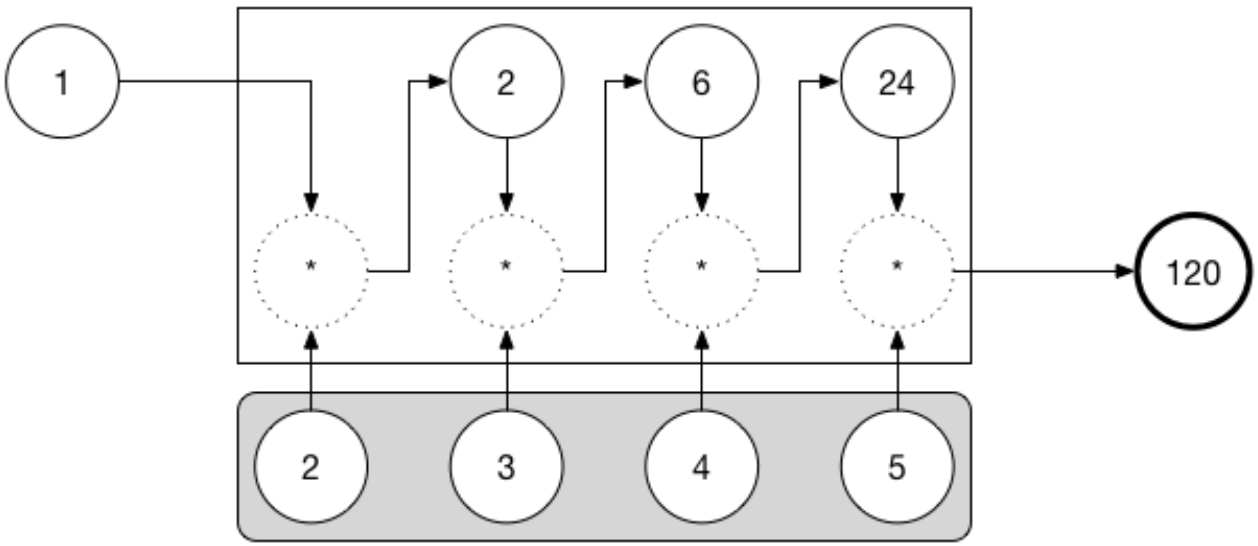
- fold or reduce or inject
- Takes a collection, a function and an initial value
- Merges initial value and first collection entry using the function
- Merges the result and the next collection entry
- Continues for all collection entries, yielding a single result

Important library functions: fold

- fold or reduce or inject

JavaScript:

```
_.foldl( [2, 3, 4, 5], function (x, y) { return x * y; }, 1 ) === 120
```



Important library functions: fold

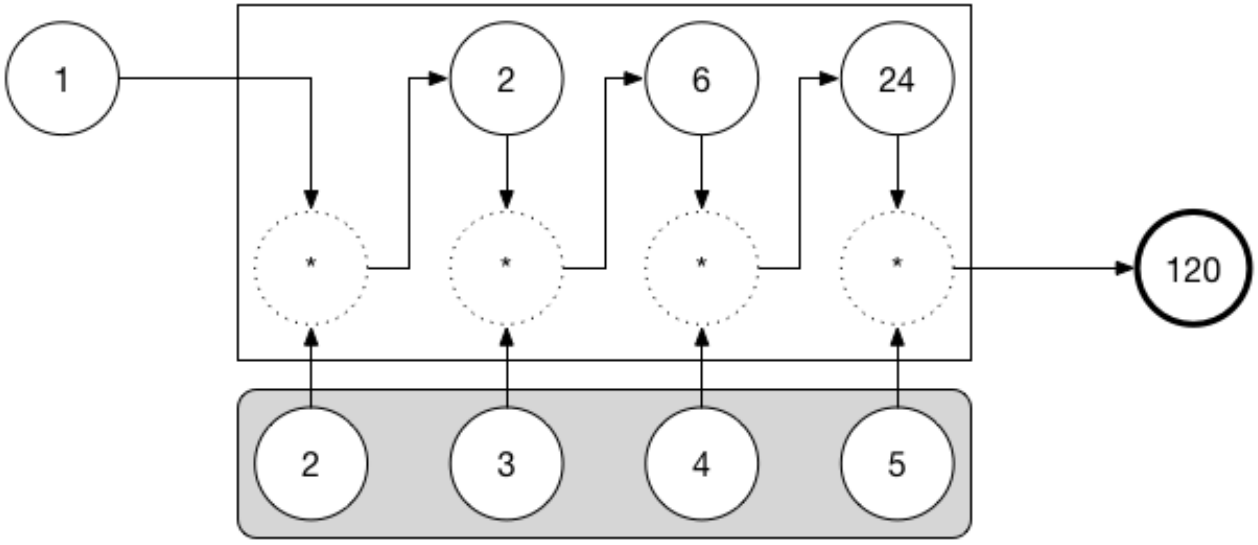
- fold or reduce or inject

JavaScript:

```
_.foldl( [2, 3, 4, 5], function (x, y) { return x * y; }, 1 ) === 120
```

Haskell:

```
foldl   (*) 1 [2,3,4,5] == 120
```



Type inference

- Haskell: strong static type system
- Lightweight usage thanks to type inference
- Derivation of the most general type

Type inference

- Haskell: strong static type system
- Lightweight usage thanks to type inference
- Derivation of the most general type

```
foldl :: (a -> b -> a) -> a -> [b] -> a
(*) :: Num a => a -> a -> a

foldl   (*) 1 [2,3,4,5]
```

Type inference

- Haskell: strong static type system
- Lightweight usage thanks to type inference
- Derivation of the most general type

```
foldl :: (a -> b -> a) -> a -> [b] -> a
(*) :: Num a => a -> a -> a

foldl   (*) 1 [2,3,4,5]
```

Compile error for:

```
foldl   (*) "x" [2,3,4,5]

No instance for (Num [Char]) arising from a use of `*'
Possible fix: add an instance declaration for (Num [Char])
```

A simple calculation

$$sum = \sum_{i=1}^{10} i^2$$

A simple calculation

$$sum = \sum_{i=1}^{10} i^2$$

```
int sum = 0;
for(int i = 1; i <= 10; i++) {
  sum = sum + i * i;
}
```

Excursion: Clean Code

Single Responsibility Principle

Excursion: Clean Code

Single Responsibility Principle

How many responsibilities does this code have?

```
int sum = 0;
for(int i = 1; i <= 10; i++) {
  sum = sum + i * i;
}
```

Excursion: Clean Code

Single Responsibility Principle

How many responsibilities does this code have?

```
int sum = 0;
for(int i = 1; i <= 10; i++) {
  sum = sum + i * i;
}
```

- Creating the sequence of numbers from 1 to 10

Excursion: Clean Code

Single Responsibility Principle

How many responsibilities does this code have?

```
int sum = 0;
for(int i = 1; i <= 10; i++) {
  sum = sum + i * i;
}
```

- Creating the sequence of numbers from 1 to 10
- Calculating the square of a number

Excursion: Clean Code

Single Responsibility Principle

How many responsibilities does this code have?

```
int sum = 0;
for(int i = 1; i <= 10; i++) {
  sum = sum + i * i;
}
```

- Creating the sequence of numbers from 1 to 10
- Calculating the square of a number
- Calculating the square of each number in the sequence

Excursion: Clean Code

Single Responsibility Principle

How many responsibilities does this code have?

```
int sum = 0;
for(int i = 1; i <= 10; i++) {
  sum = sum + i * i;
}
```

- Creating the sequence of numbers from 1 to 10
- Calculating the square of a number
- Calculating the square of each number in the sequence
- Calculating the sum of two numbers

Excursion: Clean Code

Single Responsibility Principle

How many responsibilities does this code have?

```
int sum = 0;
for(int i = 1; i <= 10; i++) {
  sum = sum + i * i;
}
```

- Creating the sequence of numbers from 1 to 10
- Calculating the square of a number
- Calculating the square of each number in the sequence
- Calculating the sum of two numbers
- Calculating the sum of all squares

Separation of Concerns

- Creating the sequence of numbers from 1 to 10
- Calculating the square of a number
- Calculating the square of each number in the sequence
- Calculating the sum of two numbers
- Calculating the sum of all squares

Separation of Concerns

- Creating the sequence of numbers from 1 to 10

```
var sequence = _.range(1, 11);
```

- Calculating the square of a number

- Calculating the square of each number in the sequence

- Calculating the sum of two numbers

- Calculating the sum of all squares

Separation of Concerns

- Creating the sequence of numbers from 1 to 10

```
var sequence = _.range(1, 11);
```

- Calculating the square of a number

```
var square = function (i) { return i * i; };
```

- Calculating the square of each number in the sequence

- Calculating the sum of two numbers

- Calculating the sum of all squares

Separation of Concerns

- Creating the sequence of numbers from 1 to 10

```
var sequence = _.range(1, 11);
```

- Calculating the square of a number

```
var square = function (i) { return i * i; };
```

- Calculating the square of each number in the sequence

```
var squaredSequence = _.map(sequence, square)
```

- Calculating the sum of two numbers

- Calculating the sum of all squares

Separation of Concerns

- Creating the sequence of numbers from 1 to 10

```
var sequence = _.range(1, 11);
```

- Calculating the square of a number

```
var square = function (i) { return i * i; };
```

- Calculating the square of each number in the sequence

```
var squaredSequence = _.map(sequence, square)
```

- Calculating the sum of two numbers

```
var add = function (s1, s2) { return s1 + s2; };
```

- Calculating the sum of all squares

Separation of Concerns

- Creating the sequence of numbers from 1 to 10

```
var sequence = _.range(1, 11);
```

- Calculating the square of a number

```
var square = function (i) { return i * i; };
```

- Calculating the square of each number in the sequence

```
var squaredSequence = _.map(sequence, square)
```

- Calculating the sum of two numbers

```
var add = function (s1, s2) { return s1 + s2; };
```

- Calculating the sum of all squares

```
var sum = _.reduce(squaredSequence, add);
```

Combining the components

JavaScript:

```
var square = function (i) { return i * i; };
var add = function (s1, s2) { return s1 + s2; };
```

```
_.reduce(_.map(_.range(1, 11), square), add) === 385;
```

Combining the components

JavaScript:

```
var square = function (i) { return i * i; };
var add = function (s1, s2) { return s1 + s2; };
```

```
_.reduce(_.map(_.range(1, 11), square), add) === 385;
```

or

```
_(1).range(11).map(square).reduce(add) === 385;
```

Combining the components

JavaScript:

```
var square = function (i) { return i * i; };
var add = function (s1, s2) { return s1 + s2; };
```

```
_.reduce(_.map(_.range(1, 11), square), add) === 385;
```

or

```
_(1).range(11).map(square).reduce(add) === 385;
```

Haskell:

```
foldl   (+) 0 (map (\x -> x*x) [1..10]) == 385
```

Combining the components

JavaScript:

```
var square = function (i) { return i * i; };
var add = function (s1, s2) { return s1 + s2; };
```

```
_.reduce(_.map(_.range(1, 11), square), add) === 385;
```

or

```
_(1).range(11).map(square).reduce(add) === 385;
```

Haskell:

```
foldl   (+) 0 (map (\x -> x*x) [1..10]) == 385
```

or

```
(>.>) x f = f x
[1..10] >.> map (\x -> x*x) >.> foldl (+) 0 == 385
```

Phew!

OK, everybody take a deep breath :-)

Pattern Matching

Fibonacci-Function „naïve“:

```
fib x = if x < 2 then x else fib (x-1) + fib (x-2)
```

Pattern Matching

Fibonacci-Function „naïve“:

```
fib x = if x < 2 then x else fib (x-1) + fib (x-2)
```

Fibonacci-Function with Pattern Matching:

```
fib 0 = 0
fib 1 = 1
fib x = fib (x-1) + fib (x-2)
```

Algebraic Datatypes

Binary tree:

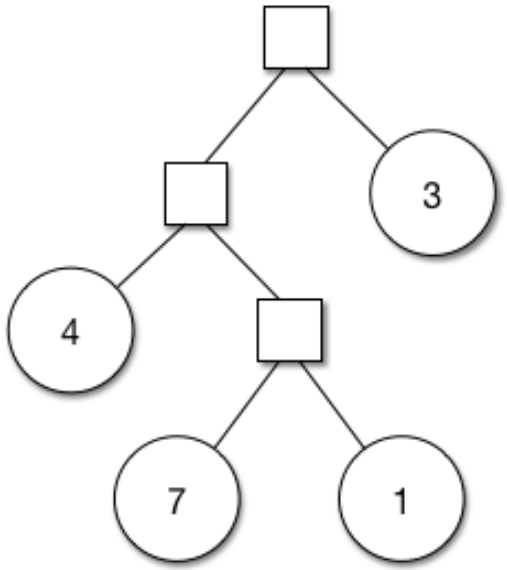
```
data Tree =
       Node Tree Tree
     | Leaf Int
```

Algebraic Datatypes

Binary tree:

```
data Tree =
       Node Tree Tree
     | Leaf Int
```

```
myTree = Node (Node (Leaf 4) (Node (Leaf 7) (Leaf 1))) (Leaf 3)
```



Algebraic Datatypes

Binary tree:

```
data Tree =
       Node Tree Tree
     | Leaf Int
```

```
myTree = Node (Node (Leaf 4) (Node (Leaf 7) (Leaf 1))) (Leaf 3)
```

Summary function:

Algebraic Datatypes

Binary tree:

```
data Tree =
       Node Tree Tree
     | Leaf Int
```

```
myTree = Node (Node (Leaf 4) (Node (Leaf 7) (Leaf 1))) (Leaf 3)
```

Summary function:

```
treeSum (Leaf x) = x
```

Algebraic Datatypes

Binary tree:

```
data Tree =
       Node Tree Tree
     | Leaf Int
```

```
myTree = Node (Node (Leaf 4) (Node (Leaf 7) (Leaf 1))) (Leaf 3)
```

Summary function:

```
treeSum (Leaf x) = x

treeSum (Node m n) = treeSum m + treeSum n
```

Algebraic Datatypes

Binary tree:

```
data Tree =
       Node Tree Tree
     | Leaf Int
```

```
myTree = Node (Node (Leaf 4) (Node (Leaf 7) (Leaf 1))) (Leaf 3)
```

Summary function:

```
treeSum (Leaf x) = x

treeSum (Node m n) = treeSum m + treeSum n
```

```
treeSum myTree == 15
```

Bottom line

- Functional programming is more common than you may have expected
- Some of it can be integrated into non-functional coding
- Many languages have functional aspects or additional modules

Bottom line

- Functional programming is more common than you may have expected
- Some of it can be integrated into non-functional coding
- Many languages have functional aspects or additional modules

References:

- Functional JS-Library: `http://ramdajs.com`
- Haskell: `http://www.haskell.org`

Thank you very much!

Code & slides on GitHub:

`https://github.com`
`/NicoleRauch/FunctionalProgrammingForBeginners`

Nicole Rauch

E-Mail `info@nicole-rauch.de`

Twitter `@NicoleRauch`

Web `http://www.nicole-rauch.de`