universität innsbruck

WS 2025/2026

Funktionale Programming

Woche 1 – Organisation und Einführung

René Thiemann
Philipp Dablander Joshua Ocker Michael Schaper Lilly Schönherr Adam Pescoller

Institut für Informatik

Organisation

Vorlesung (VO 2)

- LV-Nummer: 703024
- Dozent: René Thiemann
  Sprechstunde: Dienstags 10:15 – 11:15 in Raum 3M09 (ICT Gebäude)
- Zeit und Ort der VO: Montags, 12:00 – 13:30 in HS B
- Webseite zum Kurs:
  `http://cl-informatik.uibk.ac.at/teaching/ws25/fp/`
- Sprache: Deutsch mit englischem Quelltext
- Folien gibt es online und enthalten Links
- online Registrierung erforderlich bis zum 31. Januar, 2026
- Vorlesung wird aufgezeichnet; Videos sind in OLAT-VO verfügbar

Zeitplan

Vorlesung 1	Oktober	6	Vorlesung 8	November	24
Vorlesung 2	Oktober	13	Vorlesung 9	Dezember	1
Vorlesung 3	Oktober	20	Vorlesung 10	Dezember	15
Vorlesung 4	Oktober	27	Vorlesung 11	Januar	12
Vorlesung 5	November	3	Vorlesung 12	Januar	19
Vorlesung 6	November	10	Vorlesung + Q & A	Januar	26
Vorlesung 7	November	17			

- Vorlesung am 26. Januar
  - Inhalt ist nicht Klausur-relevant
  - offene Fragestunde

Proseminar (PS 1)

- LV-Number: 703025
- wöchentliche Übungsblätter werden spätestens Mittwochs online gestellt
- Lösungen müssen in OLAT-PS hochgeladen werden
  - Markierung aller gelöster Aufgaben mittels Kreuzliste
  - Hochladen der Quelltexte bei Programmier-Aufgaben
  - Frist: 6 Uhr morgens am Mittwoch vor den Proseminaren
- Lösungen werden in Proseminar Gruppen präsentiert
- Übungsblatt 1 wird bis spätestens Mittwoch verfügbar sein
- Beginn der Proseminare: 8. oder 15. Oktober
- 8. Oktober
  - freiwillige Teilnahme
  - Besprechung von Übungsblatt 0 (Einführung in die Nutzung eines Haskell-Compilers, OLAT)
- Teilnahme an Proseminaren ist ab dem 15. Oktober verpflichtend
- separate Anmeldung für Proseminare erforderlich, offizielle Anmeldefrist im September

Proseminar Gruppen

- momentan gibt es 8 Gruppen, siehe auch LFU online
- Gruppen 1–5 sind bereits voll, in Gruppen 6–8 gibt es noch freie Plätze
  - noch nicht angemeldet? → kontaktieren Sie mich in der Pause oder nach der Vorlesung
- Gruppenwechsel sind nur mittels SWAp-Tool möglich
  - es gab eine Willkommens-Nachricht in OLAT-PS mit Details zum SWAp-Tool
  - optionalen Wünsche eintragen, um anderen Studierenden zu helfen
  - ein paar Wechsel wurden bereits durchgeführt
  - nächste Durchführung von Gruppenwechseln: heute, 6. Oktober, 16 Uhr

Tutorium

- Gelegenheit, um über Themen aus VO und PS Fragen zu stellen
- Präsentation weiterer Beispiele
- keine neuen Themengebiete, kein Einfluss auf die Note, keine Lösungen zu Übungsblättern
- Teilnahme ist freiwillig
- Tutor: Adam Pescoller
- Dienstags 17:15 – 18:00 im HS 11
- Beginn: 7.10.

Wöchentlicher Ablauf

- Montag 12:00 – 13:30: Vorlesung über Thema n
- Dienstag 17:15 – 18:00: Tutorium zu Thema $n-1$ oder n
- Mittwoch 06:00: Frist zum Hochladen und Kreuzen von Übungsblatt $n-1$
- Mittwoch: Proseminare zur Besprechung von Übungsblatt $n-1$
- bis Mittwoch: Übungsblatt n über Thema n verfügbar
- ...

Benotung

- Separate Noten für Vorlesung und Proseminar
- Vorlesung
  - Note wird nur durch Klausur bestimmt
  - Klausur: schriftlich auf Papier, ohne Unterlagen (closed book)
  - 1. Klausurtermin: 2. Februar 2026
  - separate Klausur-Anmeldung vom 29. Dezember bis 18. Januar mittels LFU online (Abmeldung ist auch bis 3 Tage vor der Klausur möglich)
  - 2. Klausurtermin: März oder April 2026
  - 3. Klausurtermin: September 2026
  - es reicht aus, eine der Klausuren zu bestehen
- Proseminar
  - 80 %: Bearbeitung der wöchentlichen Übungsblätter
  - 20 %: Präsentation der Lösung

Chat-GPT

- Chat-GPT und ähnliche KI-basierte Systeme sind in der Lage, funktionale Programme für leichte Aufgaben automatisch zu generieren
- positive Aspekte der Nutzung von Chat-GPT
  - Sie können das Proseminar bestehen, ohne selber programmieren zu können
  - Sie können Unterstützung bekommen, wenn Sie an manchen Stellen keine Lösung finden
- negative Aspekte der Nutzung von Chat-GPT
  - wenn Sie nicht selber programmieren können, haben Sie keine Chance, die Klausur zu bestehen
  - wenn Sie an gewissen Stellen nicht weiterkommen, dann können Sie auch mit anderen Studierenden diskutieren
- Fragen Sie sich, warum Sie Informatik studieren?
  - um selber programmieren zu lernen?
  - oder um zu lernen, wie man Chat-GPT nutzt, um einfache Programmieraufgaben zu lösen?
- Fazit: von der Nutzung von Chat-GPT usw. wird abgeraten (für diesen Kurs)

Literatur

Folien

- es werden keine anderen Themen in der Klausur vorkommen ...
- ...aber diese Themen sollten gründlich verstanden werden
  - Lesen und Schreiben von funktionalen Programmen
  - Anwendung der gelernten Techniken auf neuen Beispielen
  - es reicht nicht aus, die Folien auswendig zu lernen

Richard Bird. Introduction to Functional Programming using Haskell, 2nd Edition, Prentice Hall.

Einführung

(Funktionale) Programming

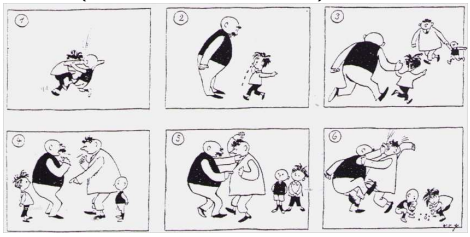
- Aufgabe: Lösen Sie eine Problemstellung
  - sortiere eine Liste von Werten
  - generiere eine Webseite
  - navigiere von Innsbruck nach Köln
- Unterscheiden Sie zwischen Daten ...
  - Eingabe `[26,52,13]` und Ausgabe `[13,26,52]`
  - Anfrage "Was ist funktionale Programmierung?" und der generierten Webseite
  - Karte von Europa sowie zwei Orte, und ein Weg auf der Karte
- ... und Programmen
  - diese kontrollieren, wie die Daten verarbeitet werden
  - werden meist von Menschen geschrieben
- Computer werden genutzt, um ein Programm auf Eingabe-Daten auszuführen, um die Ausgabe-Daten zu erhalten, aber prinzipiell kann diese Berechnung auch auf Papier oder im Kopf durchgeführt werden

Wie man Programmieren lernt

+ lesen, verstehen und schreiben Sie (viele) Programme
+ nehmen Sie aufmerksam an Vorlesung und Proseminar teil
+ versuchen Sie die Übungsaufgaben zu lösen (alleine oder in kleinen Gruppen)
– kopieren Sie Lösungen von anderen Studierenden oder aus dem Internet

Algorithmen und Programme

Story (Sprach-unabhängig)



Text (Sprach-abhängig)

- Thomas und Paul rauften solange bis ...
- Tom and Paul were struggling until ...
- 토마스와 파울은 싸우고 있었는데...

Algorithmus (Progr.-Sprachen unabh.)

- Problem: bestimme das Maximum von m und einer Liste von Zahlen
- wenn die Liste leer ist, dann ist das Resultat m
- sonst ändere m auf $max(m, n)$, wobei $n =$ erstes Element der Liste
- mache mit dem Rest der Liste weiter

Programm (Programmier-Sprachen abhängig)

- ```
  maxlist m [] = m
  maxlist m (x : xs) =
    maxlist (max m x) xs
  ```
- ```
  while (list != null) {
    m = max(m, list.head);
    list = list.next; }
  return m;
  ```

Arten von Programmiersprachen (Programmierparadigmen)

- Imperative Programming (VO Einführung in die Programming)
  - Zustand ist eine Abbildung von Variablen auf Daten
  - Wertzuweisungen modifizieren den Zustand
  - Beispiel
    - Betrachten Sie eine Wertzuweisung `x := (x + y) / 2`
      – wenn im aktuellen Zustand `x` den Wert 7 speichert und `y` den Wert 3,
      dann speichert nach der Wertzuweisung `x` den Wert 5, und `y` immer noch 3
- Funktionale Programming (diese Vorlesung)
  - Programme definieren mathematische Funktionen
    (mathematisch: gleiche Eingabe liefert gleiches Resultat)
  - Funktionsaufrufe werden nach-und-nach ausgewertet, ohne Zustandsänderungen
  - Beispiel
    - schreibe Funktions-Definition `average x y = (x + y) / 2` mit zwei Variablen `x` und `y`
    - Auswertungsmechanismus: ersetze Gleiches durch Gleiches (linke Seite durch rechte Seite)
    - Beispiel-Auswertung: `average 7 3 = (7 + 3) / 2 = 10 / 2 = 5`
      – hier wird beim ersten Schritt `x` mit `7` und `y` mit `5` substituiert
      – es gibt keine Zustand mit Variable `x`, der von 7 auf 5 geändert wird
- Logik-Programming, Objekt-Orientierte Programming, ...

Unterschiedliche Programmierparadigmen

- Fakt: die meisten Programmiersprachen sind gleich mächtig
- trotzdem gibt es Bedarf für unterschiedliche Sprachen
  - jede Sprache hat besondere Vorzüge aber auch Einschränkungen
    (ähnlich zu normalen Sprachen: übersetzen Sie "Ohrwurm" und "Internetbrowser")
  - gute ProgrammiererInnen sollten Alternativen kennen:
    wählen Sie geeignete Sprachen abhängig vom Problem und vom Kontext
- Vorteile der funktionalen Programmierung
  - intuitiver Auswertungs-Mechanismus
  - gut geeignet für die Verifikation
  - ausdrucksstarke Sprachkonzepte
  - parallele Auswertung ist leicht(er) möglich
- Nachteile der funktionalen Programmierung
  - aufwändiger, Probleme zu lösen, die Zustände benötigen oder I/O durchführen
  - rein funktionale Sprachen sind selten, aber FP-Konzepte werden populärer
    (Scala, Python, F#, Rust, Clojure, Kotlin, ...)

Funktionale Programmiersprachen (FP)

- Kombinatorische Logik (Moses Schönfinkel 1924, Haskell Curry 1930): Grundlage der FP
- λ-Kalkül (Alonzo Church 1936): Grundlage der FP
- LISP (John McCarthy, 1958): List Processing (Ableger: Clojure)
- ML (Robin Milner, 1973): Meta Language (Ableger: OCaml, SML)
- Erlang (Ericsson, 1987): verteilte Programmierung (WhatsApp)
- Haskell (Paul Hudak und Philip Wadler, 1990): (Sprache in diesem Kurs)
- F# (Microsoft, 2002) und Scala (Martin Odersky, 2003) und Rust und Kotlin und ...: Multi-Paradigmen Sprachen, die FP-Konzepte inkludieren

Syntax and Semantik

- Syntax einer (Programmier-)Sprache definiert gültige Sätze (Programme)
  - "Dies ist ein gültiger deutscher Satz."
  - "Dieser nit gültich sein"
  - Computer lehnen Programme ab, die syntaktische Fehler beinhalten!
- Semantik definiert die Bedeutung von gültigen Sätzen / Programmen
  - "Räum dein Zimmer auf!" ✔
  - `let xs = 1 : 1 : zipWith (+) xs (tail xs) in take 9 xs` ✘
- Wir werden sowohl Syntax als auch die Semantik von Haskell kennenlernen

Haskell Skripts

```
-- This script is stored in file Script_01.hs

average x y = (x + y) / 2

{- the following function takes a temperature in
   degree Fahrenheit and converts it into Celsius -}

fahrenheitToCelsius f = (f - 32) * 5 / 9
```

- ein Haskell Skript (= Programm = Code = Quelltext = Source) hat die Dateiendung `.hs`
- ein Skript besteht aus einer Anzahl von Funktions-Definitionen
- Kommentare sind wichtig für Menschen, haben aber keine semantische Bedeutung
- Haskell unterstützt einzeilige und mehrzeilige Kommentare
  - einzeilig: `-- everything right of -- is a comment`
  - mehrzeilig:
    ```
    {- multi-line comments can deactivate
       areaRectangle width height = width * height
       parts of script easily -}
    ```

Erstellung von Haskell Skripts

```haskell
-- This script is stored in file Script_01.hs
average x y = (x + y) / 2
fahrenheitToCelsius f = (f - 32) * 5 / 9
```

- Farbe
  - Bei der Programm-Erstellung färbt man Text nicht manuell ein
  - automatisches Syntax Highlighting färbt Computer-Programme selbständig ein; Vorteil: Farben vereinfachen das Lesen der Programme; man erkennt direkt
    - Kommentare, Schlüsselworte, Funktionsnamen, Variablennamen, ...
- in Haskell: Namen von Funktionen und Variablen (`average`, `x`, ...)
  - beginnen immer mit einem Kleinbuchstaben, dürfen auch Zahlen beinhalten
  - Konvention: lange Namen nutzen camelCase (`fahrenheitToCelsius`, ...)
- Leerraum (Leerzeichen, Tabulator, Zeilenumbruch, ...)
  - in Haskell ist die Struktur des Leerraums wichtig
  - momentane Richtline: beginnen Sie jede Zeile ohne Leerraum am Anfang
  - Beispiel: das Programm unten ist syntaktisch falsch

```haskell
average x y = (x + y) / 2
 fahrenheitToCelsius f = (f - 32) * 5 / 9
```

Haskell Sitzungen

- eine Haskell Sitzung zu starten ist wie einen Taschenrechner einzuschalten
- wir nutzen ghci, den Glasgow Haskell Compiler im Interpreter Modus

```
rene$ ghci                          -- start the interpreter
Prelude> 42                         -- enter a value
42
Prelude> 5 * (3 + 4)                -- evaluate an expression
35
Prelude> :load Script_01.hs         -- load script from file
[1 of 1] Compiling Main  ( script_01.hs, interpreted )
Ok, 1 module loaded.                -- script was accepted
*Main> fahrenheitToCelsius 95       -- invoke our function
35.0
*Main> :quit
```

Ablauf bei der Erstellung eines Funktionalen Programms

- Definiere Funktionen in einem Haskell Skript
- Lade das Skript mit `ghci` (Erfolg oder Fehlermeldung)
  - Fehler beim Parsen: `5 + (3 * 7` (schliessende Klammer fehlt)
  - Fehler bei Typprüfung: `5 + "five"` (Zahlen und Texte können nicht addiert werden)
  - Fehlermeldungen können manchmal verwirrend sein: `True + False`, `if 0 then 1 else 2`
- Wiederholte Eingabe von Ausdrücken mit Auswertung (read-eval-print loop, REPL)
  - Resultat: ein Wert, der nicht weiter vereinfacht werden kann, z.B., `42`, `"hello"`, `[7,1,3]`, ..., aber nicht `5 + 7`, `fahrenheitToCelsius 8`, ...
  - Auswertung nutzt
    - eingebaute Funktionen (`+`, `*`, `:`, `++`, `head`, `tail`, ...), alle in Prelude definiert
    - Benutzer-definierte Funktionen (`fahrenheitToCelsius`,...) aus Haskell Skripts

Vergleich zwischen FP und einem Taschenrechner

- Taschenrechner nimmt ebenfalls Ausdrücke entgegen und berechnet das Resultat
- Taschenrechner ist eingeschränkt auf Zahlen und auf eingebaute Funktionen

Vergleich: FP gegen Taschenrechner

- Aufgabe: konvertiere viele Temperaturen von Fahrenheit nach Celsius: 8, 9, 300, ...
- Lösung Taschenrechner:
  - $(8 - 32) * 5/9$
  - $(9 - 32) * 5/9$
  - $(300 - 32) * 5/9$
  - ... (mühsam: mehrfache Eingabe der immer gleichen Berechnung)
- FP
  - Erstelle ein Programm: `fahrenheitToCelsius f = (f - 32) * 5 / 9`
  - werte nun die Funktionen auf den gewünschten Eingaben aus
    - `fahrenheitToCelsius 8`
    - `fahrenheitToCelsius 9`
    - `fahrenheitToCelsius 300`
    - ... (präzise, lesbar, und einfach: nur Funktionsaufrufe)
  - noch effizienter mittels `map`: `map fahrenheitToCelsius [8,9,300,...]`
- Funktionales Programm: ein Rezept, um Eingaben in Ausgaben zu verwandeln.

Zusammenfassung

- Haskell Programme werden in .hs-Dateien gespeichert
- funktionale Programmierung: Definition von Funktionen durch Gleichungen
- `ghci` lädt Programme und wertet dann Ausdrücke aus (REPL)
- nächste Vorlesung: Daten $\supsetneq$ Zahlen – strukturierte Daten