

Funktionale Programmierung

Thanh Viet Nguyen

9. Dezember 2024



**HOCHSCHULE
HANNOVER**
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

- 1 Einleitung
- 2 Herkunft der Funktionalen Programmierung
- 3 Programmier Paradigmen
- 4 Historie
- 5 Features der Funktionalen Programmierung
- 6 Vorteile und Nachteile
- 7 Anwendungsgebiete
- 8 Fazit
- 9 Quellen

- 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

Weise Worte:

- The proper use of comments is to compensate for our failure to express ourselves in code."(Robert C. Martin, Functional Programming in Java, Vorwort)
- "In programming the hard part isn't solving problems, but deciding what problems to solve."(Paul Graham, Functional Programming in Java, Vorwort)
- Testing by itself does not improve software quality. Test results are an indicator of quality, but in and of themselves, they don't improve it. Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often."(Steve McConnell, Functional Programming in Java, Vorwort)

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

Lambda Kalkül

- kleinste Universelle Programmiersprache, die keine Maschine benötigt
- Turing-Vollständigkeit: Fähigkeit eines Berechnungssystems, jede berechenbare Funktion auszuführen, die eine Turing-Maschine ausführen kann, und ist ein Maß für die Ausdruckskraft von Programmiersprachen und anderen Berechnungssystemen.

$$\langle \text{expression} \rangle := \langle \text{name} \rangle \mid \langle \text{function} \rangle \mid \langle \text{application} \rangle$$
$$\langle \text{function} \rangle := \lambda \langle \text{name} \rangle . \langle \text{expression} \rangle$$
$$\langle \text{application} \rangle := \langle \text{expression} \rangle \langle \text{expression} \rangle$$

Typen Theorie

Formales System zur Typklassifizierung zur Vermeidung von Widersprüchen, hierarchisch strukturiert, mit Anwendungen in Verifikation, Logik und Grundlagenforschung; konstruktivistisch und mit intuitionistischer Logik verbunden.

- Werte durch Typen klassifiziert, die definieren, welche Art von Daten ein Wert repräsentieren kann.
- Statische Typisierung: Typen werden zur Compile-Zeit überprüft, was bedeutet, dass viele Fehler frühzeitig erkannt werden können.
- Dynamische Typisierung: Typen werden zur Laufzeit überprüft, was mehr Flexibilität bietet, aber auch zu Laufzeitfehlern führen kann.

Kategorie Theorie

- Kategorien: Objekte (Typen) und Morphismen (Funktionen).
- Funktoren: Abbildungen zwischen Kategorien, die Struktur bewahren.
- Monaden: Spezielle Funktoren für Nebenwirkungen und sequentielle Berechnungen.
- Natürliche Transformationen: Verbindung zwischen zwei Funktoren.
- Typklassen: Polymorphe Programmierung, inspiriert von Kategorien.
- Algebraische Datentypen: Datentypen als Objekte in der Kategorientheorie.

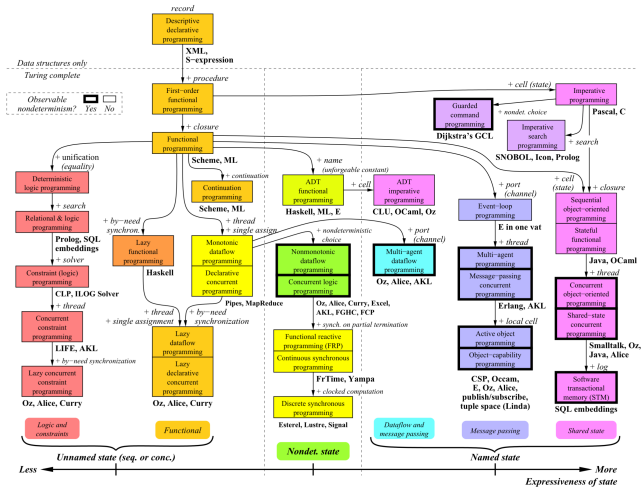
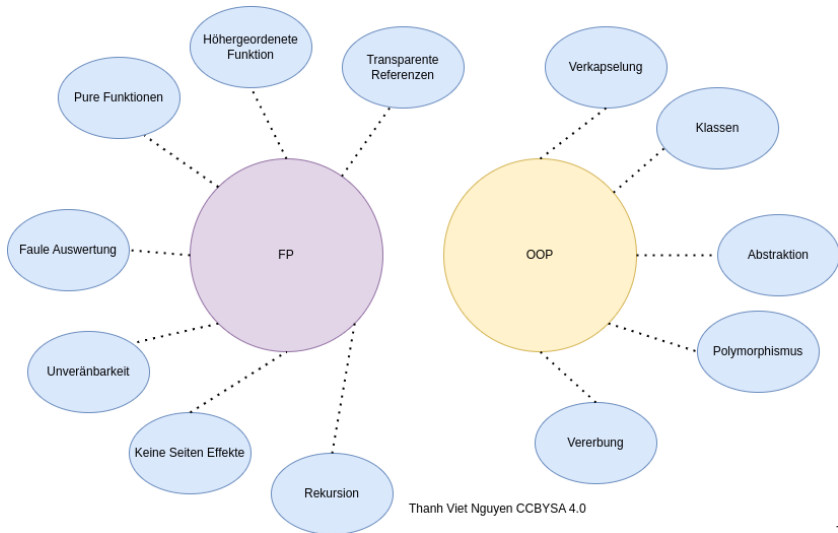


Figure 2. Taxonomy of programming paradigms

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

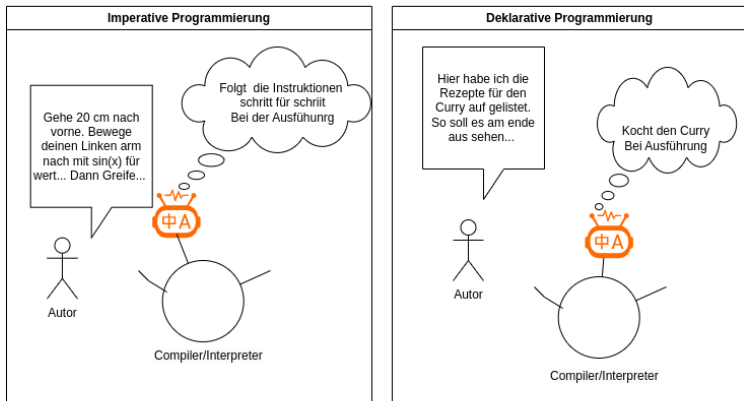
Objekt Orientierung und Funktionale Programmierung



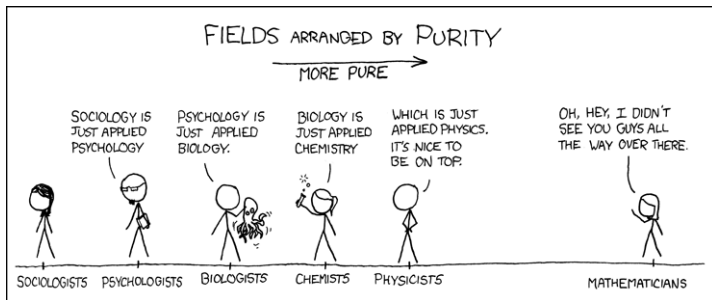
- " Object oriented programming makes code understandable by encapsulating moving parts. Functional programming makes code understandable by minimizing moving parts."
(Michael Feathers, Functional Programming in Java, Vorwort)

Deklarative und Imperativen Programmierung

Vereinfachte Veranschaulichung



Thanh Viet Nguyen CCBYSA 4.0



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

Lisp Maschinen

- 1960er
- Hoffnung an Digitales Computing für Mathematische Problemlösungen
- Erforschung Künstlicher Intelligenz



https://en.wikipedia.org/wiki/Space-cadet_keyboard

https://en.wikipedia.org/wiki/Lisp_machine

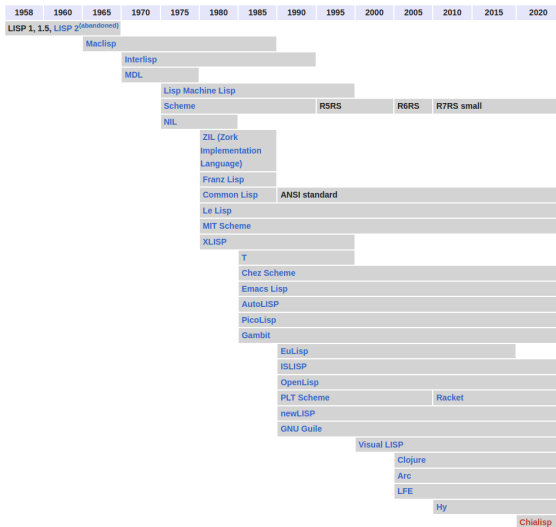


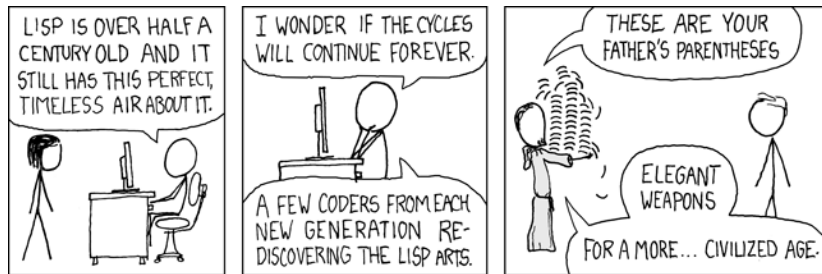
```
(print "Ich bin nicht alt, sondern nachhaltig")  
(define fact  
  (lambda (n)  
    (if (= n 0) ; Basisfall  
        #;(= n 1)  
        1 ; Identitaet von *  
        (* n (fact (- n 1))))))
```

https:

[//en.wikipedia.org/wiki/Lisp_\(programming_language\)](https://en.wikipedia.org/wiki/Lisp_(programming_language))

Lisp-Dialekte





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

- Purity: Eine Funktion gibt immer das gleiche Ergebnis für die gleichen Eingaben zurück und verändert keine externen Zustände.
- First-Class-Funktionen: Funktionen können wie Daten behandelt werden, d.h. sie können als Argumente übergeben, zurückgegeben und in Variablen gespeichert werden.
- Unveränderlichkeit: Datenstrukturen sind unveränderlich, d.h., dass sie nicht verändert werden können, nachdem sie erstellt wurden.
- Wenn eine Funktion pur ist, folgt: Kein State -> keine Seiteneffekte
- Basiert sich auf mathematische Aussagen
- Referenzen sind Transparent: input -> output

Konzepte von FP part 1/2

- Rekursion: Funktion, die sich selbst aufruft.
- Höhere geordnete Funktionen: Rufen andere Funktionen als Argumente zu übergeben oder als Ergebnisse zurückzugeben.
- Currying: Umwandlung von Funktionen, die mehrere Argumente annehmen, in eine Kette von Funktionen, die jeweils ein Argument akzeptieren.
- Functor:
 - Functor: Funktion mit mehreren Argumenten in eine Kette von Funktionen umgewandelt wird, die jeweils ein Argument akzeptieren, was die teilweise Anwendung von Argumenten ermöglicht.
 - wird über die Funktion "fmap" Bereitgestellt.
 - Befolgt Identitätsregel und Kompositionsregel.

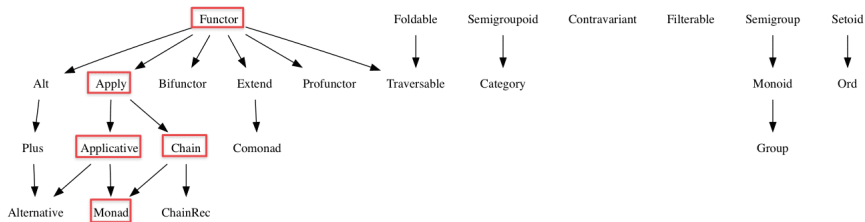
Konzepte von FP part 2/2

■ Applikat:

- Erweiterung des Funktors: Funktionen, die in einem Kontext sind, auf Werte in einem anderen Kontext anzuwenden.
- Verwendet Funktionen "pure" oder "return", um Werte in den Kontext zu bringen
- Befolgt Identitäts- und Assoziativitätsregeln.
- Heißt Applikation in der Mathematik: Anwendung einer Funktion auf ihre Argumente, wobei das Ergebnis der Funktion zurückgegeben wird.

■ Monad:

- Spezielle art des Funktors: Bietet zusätzlich Möglichkeit, Werte in einem Kontext zu verketteten.
- Abstraktion um mit Effekten wie Zustandsänderungen, Ein- und Ausgabe oder Fehlern umzugehen
- Befolgt Identitätsregel und Assoziativitätsregel.



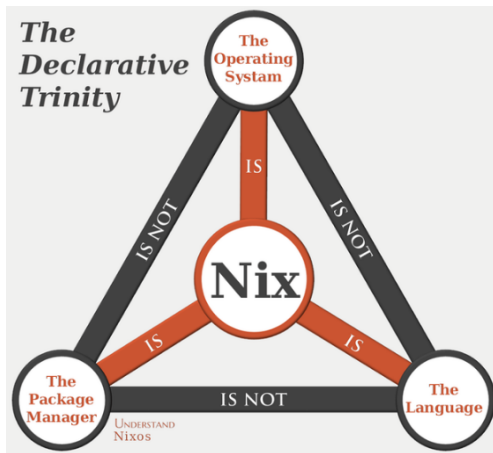
<https://yabai.tw/category/monad/overview/>

Vorteile	Nachteile
Mehr Sicherheit beim Entwickeln: Nebenläufigkeit, Wartbarkeit und Testbarkeit	keine Zustandskontrolle
Parallele Verarbeitung	meist speicherintensiver bei der Laufzeit
Code ist in der Regel lesbarer und ordentlicher	Lesbarkeit \neq Verständlichkeit
Modularität	
Höhere Abstraktion	

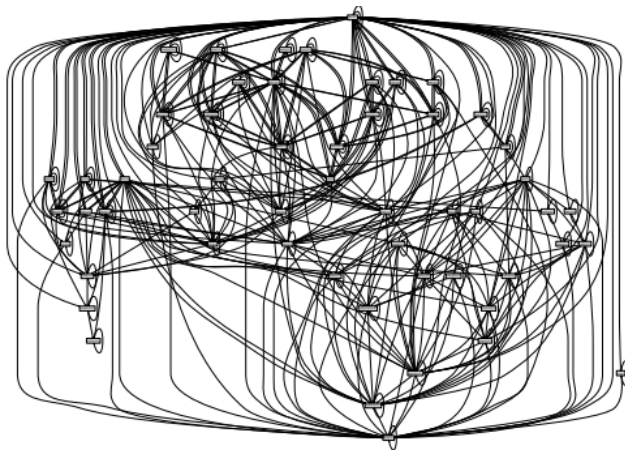
- Software Anwendungen
- Systemkonfigurationen
- Beweisasistent für Mathematische Theoreme sowie Automatische Beweisführung
 - Coq/Rocq für Lisp-Dialekt-Liebhaber
 - Lean für nicht Lisp-Dialekt-Liebhaber
- Verarbeitung von Daten

Deklarative Systemkonfigurationen und Reproduzierbare Bauten

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

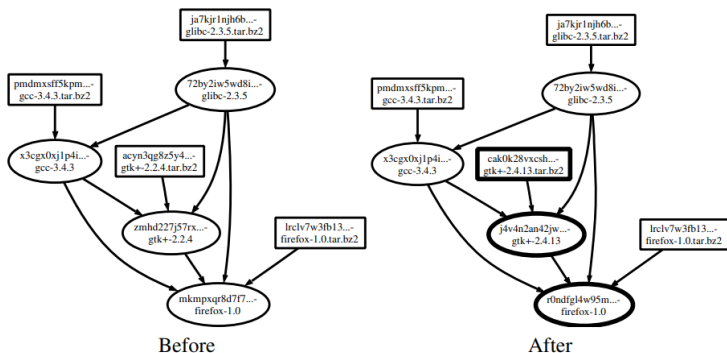


<https://github.com/gytis-ivaskevicius/high-quality-nix-content/tree/master/memes>



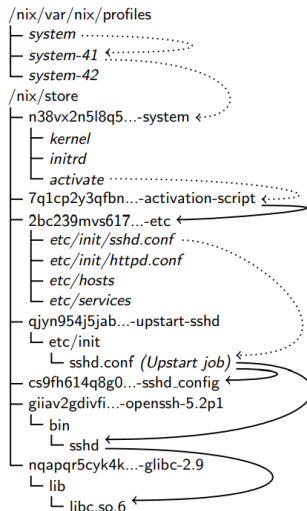
Abhängigkeitsgraph von Firefox

ThePurelyFunctionalSoftwareDeploymentModel, Eelco Dolstra, Seite 10

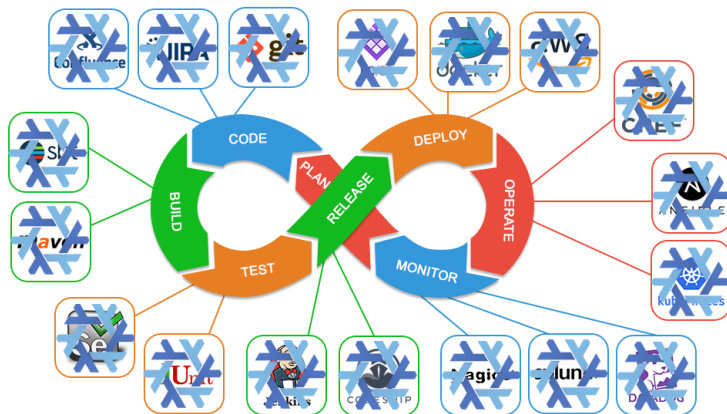


Firefox in Nix

ThePurelyFunctionalSoftwareDeploymentModel, Eelco Dolstra, Seite 22



Altes Beispiel von OpenSSH-Daemon



<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

- Hier zu der Rekursionspräsentation
- Mehr über Lisp-Dialekt: <https://docs.racket-lang.org/>
- Mehr über Haskell: <https://learnyouahaskell.github.io/>
- FP für weitere Programmier Sprachen: <https://learnfp.org/>
- 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)))
```

## Literatur

- Mathematics in Programming (Xinyu Liu)
- Functional and Logic Programming (Gerhard Goos, Juris Hartmanis)
- Revised<sup>6</sup> Report on the Algorithmic Language Scheme (Robert Bruce Findler, Jacob Matthews)
- Functional Programming in Java (Pierre-Yves Saumont)
- The Design and Implementation of Programming Languages (John Hughes)
- A short introduction to the Lambda Calculus (Achim Jung)
- Lambda Calculus (Tobias Nipkow)
- Functional Programming with Overloading and Higher-Order Polymorphism (Mark P. Jones)
- A Tutorial Introduction to the Lambda Calculus (Raúl Rojas)
- Mathematical Components (Assia Mahboubi and Enrico Tassi)
- NixOS: A Purely Functional Linux Distribution (Eelco Dolstra)
- The Purely Functional Software Deployment Model (Eelco Dolstra)

## Weblinks

- <https://yabai.tw/category/monad/overview/>
- [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>
- <https://docs.racket-lang.org/>
- <https://learnyouahaskell.github.io/>
- [https://rand.cs.uchicago.edu/cufp\\_2015/](https://rand.cs.uchicago.edu/cufp_2015/)
- [https://www.adit.io/posts/2013-04-17-functors,\\_applicatives,\\_and\\_](https://www.adit.io/posts/2013-04-17-functors,_applicatives,_and_)



# Danke für die Aufmerksamkeit



<https://github.com/SpiXFamily/FP-Beginners-Presi>