- title : Programmieren mit FUN
- description : Slides für die Spartakiade 2015
- author : Carsten König
- theme : beige
- transition : default

Spartakiade

Figure 1: Spartakiade

Programmieren mit FUN

Carsten König

---

ein paar Zitate

Weil: Das Internet hat immer Recht ...

Quelle Quora

> Functional programming is way too impractical.

> It's really difficult to understand and, besides that, just kinda weird.

> No one uses FP in the real world.

> Save yourself some time and headaches and just use *Java* (*C#*).

> Leave functional programming to the pointy-headed ivory-tower types.

Was ist überhaupt eine *funktionale* Sprache?

Panik ...

- **Monaden**
- *Funktoren*
- Catamorphismen
- ...

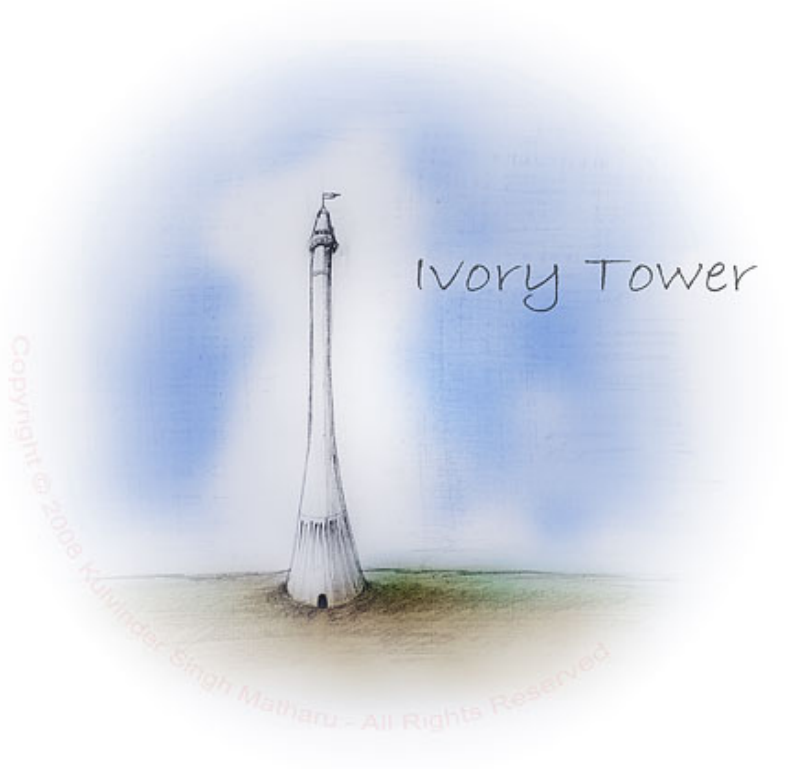

Figure 2: Elfenbeinturm

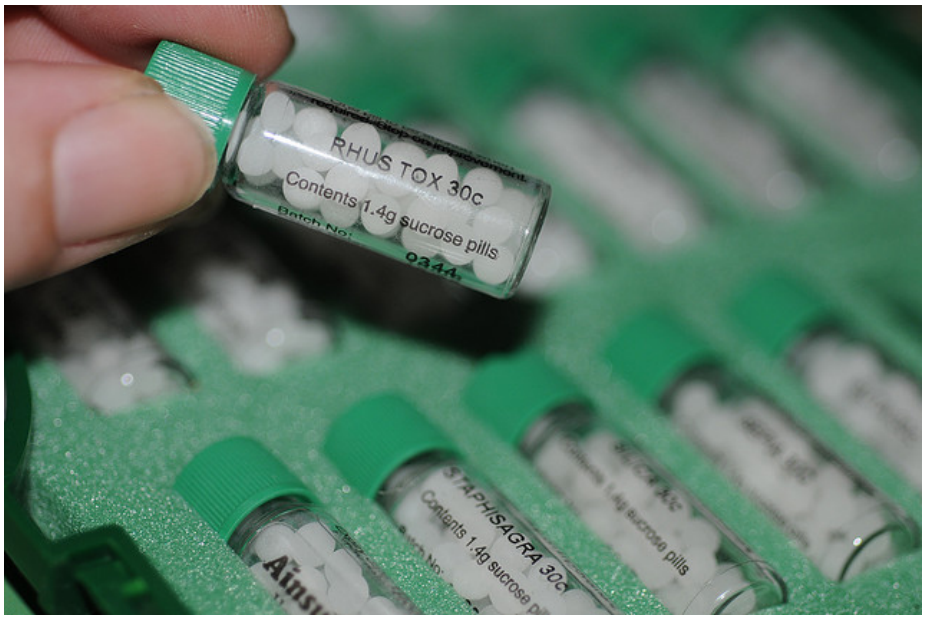

Figure 3: Homeophaty

erinnert mich an

klingt gefährlich ist aber nur Zucker

---

eigentlich geht es nur um

reine Funktionen

```
let f n = n+n
```

und **unveränderliche** Datenstrukturen

```
type 'a List =
    | []
    | (::) : 'a -> 'a List -> 'a List
```

---

aber...

Nun sag, wie hast du's mit den *Typen*?

Figure 4: Gretchenfrage

static VS *dynamic*

mehr Infos

Figure 5: bottom

strict VS *non-strict*

mehr Infos

Vielleicht sogar **total**?

strong VS *weak* functional Programming

mehr Infos

Heute:

Grundlagen in F#

- statisches Typsystem
- non-strict (*lazy* wird aber unterstützt)
- ganz sicher nicht *total*

Der Plan

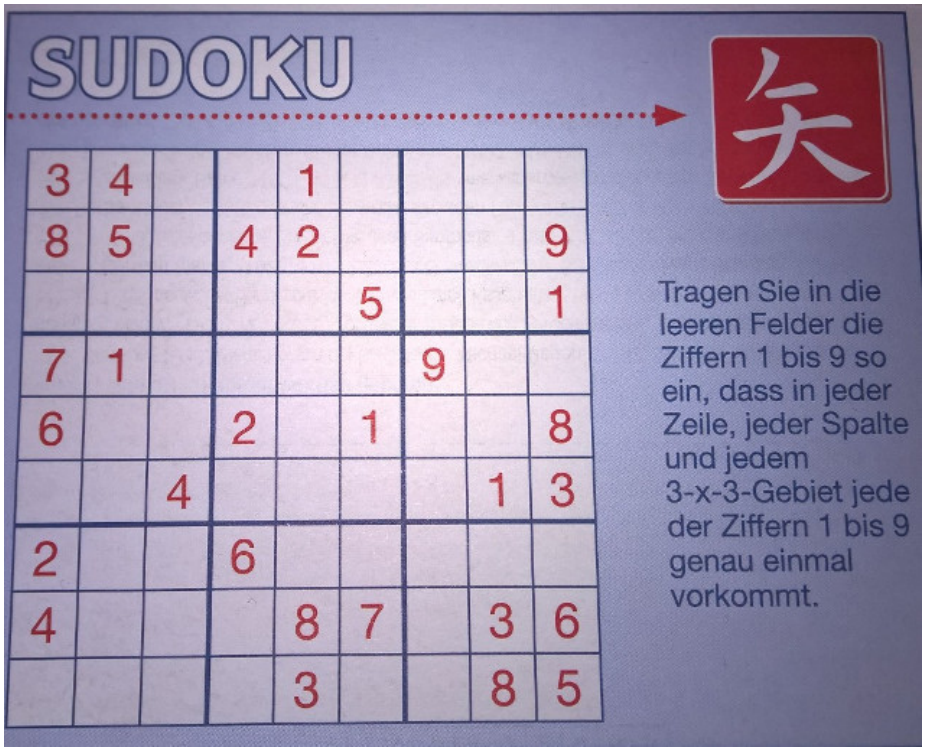

Figure 6: Sudoku

- Einführung in FP und F#
- Viele kleine Übungen

- Vorstellung Sudoku-Projekt
- nach Zeit / Interesse
- Minieinführung CT / Funktoren
- Monaden
- wir spielen TDD

Tools

Grundlagen

Übung

Sind folgende Funktionen gleich?

```
let f a b c = a + b + c
```

und

```
let g a = fun b c -> a + b + c
```

Übung

Implementiere **FizzBuzz**

Siehe `Uebungen/FizzBuzz.fsx`

Übung

Suche Beispiele für

- `bool * unit * char`
- `string * (int * int) * bool`

Schreibe Funktionen

- `first : 'a * 'b -> 'a`
- `second : 'a * 'b -> 'b`
- `curry : ('a*'b -> 'c) -> ('a -> 'b -> 'c)`
- `uncurry`

Übung

Schreibe Funktionen

- `head : 'a list -> 'a`
- `tail : 'a list -> 'a list`
- `third` : soll das 3. Element einer Liste liefert
- Implementiere Dein eigenes *concat*

Übung

Implementiere **Coin Change**

Siehe `Uebungen/CoinChange.fsx`

Übung

- gib einen Typ an, der entweder ein `int*int` Tupel oder nichts enthält
- schreibe eine Funktion `eval : Expr -> int`

Wobei

```
type Expr =
    | Zahl of int
    | Plus of Expr * Expr
```

Übung

Wir *lösen quadratische Gleichungen*

Siehe `Uebungen/Mitternacht.fsx`

Listen falten

Übung

Implementiere eine Funktion `produkt` die das Produkt einer `int list` berechnet.

Übung

Implementiere die Funktion `filter`:

gegeben: Prädikat `p : 'a -> bool` und `ls : 'a list` gesucht: `'a list` mit Elementen `l` aus `ls` mit `p l = true`

Übung

Implementiere jeweils mit `foldR`:

- `and : bool list -> bool`
- `or : bool list -> bool`
- `any : ('a -> bool) -> 'a list -> bool`
- `all : ('a -> bool) -> 'a list -> bool`

---

harte Übung

Definiere `foldL` durch `foldR` (ohne direkte Rekursion)

Sequenzen, Rekursion und Kombinatorik

Übung

Implementiere eine Funktion

`crossProd : ('a list) list -> 'a list seq`

mit

- `[[1;2];[3];[4;5]] -> { [1;3;4]; [1;3;5]; [2;3;4]; [2;4;5] }`
- `[[1;2];[];[4;5]] -> { }`

Übung

Implementiere eine Funktion

`permutationen : 'a list -> 'a list seq`

Übung

Implementiere `zip : 'a list -> 'b list -> ('a*'b) list` mit `Seq.unfold`

Beispiel: `zip [1;2;3] ['x'; 'y'] = seq [(1,'x'), (2,'y')]`

Übung

Pascals Dreieck

Siehe `Uebungen/Pascal.fsx`

Sudoku