Functional Programming

in JavaScript

Ein Programmier-Paradigma

- Ein Stil zum Schreiben von Code
- Ein Mindset beim Lösen von Aufgaben
- Abhängig vom Design der Programmiersprache

Was?

- Fokus auf Daten Transformation
- In Funktionen denken
  - Eingaben entgegenehmen (Argumente)
  - Ausgabe zurückliefern (return Statement)

Warum?

Weniger Bugs

- Weniger Code
- Kleinere logische Einheiten => verständlicher
- Keine Seiteneffekte

Weniger Zeit

- Code ist wiederverwendbar

Pure Functions

Keine externen Abhängigkeiten

Alle Daten müssen als Argument übergeben werden

Keine Seiteneffekte

Funktion gibt etwas zurück (return Statement)

=> Mit der selben Eingabe immer zum selben Ergebins kommen

"Functions in JS are First Class Citizens"

Funktionen sind Werte

(wie String, Number, Boolean, Array)

- Können in Variablen gespeichert werden
- Können als Argument an eine andere Funktion gegeben werden
- Können der Rückgabewert von einer anderen Funktion sein

Higher Order Functions

Funktionen, die mit anderen Funktionen zusammenarbeiten

- Funktion als Argument oder Rückgabewert
- Erlaubt Abstraktion von Aktionen

Array Functions: classics

Funktion zur Transformation von Arrays

- .filter()
- .find()
- .map()

Array Functions: .reduce()

Wenn vorhandene Array Funktionen nicht weiterhelfen

- Multi-Tool der Array Transformation
- Implementierung jeder beliebigen Array Transformation

Function Composing

Funktionen als Argument oder Rückgabewert

- Erlaubt Komposition
  - Viele kleine, leicht verständliche Funktionen
  - Jeweils genau eine Aufgabe
- Statt einer großen, komplexen, unübersichtlichen Funktion

Recap

- Pure Functions
- Funktionen wie Variablen behandeln
- Higher Order Functions
- Array Funktionen
- Function Composition