

Funktionale Programmierung

Thanh Viet Nguyen

7. Dezember 2024

- Was ist ein Funktionale Programmierung?
- Ursprung
- Unterschiede zur imperativen Programierung
- Historische Entwicklung
- Wichtige Konzepte: Funktionen als erste Klasse, Unveränderlichkeit und Rekursion
- Praktische Beispiele
- Ziel der Präsentation: grundlegige Verständnis zur funktionalen Programmierung

- Lambda Kalkül (1930): Alonzo Church (14.06.1903 - 11.08.1995)
- Vor Maschinen, die solchen Code ausführen konnten, gab es bereits Ansätze dazu.
- Typen Theorie

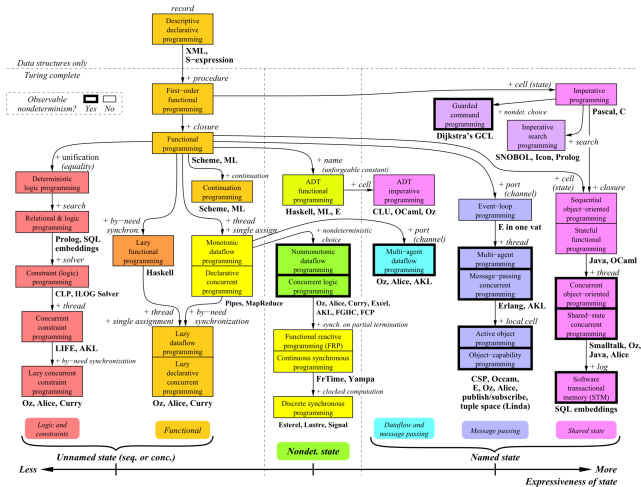
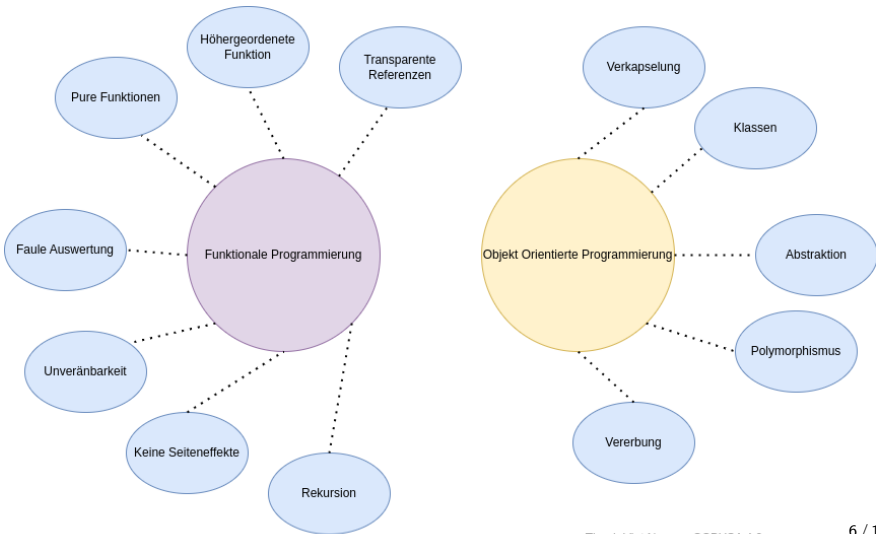


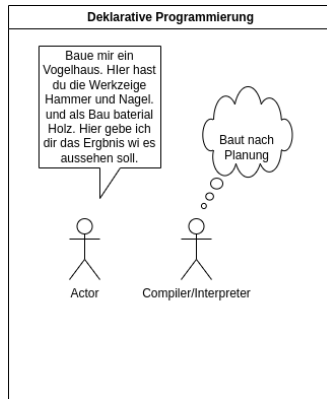
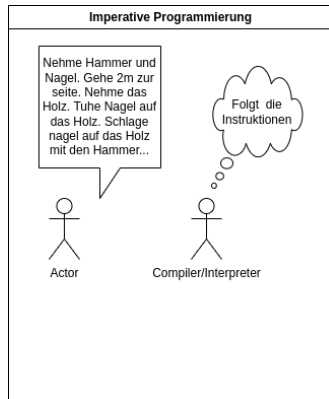
Figure 2. Taxonomy of programming paradigms

<https://blog.acolyer.org/2019/01/25/programming-paradigms-for-dummies-what-every-programmer-should-know/>

Objekt Orientierung und Funktionale Programmierung

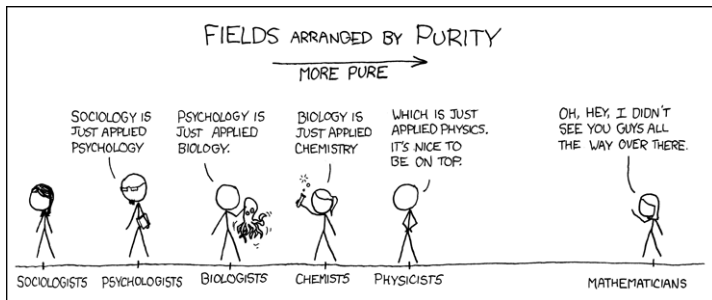


Deklarative und Imperativen Programmierung



Thanh

Viet Nguyen CCBYSA 4.0



<https://xkcd.com/435/>

- Formales System in der Mathematik für Aussagen bearbeitung
-

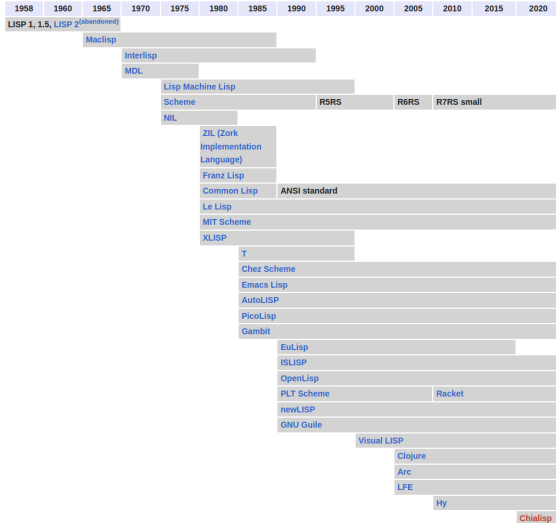
Lisp Maschinen

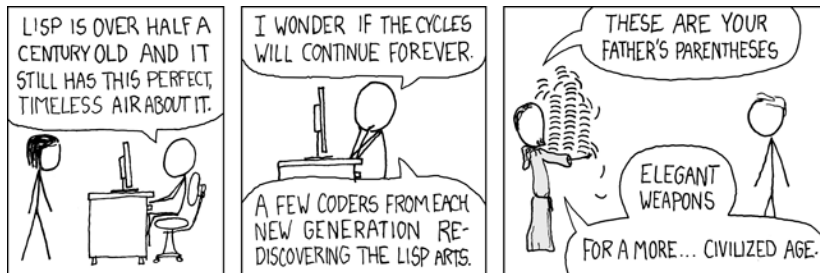
- 1960er
- Zweck rforschung von Künstlicher Intelligenz



content...

Lisp Dialekte





https://www.explainxkcd.com/wiki/index.php/297:_Lisp_Cycles

- Fokus auf unveränderbare Daten
- Wenn eine Funktion pur ist, folgt: Kein State -> keine Seiteneffekte
- Basiert sich auf Aussagen
- Referenzen sind Transparent: `input` -> `output`

Zusätzliche Konzepte


- Currying: Umwandlung von Funktionen, die mehrere Argumente annehmen, in eine Kette von Funktionen, die jeweils ein Argument akzeptieren.
- Functor:
 - Functor: Beschreibt, wie Funktionen mit `fmap`", auf Werte in einem bestimmten Zusammenhang angewendet werden.
 - Befolgt Identitätsregel und Kompositionsregel.
- Applificat:
 - Verwendet die Funktionen `pure` oder `return`", um Werte in den Kontext zu bringen
 - Nutzt den Operator `»>=`", um Berechnungen zu verknüpfen
 - Befolgt Identitäts- und Assoziativitätsregeln.
- Monad:
 - Verknüpft Berechnungen in einem Kontext.
 - Definiert durch `return` oder `pure` und `»>=`"(Bind).
 - Befolgt Identitätsregel und Assoziativitätsregel.


Currying

```
... codebsp > » Curry.hs
1  |-- Curry
2  add :: Int -> Int -> Int
3  add x y = x + y
4  main :: IO ()
5  main = do
6      let result = add 3 5
7      putStrLn ("Curry Addition:" ++ show result)
8
```

```
... codebsp > Curry.java
1  import java.util.function.Function;
2
3  public class Curry {
4      public static Function<Integer, Function<Integer, Integer>> add =
5          x -> y -> x + y;
6
7      public static void main(String[] args) {
8          int result = add.apply(3).apply(5);
9          System.out.println("Curry Addition Java:" + result);
10     }
11 }
12
```


Functor

```
... codebsp >  Functor.hs
1  -- Functor
2  import Data.Functor
3  -- Beispiel mit Maybe
4  maybeWert :: Maybe Int
5  maybeWert = Just 5
6  incrementedWert :: Maybe Int
7  incrementedWert = fmap (+1) maybeWert
8  main :: IO ()
9  main = do
10     let result = incrementedWert
11     putStrLn (show result)
12  |
```

```
... codebsp >  Functor.java
1  import java.util.Optional;
2
3  public class Functor{
4      public static void main(String[] args) {
5          Optional<Integer> maybeValue = Optional.of(1);
6          Optional<Integer> incrementedValue = maybeValue.map(x -> x + 1);
7          System.out.println(incrementedValue);
8      }
9  }
10
```

Applicative

```
... codebsp > »: Applicative.hs
1  import Control.Applicative
2  -- Beispiel mit Maybe
3  maybeA :: Maybe Int
4  maybeA = Just 3
5  maybeB :: Maybe Int
6  maybeB = Just 5
7  result :: Maybe Int
8  -- Anwendung von pure und Verknüpfung
9  result = pure (+) <*> maybeA <*> maybeB
10 main :: IO()
11 main = do
12     print result
13 |
```

```
... codebsp > 📄 Applicative.java
1  import java.util.Optional;
2
3  public class Applicative{
4      public static void main(String[] args) {
5          Optional<Integer> maybeA = Optional.of(3);
6          Optional<Integer> maybeB = Optional.of(5);
7          // Anwendung von map und flatMap
8          Optional<Integer> result = maybeA.flatMap(a ->
9              maybeB.map(b -> a + b));
10         System.out.println(result);
11     }
12 }
13
```

Monad

```
... codebsp > » Monad.hs
1  import Control.Monad
2  -- Beispiel mit Maybe
3  maybeValue :: Maybe Int
4  maybeValue = Just 10
5  -- Verwendung von bind ">=>"
6  ergebnis = maybeValue >=> \x -> Just (x * 2)
7
8  main :: IO ()
9  main = do
10     print ergebnis
11
```

```
... codebsp > » Monad.java
1  import java.util.Optional;
2
3  public class Monad {
4      public static void main(String[] args) {
5          Optional<Integer> maybeValue = Optional.of(10);
6          // Verwendung von flatMap
7          Optional<Integer> result = maybeValue.flatMap(x -> Optional.of(x * 2));
8          System.out.println(result);
9      }
10 }
```

Vorteile	Nachteile
Mehr Sicherheit beim Entwickeln: Nebenläufigkeit, Wartbarkeit, und Testbarkeit	Kein Zustand -> Keine Zustandskontrolle
Modularität Parallele Verarbeitung	
Mehr Speichersicherheit	meist Speicher intensiver bei der Laufzeit
Code ist in der Regel lesbarer und Kürzer Höhere Abstraktion	Lesbarkeit \neq Verständlich

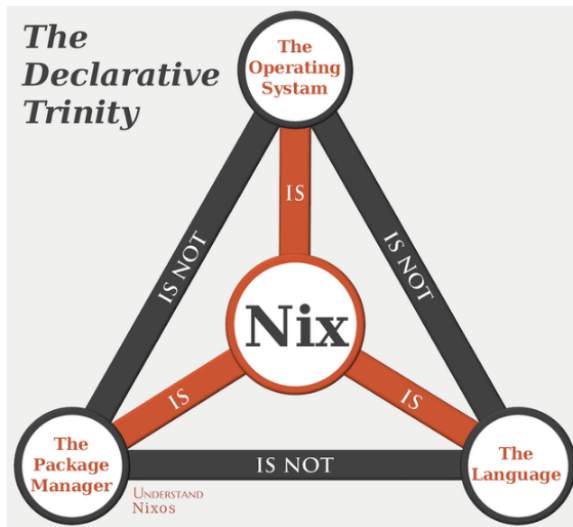
- Software Anwendungen (Webanwendung, Desktopanwendungen, Services, etc.)
 - Sicherstellung von Software
- Systemkonfigurationen
- Beweisasistent für Mathematische Theoreme
- Verarbeitung von Forschungsdaten

Beweisassistent für Theoreme

- Roqc für Lisp Dialekt Liebhaber
- Lean für nicht Lisp Dialekt Liebhaber

Deklarative Systemkonfigurationen und Reproduzierbare Bauten

- NixOS: GNU/Linux Distro
- Guix: GNU/Linux Distro, welches mit Guile Scheme Dialekt Konfiguriert wird
- Emacs: Ein Texteditor mit vielen erweiterungen, welches mit Lisp Dialekt (Elisp)
- Debian GNU/Linux verwendung von OCAML für Reproduzierbare Checks
- Code als Basis für Infrastrukturen



<https://github.com/gytis-ivaskevicius/high-quality-nix-content/tree/master/memes>

- Grundverständnis zur Funktionalen Programmierung
- Lambda Kalkulus etwas kennen gelernt
- Schwächen und Stärken der Funktionalen Programmierung
- Anwendungsgebiete die von FP geführt werden
- Wird in der Zukunft häufiger verwendet durch Entwickeln mit KI und Quantencomputer
- FP ist nicht die Lösung für alle Probleme in der Softwareentwicklung.

Mehr erfahren?

Hier sind einige Einsteiger freundliche Projekte

- FP für weitere Programmier Sprachen:
<https://learnfp.org/>
- Mehr über
Haskell:<https://learnyouahaskell.github.io/>
- Falls jemand nicht genug hat: <https://nixos.org/learn/>
- ```
(defun display-message ()
 (let ((message Falls jemand immer noch nicht
 genug hat: https://guix.gnu.org/cookbook/)))
 (display-message)))
```

- <https://achimjungbham.github.io/pub/papers/lambda-calculus.pdf>
- <https://hackmd.io/@9LWWQ7ZlTjKOWMATI1lXgg/HkDzV88Sp>
- [https://en.wikipedia.org/wiki/Lambda\\_calculus](https://en.wikipedia.org/wiki/Lambda_calculus)
- [https://bitsavers.org/pdf/mit/cadr/chinual\\_6thEd\\_Jan84/](https://bitsavers.org/pdf/mit/cadr/chinual_6thEd_Jan84/)
- <https://www.mdc-berlin.de/research/publications/pigx-reproducible-genomics-analysis-pipelines-gnu-guix>

## Literatur

- Mathematics in Programming (Xinyu Liu)
- Functional and Logic Programming (Gerhard Goos, Juris Hartmanis)
- Revised<sup>6</sup> Report on the Algorithmic Language Scheme ()

## Temporary page!

$\text{\LaTeX}$  was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added on the final page this extra page has been added to receive it. If you rerun the document (without altering it) this surplus page will go away, because  $\text{\LaTeX}$  now knows how many pages to expect for this document.