# Funktionale Programmierung

## Teil 1: Hallo, willkommen, etc.

# Organisatorisches

## Vorbemerkungen

- Kurs über Konzepte und Grundlagen der funktionalen Programmierung
- Haskell als “Arbeitssprache”
- Keine Vorlesung über “die Sprache Haskell”
- Unterlagen und weitere Informationen: Gitlab

| Zweigeteilter Lektionsablauf | |
|---|---|
| Theorie | Praktikum |
| ■ “Frontalunterricht” mit möglichst viel “live coding”<br>■ Basiert auf Folien (Gitlab)<br>■ Auflockerung durch Programmieraufgaben | ■ Theorieaufgaben, Programmieraufgaben und einiges dazwischen<br>■ Mini Projekte zum selber machen und/oder erweitern |

- Die Kursnote ergibt sich aus der Semesterendprüfung
- Die Prüfung beinhaltet Theoriefragen und praktische Programmieraufgaben
- Die Prüfung wird “elektronisch” (am Notebook) geschrieben

# Administrative und organisatorische Fragen?

Beispiel (Elemente einer Liste aufsummieren):

```
1  sum(L){
2      l = length(L)
3      i = 0
4      acc = 0
5      while (i < l){
6          acc = acc + L[i]
7          i = i + 1
8      }
9      return acc
10 }
```

- Basiert auf Zuständen, Zustandsübergängen und deren zeitlichen Abfolge.
- Ein imperativer Algorithmus beschreibt, wie ein Problem Schritt-für-Schritt gelöst wird.
- Probleme werden in geeignete Arbeitsschritte zerlegt, welche in einer bestimmten Reihenfolge abgearbeitet werden müssen.

Beispiel (Elemente einer Liste aufsummieren):

- "Mathematisch":

$$sum(\varnothing) = 0$$
$$sum(\{a_1, \ldots, a_n\}) = a_1 + sum(\{a_2, \ldots, a_n\})$$

- Funktional (Haskell):

```haskell
sum [] = 0
sum (a1:rest) = a1 + sum rest
```

- Funktional (F#)

```fsharp
let sum = function
    | [] -> 0
    | a1::rest -> a1 + sum rest
```

- Nahe an der mathematischen Notation (Funktionen sind mathematische Objekte)
- Definition/Beschreibung einer Funktion anstelle expliziter Anweisungen (deklarativ)
- Keine “Variablen” (im Sinn von Veränderlichen). Variablen bezeichnent Werte und nicht Speicherbereiche.

*Functional programming is a style of programming that emphasizes the evaluation of expressions, rather than execution of commands. The expressions in these languages are formed by using functions to combine basic values.*
*Graham Hutton*

- 1930 - 1938 (Church, Kleene, Rosser): Theoretische Fundierung der funktionalen Programmierung durch die Entwicklung des $\lambda$-Kalküls.
- 1958 (McCarthy): Einführung von LISP.
- 1966 (Landin): Mit *The next 700 programming languages* erscheint ein visionäres Papier in dem viele moderne funktionale Sprachkonzepte vorweggenommen werden (z.B. algebraische Datentypen).
- 1977 (Backus): Einführung der Sprache FP, "Function-level programming".
- 1970er: Stark typisierte funktionale Sprachen. ML (Meta Language) als Teil eines Theorem-Beweis-Programms ("Logic of Computable Functions").

- 1983-86 (Turner): Miranda als frühe Sprache mit “lazy evaluation”, algebraischen Datentypen und polymorphem Typsystem à la Milner.
- Neuere Entwicklungen: Funktionale Sprachen für Mainstream “Plattformen” wie JVM (Scala, Clojure), .Net (F#) und Web (clojurescript, purescript, elm, . . . ) sowie die Übernahme funktionaler Konzepte in Mainstream Sprachen (lambdas, closures, continuations, . . . )
- Weiteres zur Geschichte auf Gitlab hier

Keine feste “Definition”. Einige für die funktionale Programmierung zentrale Konzepte können aber identifiziert werden:

- Vollwertige Unterstützung von Funktionen (first class functions): Funktionen höherer Ordnung, $\lambda$-Terme, Currying und partielle Anwendung, ...
- Unveränderliche (immutable) Daten und daraus folgend: Referenzielle Transparenz, Typsicherheit (type safety),...
- Hoch entwickeltes Typensystem: Algebraische Datentypen, Pattern Matching, Typinferenz,...
- Rekursion: Tail-Call-Optimization

Alternativ lässt sich funktionale Programmierung auch sprachunabhängig als “Programmierstil” oder eine Menge von “Patterns” definieren.

- Vermeiden von Seiteneffekten und veränderlichen Zuständen
- Komponierbarkeit anstreben (“transform-oriented-programming”)
- Höhere Funktionen und Rekursion der Iteration vorziehen.
- ...

Funktionale Sprachen sind Programmiersprachen, die den funktionalen Stil erleichtern oder, im Fall rein funktionaler Sprachen, sogar (weitgehend) erzwingen.

Einige modernere Sprachen nach “Funktionalität geschichtet”:

- “Rein funktional”: Idris, Haskell, Frege, Miranda, ...
- “Functional first”: Nemerle, Erlang, ML (SML, OCaml, F#), ...
- “Spezialfall Lisp-Familie”: Scheme, Clojure, ...
- “Funktional inspiriert”: Julia, Scala, Rust, ...

Anwendungen:

- Klassisch: Compilerbau, Theorembeweiser, KI
- Verbreitet: Numerik, Data science, Fintech
- Auch sonst ist alles, bis hin zu GUI Programmierung (“functional reactive programming”), möglich (und sinnvoll).

Funktionale Programmierung...

- ...ist (momentan?) in Mode.
- ...nimmt immer mehr Einzug in traditionelle Sprachen.
- ...ist in der Industrie gefragt (und wird gut bezahlt).
- ...**macht spass!**.

Gründe, die für Haskell als Unterrichtssprache sprechen:

- Eine rein funktionale Sprache, damit Sie gezwungen werden die neuen Konzepte anzuwenden.
- Plattformunabhängig (Linux, Windows, BSD, Mac) und open-source.
- Produktiv eingesetzte Sprache.
- **Wunsch der Studierenden.**

- `https://docs.haskellstack.org/en/stable/README/`
- VS Code + irgend ein Plugin für Haskell Syntax.

## Aufgabe

- Installieren Sie Haskell.
- Starten Sie den REPL mit dem Befehl `stack ghci`
- Geben Sie `2 + 1` ein.

- Grundsätzlich werden wir während dem ganzen Kurs (mit wenigen Ausnahmen) mit eher kleinen Skripten arbeiten.
- Wenn Sie Beispiele implementieren oder eigenen Code schreiben, dann nutzen Sie den REPL!
- Die Folien folgen der Notation:

```haskell
-- hier steht Code
x :: Integer
x = 5
```

```
-- hier steht eine Rueckmeldung von ghci
5
```

- Erstellen Sie ein File, z.B. `test.hs` mit dem Inhalt

```
greetings :: String
greetings = "hello world!"
```

- Navigieren Sie in den Ordner in dem sich `test.hs` befindet.
- Starten Sie den Repl mit `stack ghci`.
- Laden Sie das `test.hs` in den REPL und führen Sie das "Programm" aus:

```
Loaded GHCi configuration ...
Prelude> :l start.hs
[1 of 1] Compiling Main ...
Ok, one module loaded.
*Main> greetings
"hello world!"
*Main>
```

Folgend ist eine kleine Einführung in die Haskell Syntax. Unsere Lernziele sind Grundkenntnisse der Haskell Syntax, insbesondere:

- Einfache Werte und Typen
- Funktionen deklarieren
- Grundlegende Kontrollstrukturen und “syntactic sugar”
- Einfache mehrstellige und höhere Funktionen
- Deklarieren von eigenen Typen

Bei all diesen Themen werden wir im Verlauf der Vorlesung noch ins Detail gehen. Momentan geht es darum, dass wir die grundlegendsten syntaktischen Elemente von Haskell kennen und “bedienen” können.