Entwurfsmuster in dynamischen Sprachen

Ein vergleich von Java, Python und Clojure

Nick Zbinden und Michael Sprecher

9. November 2010

Warum und Wie

Wer?	Wie?
Sprecher, Michael	Python
Zbinden, Nick	Clojure

Warum:

Um zu zeigen, dass es noch etwas anderes gibt als statisch typisierte objektorientiert Sprachen.

Vorlage: Java

- limitier statisch typisiert (weakly)
- classenbasiertes OO
- kaum support für FP

Wie und Warum

Michael: Python

- dynamisch typisiert (strongly)
- classenbasiertes OO
- limiter support für FP

Nick: Clojure

- dynamisch typisiert (strongly)
- OO durch multimethods und records
- starker support für FP

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- Closure.
  - Closure können wie jede andere Variabel behandelt werden.
  - können in eine Variabeln gespeichert werden.
  - diese Variabeln können an Funktionen übergeben werden.
  - sie können anonym erstellt werden (wie string z.B. änonymer string"
- pure functions
  - funktionen ohne seiteneffekte
  - Warum?
    - einfache testen und debuggen
    - einfacher nach zu vollziehen
    - kann vom Compiler besser obtimiert werden
    - können einfach parallelisiert werden

Functional Programming

- no mutable globale state
  - Das geht natürlich nicht immer! Aber das Ziel sehr gross geschrieben!
- immutable Data-Structures
  - Bei jeder Änderung kommt eine neue Daten zurück.

Functional Programming

- no mutable globale state
  - Das geht natürlich nicht immer! Aber das Ziel sehr gross geschrieben!
- immutable Data-Structures
  - Bei jeder Änderung kommt eine neue Daten zurück.

Functional Programming

- no mutable globale state
  - Das geht natürlich nicht immer! Aber das Ziel sehr gross geschrieben!
- immutable Data-Structures
  - Bei jeder Änderung kommt eine neue Daten zurück.

Functional Programming

- no mutable globale state
  - Das geht natürlich nicht immer! Aber das Ziel sehr gross geschrieben!
- immutable Data-Structures
  - Bei jeder Änderung kommt eine neue Daten zurück.

Functional Programming

- no mutable globale state
  - Das geht natürlich nicht immer! Aber das Ziel sehr gross geschrieben!
- immutable Data-Structures
  - Bei jeder Änderung kommt eine neue Daten zurück.

Strategie Pattern in Clojure

1. Strategien: anstelle von Objekte benützen wir ganz normale Funktionen
2. Enten: anstelle von benützen wir hash-maps
3. Keine Basisklasse nur Funktionen die Strategien ausführen

Strategie Pattern in Clojure

1. Strategien: anstelle von Objekte benützen wir ganz normale Funktionen
2. Enten: anstelle von benützen wir hash-maps
3. Keine Basisklasse nur Funktionen die Strategien ausführen

Strategie Pattern in Clojure

1. Strategien: anstelle von Objekte benützen wir ganz normale Funktionen
2. Enten: anstelle von benützen wir hash-maps
3. Keine Basisklasse nur Funktionen die Strategien ausführen

Strategie Pattern in Clojure

1. Strategien: anstelle von Objekte benützen wir ganz normale Funktionen
2. Enten: anstelle von benützen wir hash-maps
3. Keine Basisklasse nur Funktionen die Strategien ausführen

Strategien = Funktionen

Strategie Pattern Fazit:

	Java	Python	Clojure
LoC	123	nil	23

Michael:

Python kann dank dynamik einiges gut machen.

Nick:

First Class Functions sind ein muss für jede Programmiersprache.

Clojure Observer

Merke:

Globaler State ist in FP Sprachen (sehr) selten.

- Clojure bittet fantastischen library support.
- First Class Functions werden auch hier forteilhaft eingesetzt.

Observer Fazit:

	Java	Python	Clojure
LoC	128	nil	19

Nick:

Trotz dessen, dass beide Sprachen library support anbieten kann Clojure mit ein mehr als 6 mal kürzeren Code das gleich erreichen.

Decorater Fazit:

	Java	Python	Clojure
LoC	100	nil	24

Nick:

Clojure erlaubtes einen völlig generischen ansatz zu verwenden der Daten und Logik trennt. Diese erlaubt es die funktionen 1:1 weiter zu verwenden.