Lernzettel

Prinzipien der funktionalen Programmierung: Reinheit, Referentielle Transparenz und unveränderliche Daten.

Universität:	Technische Universität Berlin
Kurs/Modul:	Programmieren II für Wirtschaftsinformatiker
Erstellungsdatum:	September 6, 2025

Lernzettel: Prinzipien der funktionalen Programmierung

(1) Reinheit. Reinheit bedeutet, dass eine Funktion keine Nebenwirkungen hat und nur von ihren Eingaben abhängt. Eine rein funktionale Funktion liefert bei exakt gleichen Argumenten immer denselben Ausgabewert. Der Funktionsaufruf verändert kein Programmverhalten außerhalb der Rückgabe.

Beispiel (rein): `def square(a: Int): Int = a * a`

```
val s = square(5)
```

Folgerungen der Reinheit:

- Leichtere Reasoning und Testing, da Funktionen keine versteckten Zustände nutzen.
- Einfachere Parallelisierung, weil keine gemeinsamen Mutationen auftreten.

Abgrenzung: Nebenwirkungen sind z. B. IO, Mutationen von globalen Variablen oder zufällige Werte, die das Ergebnis beeinflussen.

(2) Referentielle Transparenz. Eine Expression ist referentiell transparent, wenn sie durch ihren Rechenwert ersetzt werden kann, ohne das Programmverhalten zu verändern. Reine Funktionen erfüllen diese Eigenschaft. Dadurch lassen sich Programme besser optimieren, testen und parallelisieren.

Beispiel (referentielle Transparenz): `def f(x: Int): Int = x + 1`

```
val y = f(3)
```

Hinweis: Da `f` rein ist, kann `f(3)` durch den Wert `4` ersetzt werden, ohne das Verhalten zu ändern.

Auswirkungen auf das Programmieren:

- Referenzersetzungen sind sicher und vorhersehbar.
- Optimierungen (z. B. gemeinsame Auswertung, Caching) sind leichter zu begründen.
- Tests können gezielt einzelne Funktionen prüfen.

(3) Unveränderliche Daten. In der funktionalen Programmierung werden Datenstrukturen typischerweise unveränderlich erstellt. Anstatt Werte zu mutieren, erzeugt man neue Werte (mit strukturellem Teilen).

Beispiele in Scala:

- `val nums = List(1, 2, 3)` – unveränderlich gegebene Liste
- `val nums2 = 0 +: nums` – fügt 0 am Anfang hinzu, erzeugt eine neue Liste
- `val nums3 = nums :+ 4` – fügt 4 am Ende hinzu, erzeugt eine neue Liste

Wichtige Konzepte:

- Binding mit `val` ist unveränderlich; `var` wäre mutable.
- Vorteile der Unveränderlichkeit: einfache Reasoning, sichere Parallelität, weniger Bugs durch unbeabsichtigte Mutationen.

Zusammenfassung (Impuls): Reinheit, referentielle Transparenz und unveränderliche Daten bilden das Fundament der funktionalen Programmierung. Sie ermöglichen besseres Verstehen, robustes Testing und sichere Parallelität – zentrale Bausteine auch in Scala und in der Wirtschaftsinformatik, z. B. bei datengetriebenen Fragestellungen.